Summary of "Goal-conditioned Reinforcement Learning with Transformer Architectures and Intrinsic Reward Design"

June 7, 2023

1 Summary

The study focused on the application of transformer architectures in Goal-conditioned Reinforcement Learning (GCRL), within both online and offline RL settings. An intrinsic reward mechanism for training a Deep Reinforcement Learning (DRL) agent was also proposed.

The investigation began with the application of a transformer-based architecture in the online supervised RL setting, utilizing the recent Goal-Conditioned Supervised Learning (GCSL) as a starting point.

The study was driven by the recognized capabilities of transformers in managing sequence dependencies, a recurring issue in RL. Transformers, originally designed for natural language processing tasks, were incorporated into the GCRL domain with the objective of identifying performant configurations of these architectures.

The initial focus was on adjusting the transformer architecture to the specifications of the GCSL framework, navigating the distinctive challenge of the online RL setting. The intriguing setting of GCSL, where the model is not directly optimized to maximize a reward correlated with reaching goals, called for goal-reaching behavior to emerge as a result of maximum-likelihood estimation with the appropriate architectural modifications and training setup. The intention was to search for modifications to the existing decoder-only transformer within the GCSL setting, and explore whether these changes led to improvements in the offline supervised RL setting.

Incorporating the transformer architecture into the GCSL model required a series of modifications. A crucial discovery was the necessity for a delicate balance between optimization steps and data collection frequency to prevent model collapse. Performance enhancement beyond the baseline was achieved through the use of variance and covariance regularization and a gated fusion mechanism for merging goal information with the sequence of state and action tokens.

Interestingly, modifications that proved effective in the online RL scenario did not universally guarantee improvements in the offline setting. This was assessed within the context of the existing Decision Transformer, designed for the offline supervised RL setting. Stable learning and slight performance improvements were consistently contributed only by the gated fusion mechanism.

A shift towards a more discrete representation of the goal was noted through the use of the gated fusion mechanism. A single reward token was used as the goal, in contrast to feeding returns-to-go at each timestep. This approach provided a greatly simplified goal specification.

A new method for GCRL was proposed, involving training a goal-conditioned dynamics model along with a conventional DRL agent. The agent was trained through an intrinsic reward, achieved by maximizing the negative squared L2 norm between the actual next state's representation and the representation given by the goal-conditioned dynamics model. This method showed promising results in a simple 10x10 gridworld environment, outperforming the discrete Proximal Policy Optimization (PPO) baseline. However, extending these results to more complex environments such as the CarRacing environment was not possible.

The findings laid a foundation for several potential future research paths. One avenue is to evaluate the scalability and performance of the modified Decision Transformer architecture on larger offline datasets, such as the multi-task Atari dataset.

Further exploration and extension of the novel GCRL approach in more complex environments, such as the CarRacing environment, could provide valuable insights.

Finally, inspired by the work of Dehaene et al. [1], computational forms of consciousness were suggested for exploration to address perceived gaps in intent and alignment in current RL models. Approaches that generate detailed future expectations or align with the Global Workspace Theory were of particular interest. This theory involves the selection, processing, and system-wide broadcasting of vital information to guide behavior, potentially offering a more granular approach to environment navigation. This ambitious direction poses challenges but could provide exciting avenues for future research and the development of consciously aligned RL models.

References

1. Dehaene, S., Lau, H. & Kouider, S. in *Robotics, AI, and Humanity* 43–56 (Springer, 2021). https://doi.org/10.1007/978-3-030-54173-6_4.

Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East, Denmark
www.cs.aau.dk

Title

Goal-conditioned Reinforcement Learning with Transformer Architectures and Intrinsic Reward Design

Project Type

10th Semester Project: Master Thesis in Computer Science

Project Period

Spring 2023

Project Group

CS-23-MI-10-05

Authors

Milad Samim

Supervisors

Christian Schilling
Kim G. Larsen

Number of pages: 24
Date of completion: June 7, 2023

Nomenclature

DL	Deep Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DT	Decision Transformer
FCN	Fully Connected Network
FNN	Feedforward Neural Network
GCRL	Goal-conditioned Reinforcement Learning
GCSL	Goal-conditioned Supervised Learning
MDP	Markov Decision Process
RL	Reinforcement Learning
SL	Supervised Learning
SSL	Self-supervised Learning
VICReg	Variance Invariance Covariance Regularization

Mathematical Notation

x	Scalar (integer or real)
$\mathbf{x}$	Vector
X	Matrix
$\mathcal{X}$	Set
$\nabla f(x;\boldsymbol{\theta})$	The partial derivatives of $f(x;\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$

Goal-conditioned Reinforcement Learning with Transformer Architectures and Intrinsic Reward Design

Milad Samim

Abstract

This report delves into Goal-conditioned Reinforcement Learning (GCRL), primarily exploring the use of transformer-based architectures and an additional a proposal of simple intrinsic reward mechanism. GCRL holds the potential of enabling more data-driven training of Reinforcement Learning (RL) agents, which are capable of solving a diverse set of goals. The exploration of transformer-based architectures in this setting attempts to further the insights into adapting highly scalable architectures in GCRL.

The work proceeds in three main arcs. Firstly, we extend the recent Goal-conditioned Supervised Learning (GCSL) framework with a highly scalable decoder-only transformer architecture akin to the Decision Transformer. GCSL is an atypical supervised RL setting in that no expert demonstrations are provided for the agent but instead requires the goal-reaching behavior to emerge from maximum likelihood estimation. Here we show that careful modifications are required to avoid model collapse and yield performance beyond the baseline.

The second part aims to evaluate whether the modifications in the GCSL setting will transfer to the more classical offline setting of supervised RL. In particular, we evaluate the proposed modifications on the Decision Transformer, which is targeted the offline supervised RL setting. Through this work, we show that in general transference between the setting of online and offline supervised RL is limited. Nonetheless, we find that the use of a gated fusion mechanism offers simplified goal-conditioning, with similar or higher performance than the baseline.

The third part proposes an intrinsic reward mechanism through the discrepancy between the next state and a predicted next state to train a GCRL agent. We show that the proposed approach learns to solve a simple goal-reaching gridworld faster than a baseline Proximal Policy Optimization (PPO) agent.

Through the study, we show the difficulties and provide insights into adopting a transformer architecture in the setting of online supervised RL. Further, we show that there is a limited transfer between architectural configurations between the online and offline settings, whilst reaching a simplified goal-conditioning method for the Decision Transformer. Finally, the proposed intrinsic reward shows promising results in a simple gridworld environment but fell short in more complex environments.

Keywords: Goal-conditioned Reinforcement Learning, Deep Learning, Offline Learning, Online Learning

1. Introduction

Since the seminal introduction of the Deep Q-Network (DQN) by Mnih et al. [1], the field of deep reinforcement learning (DRL) has made significant progress. This evolution is evidenced by the successful application of deep learning techniques in managing complex decision-making problems with extensive state and action spaces. Some notable instances of these advancements include achieving superhuman performance in the game of Go by AlphaGo from Silver et al. [2], and defeating professional StarCraft players as accomplished by Vinyals et al. [3].

1.1. Difficulties with Reward Functions

Even though impressive achievements have been reached with DRL, limitations still exist. In particular with the design of reward functions. Reward functions are specified either in a sparse or dense approach. In the dense setting, a reward signal is given to the agent for every action it takes, whereas the sparse reward offers a binary reward denoting the success of the task upon completion [4]. Dense reward functions are preferred, as they give rise to more efficient learning. The explanation for this is straightforward, methods like DQN, Proximal Policy Optimization (PPO) [5], Soft Actor Critic (SAC) [6], and similar methods rely on randomness to ensure sufficient exploration. When combined with complex environments, numerous explorative actions must be taken before the agent receives sufficient feedback signals to discriminate between the actions to take, particularly in a sparse reward setting.

Unfortunately, dense reward functions are not themselves without problems, as they are subject to reward hacking [7], which results in the agent optimizing the reward function by learning undesired behaviors. A well-known example of this is the dust cleaning robot [8], which is rewarded for dust collection. Rather, than learning to clean as expected the robot will learn to initially collect some amount of dust followed by repeatedly dumping and collecting the same dust. Thus, it is not guaranteed that reward functions will give rise to behavior that is aligned with the actual human intent [9]. Further, when

Department of Computer Science, Aalborg University, Denmark *June 7, 2023*

moving beyond simulated environments with given reward functions, the difficulty of administering a feedback signal for every action is nontrivial. The difficulty of designing a dense reward function in complex settings is the primary motivation for sparse reward signals.

Many works have aimed to address the challenges of reward functions from different angles, such as manually designed curriculum [10, 11], where tasks are presented to the agent based on a difficulty ordering, which allows the agent to conquer simpler tasks, allowing it to benefit from the skills obtained when exposed to tougher tasks. Another approach is reward shaping [12, 13], which relies on augmenting a sparse reward signal such that more frequent feedback information is emitted, however, this would typically require domain-specific insights. Yet another line of work aims to use forms of intrinsic rewards [14, 15], a particular example here is [14], which uses the uncertainty of the next state prediction as an intrinsic reward to encourage exploration. As such the intrinsic reward does not stand alone, but is still reliant on an extrinsic reward.

Another area of limitations is that current DRL agents typically learn to master a specific task very well, but learning new tasks requires new strategies or entirely new reward functions to specify the tasks [16].

Goal-conditioned Reinforcement Learning (GCRL) presents an approach to tackle the inherent complexities involved in crafting reward functions. In this paradigm, explicit goals become integral to the policy, serving as a beacon to guide the agent's actions toward intended outcomes. This method potentially makes the learning process more streamlined by providing the agent with a focused direction. Reward functions still need to be specified, but are conditional on the goal, thus when using deep neural networks to learn goal-conditioned policies, there is potential for generalization and performance transfer between solving various goals. Typically in GCRL reward functions are specified as sparse indicator functions detecting whether the goal is achieved [17], minimizing the focus on handcrafting reward function. Instead, the challenge of specifying reward functions is traded for the challenge of enabling a capacity of acting in goal-directed ways. Additionally, GCRL has enabled supervised approaches [18, 19, 20, 21] to exceed in RL tasks, opening up a new direction to train agents without the need to specify reward functions.

Spurred by the potential of GCRL, this work aims to delve further into this area, extending upon Levine's [16] insights, which motivate GCRL as an approach to obtain more data-driven RL algorithms. The specific contributions are:

- Extending the existing work of Goal-conditioned Supervised Learning (GCSL) [18], by using a more scalable architecture in the form of the transformer architecture [22] instead of an ordinary neural network architecture.
- Augmenting the decision transformer [20] by using a cross-attention gating [23] approach to embed the reward conditioning into the transformer architecture.
- An approach of utilizing the difference between the imagined and actual next-state latent representations as an intrinsic reward. Technically this involves training a dynamics model to predict the next-state latent representation, conditioning it on the goal rather than action information. Subsequently, an RL agent is trained to minimize the difference between the imagined and actual next-state latent representations as its reward function. The potential of the approach is to replace hand-engineered reward functions with a general task-agnostic reward function.

The following section frames and motivates the proposals of this work and how they are connected.

2. Background and Motivation

Initially, the section will take off-set in the work [16] by Levine, which proposes directions to address existing limitations of current Reinforcement Learning (RL) approaches. Following this, the notions of learning from offline data and self-supervised RL and their connection to the two extended works (GCSL, decision transformer) will be presented. Afterward, a direction distinct from those proposed in [16] will be considered to reflect, that there might be more fundamental gaps to close, besides the more technical directions proposed in [24]. The remaining subsections will motivate the three proposals of this work and present the outcomes.

2.1. Addressing the Limitations: A Focus on Self-supervision and Offline Data

The current limitations beg the question of, what could be needed to move toward agents that can learn to act in complex environments in diverse and goal-directed ways.

To address these questions Levine [16] puts forth a set of suggestions to address this challenge. In [16], Levine suggests that a methodology for achieving this could be developed from current reinforcement learning algorithms for learning-based control. Further, it is contended that such agents necessitate a causal and generalizable comprehension of their surroundings. Moreover, it is emphasized that relying solely on human-engineered reward functions will not suffice; rather more data-driven methods that can efficiently learn from offline data should be incorporated. To tackle these constraints, [16] advocates for the adoption of self-supervised reinforcement learning techniques and the utilization of offline data, enabling agents to learn from vast amounts of pre-existing information and ultimately enhancing their adaptability and understanding of various environments.

2.2. Learning from offline data

The ability to learn from offline data, that is data, that is not collected by the agent itself is known as offline RL, as opposed to conventional online RL, where the agent learns from its interaction in the environment. The aim of offline RL is not that of behavioral cloning, but to learn from the static data in order to perform beyond the data [24]. Offline RL does not come without challenges and a particular one is the distributional shift between the data experienced in training, and how the environment will behave when behaving differently from the static data [24]. Nonetheless, recent strides in offline RL have been made [25, 26, 20]. A particular work here is the decision transformer by Chen et al [20], which proposes to model RL tasks as a sequence modeling task and use a transformer deep neural network architecture [22]. The use of the transformer architecture is in itself highly attractive, as it is proven to be scalable to billions of parameters and trillions of data points [27, 28, 29]. Further, the transformer architecture has been used across multiple settings and modalities ranging from text, images, and robotics [29, 30, 31, 32]. The inherent scalability of the decision transformer is a contrast to conventional approaches to RL, which has proven more tricky to scale up, albeit doable as shown in [33].

2.3. Self-supervised Reinforcement Learning

Self-supervised learning (SSL) has been a cornerstone of the recent successes of various models, including large language models [29] and image generation models [34]. In the context of reinforcement learning, Levine [16] suggests that one way to frame a self-supervised RL objective is as the problem of achieving a goal, which leads to the formulation of goal-conditioned policies.

Hindsight Experience Replay (HER) [35], a technique used in GCRL, capitalizes on the concept of reassigning reached states as the agent's intended goals. The approach of HER streamlines the learning process for achieving analogous goals in the future. In this vein, the paper, "Learning to Reach Goals via Iterated Supervised Learning" proposes the method goal-conditioned supervised learning (GCSL) [18] demonstrates learning without rewards in a constrained setting where goals are provided as states in the state space, by using the idea of hindsight relabeling. This work highlights the potential of goal-conditioned self-supervised learning to create agents that can solve multiple goals.

2.4. Other Directions to Search

In the pursuit of more capable human-like intelligence, it may be that besides moving to an offline setting, which will allow for vastly larger data sets and moving beyond handcrafted reward functions, that still a qualitative difference remains. As laid out by Dehaene et al. [36] current deep learning models mostly perform unconscious processing (C0), rather than conscious processing (C1 and C2). Dehaene et al. propose that "consciousness" encompasses two distinct information-processing computations in the brain: the selection and global broadcasting of information for flexible use and reporting (C1), and the self-monitoring of these computations, resulting in a subjective sense of certainty or error (C2). Unconscious processing (C0) is characterised as where most of our intelligence lies, as it involves the automatic, unconscious processing of information that guides our actions and decisions. Examples of complex unconscious processing are visual processing, audio processing, and chess-game evaluation, as such it is argued that even the association of reward or motivation falls into the unconscious category. [36]

The emphasis on a computational view of consciousness alludes that it is implementable. In a similar view Bengio in "The Consciousness Prior" [37] proposes in an abstract sense how computations in an information-processing system akin to the Global Workspace Theory (GWT) presented by Baars [38] could be composed. GWT is directly connected with the C2 conscious process of Dehaene et al. [36] The theory posits that conscious thoughts consist of specific information pieces temporarily held in working memory, making them globally accessible to unconscious brain computations. This creates an informational bottleneck, through which consciousness emerges and involves competition between unconscious processes for entry into working memory. In a different approach Lecun in [39] presents an architecture for how an intelligence more like human intelligence could be arranged. The proposal contains multiple components, but concretely with connection to consciousness it hypothesized that the module named "configurator" [1], could have a function assimilable to that of C1 and C2.

Thus, it may not be enough to move to an offline setting to enable a more data-driven approach combined with an SSL training approach to enable universal goal direction, but perhaps a concrete computational framework or module is needed to give rise to more human-like capabilities. However, in this work, the quest for computational forms of consciousness will be omitted, and the concrete direction will be motivated next.

2.5. Direction of work

This work aims to explore the use of transformer-based architectures in GCRL. The motivation for emphasizing the goal conditioning echoes the intuition proposed in [16], that it perhaps is a well-suited formulation of self-supervised RL, which is not necessarily governed by a single extrinsic reward function.

2.5.1. Extending Goal-conditioned Supervised Learning with the transformer architeture

To pursue this direction, the objective is to apply the transformer architecture in the same setting as GCSL [18].

[1] The configurator central component, that takes input from all the other components and "configures" them for the task at hand by sending information back out to the components [39]

The method of GCSL was evaluated in five different online RL goal-reaching environments: 2D navigation, manipulating simulated Sawyer robotic arms to push an object to a goal location, using the same arm for door opening, the classic lunar lander environment, and a dexterous object manipulation task, all are seen in Fig. 1. The environments are observed as vectors describing the states, the goal state in all environments is a particular state in the state space. The model in GCSL is a simple two-layer feedforward neural network, which takes both the current state and goal as input and emits an action.

What is interesting about the results of the GCSL setting is that model succeeds in learning from its random behavior and iteratively improves without a reward function. The capacity to learn from their interaction is not a typical feature of supervised RL methods (decision transformer [20, 21], behavior cloning, RvS [19]), as they typically target the offline setting, and require successful behavior to be prevalent in the data set. Thus, the motivation in this direction is to see if enabling a transformer architecture to be capable of learning from random behavior, will yield insights that can be beneficial in the more conventional offline setting for supervised methods.

Moving to a transformer-based difficult proved difficult, and required adjustments to be made as opposed to how transformers have typically been applied for sequential RL tasks. The adjustments include the use of Variance Invariance Covariance Regularization (VICReg) [40] a method developed for SSL, the use of a target network as proposed in [1], the careful fusion of goal information into the state representations, adapted from a transformer applied on a multi-modal text-image QA task [23], and a training strategy different from the typical transformer training. With the adjustments we show that it is possible to match or exceed the performance of the simpler network used in [18], which already matched or exceeded the benchmark methods PPO [5], TD3+HER [41, 35].

The outcome of this work is insights into the challenges of adapting a more scalable architecture transformer for the online goal-conditioned setting. Previous work has indicated similar challenges when scaling up architectures in the online setting [33]. Additionally, we acknowledge that the setting is constrained, in that the environments are fully observable MDPs and that the goals are a part of the state space. Thus in partially observable MDPs and with more complex goals the overall approach is likely not work. However, it shows that in shorter trajectories of movement where there is a connection between the initial state and the terminal state, it is possible to learn actions connecting the states, without reward but by maximum likelihood estimation.

2.5.2. Extending the Decision Transformer

In the setting of offline GCRL, we investigate whether the insights drawn from adapting the transformer to the online setting without rewards are transferable. Specifically, we aim to modify the decision transformer with the modifications found to work in the previous setting. A primary difference is that in the offline setting, the data set is static and contains in some cases examples of expert-level performance, thus the model is not forced to learn from its own initially random behavior. However, the setting is more unconstrained in that the setting may be any RL task and not just goal-reaching tasks. The goal conditioning in the context of the decision transformer is more precisely stated as reward conditioning, as the model during training learns to predict autoregressively which actions to take conditioned on the reward. Thus during inference, the agent is conditioned or "prompted" to generate trajectories of high reward by feeding it high-reward tokens. The aim of evaluating a modified decision transformer is to test whether, the model will be able to achieve performance beyond the data set, as the adaptions in the previous setting improved iteratively from random performance.

What we find is that there is a gap between the online and offline settings and that the modifications in general do not transfer straightforwardly between the settings, similar observations have been made in earlier work, which have prompted tailored approaches to the offline setting [24, 25]. However, we find that using the gated fusion mechanism to fuse the information of the reward conditioning with the state and action representations yields more stability in some of the evaluated environments, and is on par with the default decision transformer in the remaining.

The remainder of the report will be structured by first presenting the preliminaries in Sec. 3. Afterward, the work of adapting GCSL into a transformer-based architecture will be presented in Sec. 5, followed by the work of modifying the decision transformer in Sec. 6. Following, in Sec. 7 the work of training a GCRL agent through intrinsic reward will be presented. Lastly, there will be a conclusion and discussion of possible future work in Sec. 8.

3. Preliminaries

In the following, an overview of the methods applied will be explored. First, the general foundations of Reinforcement Learning (RL) will be presented. Afterward, a brief presentation of deep learning (DL) will be given, followed by deep reinforcement learning (DRL). Next, the transformer architecture will be presented. At last Variance Invariance Covariance Regularization (VICReg), a method applied in the setting of self-supervised learning will be examined.

3.1. Reinforcement Learning (RL)

The crux of RL is the ability of an agent to learn to maximize a numerical reward signal through interaction with its environment. The classic graphical visualization of RL popularized Sutton et al. [42] is illustrated by Fig. 2

Fig. 2 highlights that the agent interacts with the environment by taking actions A_t for which the environment will emit a reward signal R_{t+1} and transition to the next state S_{t+1}. This method of interaction is the primary method available to the

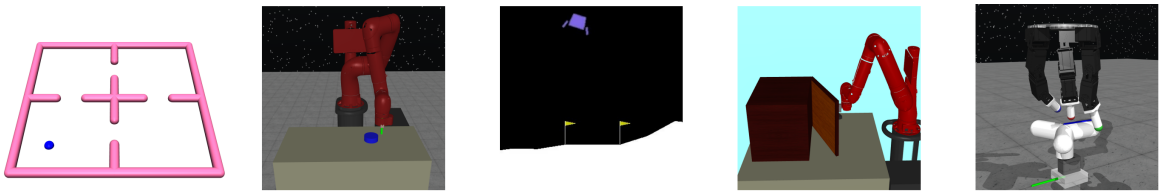


Fig. 1: The environments used in evaluating GCSL. From left to right: 2D navigation, robotic pushing, lunar lander, robotic door opening, dexterous object manipulation [18].

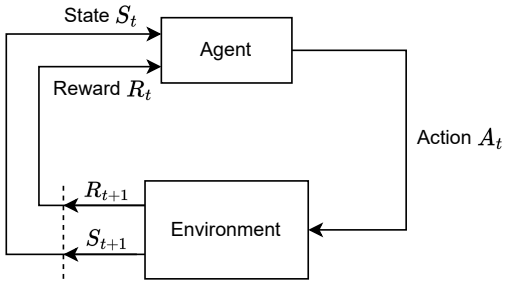


Fig. 2: A high-level overview of the agent-environment interaction, where the environment is modeled as an MDP. The agent in a particular state S_t at time step t takes an action A_t, which affects the environment. As a consequence, the environment will emit a reward R_{t+1} reflecting the reward of the agent's action A_t, and transition to the next state S_{t+1}[42]. Figure and related text is adapted from a previous work [43] (note figure originates from [42]).

agent to infer the value of its actions and the dynamics of the environment. In certain environments for example chess, the dynamics are fixed, and the learning will primarily focus on learning the optimal actions. However in more complex environments, for example maneuvering a robot in the real world the dynamics are unknown, thus the agent may infer only about the dynamics through interaction as well. Further, as embedded into Fig. 2 the mathematical framework of Markov Decision Processes (MDP) provides an approach to formalizing RL.

On the basis of 'Reinforcement Learning: An Introduction' by Sutton and Barto [42] an overview of MDPs and a mathematical formulation of RL will be given next. Some parts of the structure and outline of this section bear similarities to the previous work [43].

3.1.1. Markov Decision Processes (MDPs)

A MDP is a 6-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, \rho)$, where

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, denoting the conditional probability of the next state given the current state and action, with the property that for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, we have $\sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') = 1$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function
- $\gamma \in [0, 1)$ is the discount rate
- $\rho : \mathcal{S} \to [0, 1]$ is the distribution of initial states

Given a MDP $\mathcal{M}$ the aim of the agent is to learn a behavior, that will ensure the maximization of the reward. To formalize this intuition we introduce the idea of a policy. A policy π Eq. (1) is a function denoting the agent's conditional probability of taking a particular action in a particular state, that is $\pi(a_t|s_t) \in [0, 1]$ where $a_t \in \mathcal{A}$, $s_t \in \mathcal{S}$.

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1] \tag{1}$$

Given a concrete policy π we can measure the expected discounted cumulative returns $J(\pi)$ by Eq. (2)

$$J(\pi) = \mathop{\mathbb{E}}_{s_0 \sim \rho, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \right] \tag{2}$$

Now with the ability to discriminate between policies the aim of an agent given a particular MDP $\mathcal{M}$ is to learn an optimal policy π^*, which maximizes the expected discounted cumulative returns as given by Eq. (3)

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3}$$

The formalization of RL into an MDP together with Eq. (3) provides an overview of the task and the aim. However, it is not readily evident how to then endow the agent with the ability to learn π^* from its interaction with the environment. As such the following introduces the notion of state-value functions, Q-functions, and how its formulation as a recursive Bellman equation provides a method to estimate these functions.

3.2. State-value functions and Q-functions

In the following, we will assume that the states and actions of the MDPs are discrete and finite.

The state-value function $v_\pi : \mathcal{S} \to \mathbb{R}$ denotes the expected returns of following the policy π in the state $s \in \mathcal{S}$ and is given by Eq. (4)

$$v_\pi(s) = \mathop{\mathbb{E}}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)} \left[\sum_{t=k}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \mid s_k = s \right] \tag{4}$$

The state-value function expresses the expected returns achievable by following a particular policy at a given state $s_t \in \mathcal{S}$, however, this notion can be partitioned into the following parts. At s_t we follow the policy π giving rise to the expected immediate return $\mathop{\mathbb{E}}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)} [R(s_t, a_t, s_{t+1})]$ and a part representing the discounted expected returns achievable in the next state s_{t+1}, which we can express as the state-value of s_{t+1}. This

gives rise to a recursive formulation as a Bellman equation of the state-value function, which is given by Eq. (5)

$$v_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s),s'\sim\mathcal{T}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma v_\pi(s')\right] \tag{5}$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma v_\pi(s')\right] \tag{6}$$

Given the recursive formulation of the state-value function, it is now possible to envision an algorithm, which could use the agent's interaction with the environment to estimate the state-value function. In particular the following iterative update rule in Eq. (7) ensures convergence.

$$v_{\pi_{k+1}} = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma v_{\pi_k}(s')\right] \tag{7}$$

Theorem 3.1 (Convergence of Iterative Value Function Update)**.** *The sequence of iterative application of the update rule Eq.* (7) $v_{\pi_0}, v_{\pi_1}, v_{\pi_2}, \ldots$ *will converge to the true value function* v_π *given* $\gamma \in [0, 1)$ *[44, 4].*

However, given an estimate for the value of a particular state, does not provide insights for which actions to select.

To formalize the value of taking a particular action in a particular state we use Q-functions, also known as action-value functions. The Q-function $q_\pi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ denotes the expected returns of taking action a in state s, and then acting according to policy π. Concretely Eq. (8) expresses the Q-function making use of the definition of the state-value function from Eq. (4).

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma v_\pi(s')\right] \tag{8}$$

The Bellman equation for Q-functions is given by Eq. (9)

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma \mathop{\mathbb{E}}_{a'\sim\pi(\cdot|s')}\left[q_\pi(s',a')\right]\right] \tag{9}$$

$$= \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,a)\left(\mathcal{R}(s,a,s') + \sum_{a'\in\mathcal{A}} \pi(a'|s') q_\pi(s',a')\right) \tag{10}$$

Now with the Q-function q_π, the agent will be able to evaluate the Q-function for every action $a \in \mathcal{A}$ and then select the action, which maximizes the expected return. However, the Bellman equations for the Q-function and state-value function merely provide an approach to approximate the function values for a particular policy π and not necessarily π^*.

3.2.1. Learning Policies

In the following, we will assume deterministic policies, and write $a = \pi(s)$ to denote the action a selected in state s. We still assume finite state and action spaces.

To discriminate between policies, we will define a partial ordering over the policies:

Definition 1 (Partial Order over Policies)**.**
Let Π *be the set of all policies.*
Then the binary relation $\leq \subseteq \Pi \times \Pi$ *defined as* $\pi \leq \pi' \iff \forall s \in \mathcal{S}.\ v_\pi(s) \leq v_{\pi'}(s)$ *for* $\pi, \pi' \in \Pi$ *is a partial order.*

To find the optimal policy it suffices to repeatedly improve the current policy. To motivate the intuition, suppose an agent has a current policy π, which is not optimal. Now suppose there exists a policy π', which is similar to π at all but one state $s \in \mathcal{S}$, such that $\pi(s) \neq \pi'(s)$ and $v_\pi(s) \leq q_\pi(s, \pi'(s))$. That is the modified policy π' selects an action in state s, which ensures at least the expected returns of the previous policy. This intuition can be extended to show, that the state-value function of the modified policy $v_{\pi'}$ will be at least as good as the previous v_π at all states, reaching a result known as the policy improvement theorem.

Theorem 3.2 (Policy Improvement Theorem)**.**

For $\pi, \pi' \in \Pi$ *if* $\forall s \in \mathcal{S}.\ v_\pi(s) \leq q_\pi(s, \pi'(s)) \implies \forall s \in \mathcal{S}.\ v_\pi(s) \leq v_{\pi'}(s)$*, thus* $\pi \leq \pi'$*. That is* π' *is an improved policy over* π *[42].*

As such selecting an action in a particular state, which ensures that the expected returns are better than the previously expected returns as computed by the state-value function results in policy improvements. This can be generalized into the idea of a greedy policy π', which considers such improvement for any state. The greedy policy is given by (11).

$$\pi'(s) = \arg\max_{a\in\mathcal{A}} q_\pi(s,a) \tag{11}$$

$$= \arg\max_{a\in\mathcal{A}} \left(\mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma v_\pi(s')\right]\right) \tag{12}$$

By construction, the greedy policy in Eq. (11) satisfies the premise of the policy improvement theorem, as such guaranteeing that the policy can only be improved or remain stationary. Specifically, at a certain point, the greedy policy will no longer constitute an improvement of the previous policy, but a fixed point will be reached, such that $v_\pi = v_{\pi'}$ as expressed in Eq. (13).

$$v_{\pi'}(s) = \max_{a\in\mathcal{A}} \left(\gamma \mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + v_{\pi'}(s')\right]\right) \tag{13}$$

Eq. (13) is also known as the Bellman optimality equation for state-value functions and expresses that at any point along a trajectory the action, which maximizes the expected return is chosen. That is upon convergence the reached policy is optimal. In the context of finite state and action spaces a finite amount of modifications can be made to any policy, and coupled with the non-decreasing improvements of the greedy policy, the process of policy iteration will converge as shown in [4]. However, it remains, to show the uniqueness of the Bellman optimality equation.

Theorem 3.3 (Uniqueness of the Bellman Optimality Equation)**.** *For any state-value functions* v_{π_i}, v_{π_j} *if* v_{π_i}, v_{π_j} *satisfies the Bellman optimality equation Eq.* (13) *then* $\forall s \in \mathcal{S}.\ v_{\pi_i}(s) = v_{\pi_j}(s)$*. [44, 4]*

From Theorem 3.1, Theorem 3.2 and Theorem 3.3 an algorithm for acquiring optimal policies known as policy iteration can be devised. The algorithm starts initially with a random policy, and thereafter it applies the value iteration update rule to obtain the respective state-value function. Afterward, the policy is improved by obtaining the greedy policy. This procedure of policy improvement $\rightarrow$ policy evaluation $\rightarrow$ policy improvement $\rightarrow$ policy evaluation $\ldots$ is iterated until the greedy policy converges to the optimal policy.

However, the exact policy iteration procedure requires the state and action spaces to be finite along with the transition dynamics to be known. In many scenarios, these assumptions do not hold, and methods that generalize to larger state spaces and action spaces are required. Further, in many real-world settings, the transition dynamics are not known. To apply the ideas of RL in those settings, function approximation is required, in particular through deep neural networks.

3.3. Deep Learning (DL)

Deep learning refers to the use of artificial neural networks stacked in multiple layers. The word "deep" refers to the fact that multiple layers are used. Artificial neural networks have their origin in biological neural networks, and early implementations were intended to be computational models of how learning could occur in the brain [45, 46].

Deep artificial neural networks can be described as a composition of nonlinear functions, where each function can be viewed as constituting a layer. In particular, a single layer neural network can be expressed as the function $f^1 : \mathbb{R}^N \rightarrow \mathbb{R}^{H_1}$, typically $f^1(\mathbf{x}) = \sigma_1(W_1^\top \mathbf{x} + B_1)$, where $W_1 \in \mathbb{R}^{N \times H_1}$ is the weight matrix denoting and $B_1 \in \mathbb{R}_1^H$ is the bias, $\mathbf{x} \in \mathbb{R}^N$ is the input, and σ_1 is a nonlinear activation function. Here the weight matrix and the bias vector is the parameterization of f^1. From a computational view, we can see that initially an affine transformation is applied to the input followed by a nonlinear function (known as the activation function). An example of an activation function is the Rectified Linear Unit (ReLU) function (14).

$$\text{ReLU}(x) = \max(0, x) \tag{14}$$

To obtain a deep neural network, multiple layers $f^1, f^2, \ldots, f^n$ are composed $f = f^n \circ (f^{n-1} (\circ \ldots (f^2 \circ f^1) \ldots)$. The layers which are not the first or last layers that is the layers $2, \ldots, n-1$ are known as hidden layers. The first and last layer is also known as the input layer and the output layer respectively. In practice, each layer can be more complex than affine transformations composed with a nonlinear function. For example, operations known to be useful for image data are convolutions and could be used instead of any particular layer [47, 45].

The neural network $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$ is then viewed as a parameterized function $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters of f. The parameters are then learned to approximate a function f^*. For example $f^*(\mathbf{x}) = y$ could be a function, which assigns inputs $\mathbf{x} \in \mathbb{R}^N$ to class labels.

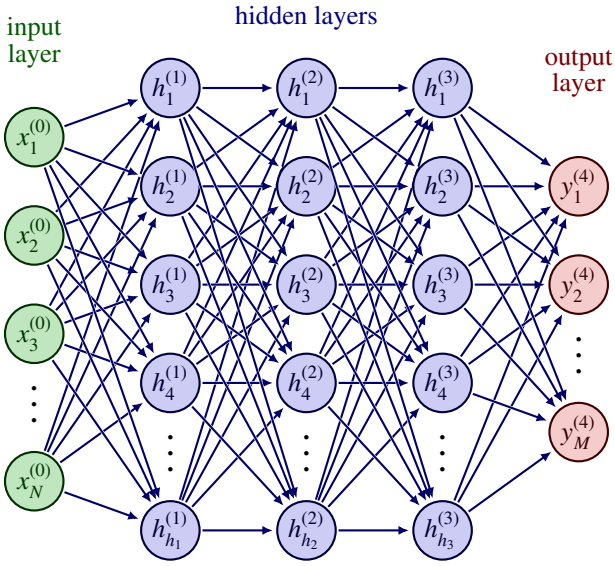


Fig. 3: A deep neural network with 4 layers mapping from $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$. x_i denotes the i'th entry of the input $\mathbf{x}_i \in \mathbb{R}^N$ for $1 \leq i \leq N$, y_j denotes the j'th entry of the output $\mathbf{y} \in \mathbb{R}^M$. $h_k^{(l)}$ denotes the k'th entry in of the hidden vector $\mathbf{h}^{(l)} \in \mathbb{R}_1^h$, which is obtained by the l'th layer $f^l : h_{l-1} \rightarrow h_l$ for $1 \leq l \leq 4$. Special cases in the notation are $h_0 = N, h_4 = M$ (the input and output dimensionalities). The visualization visualizes simple feed-forward layers, s.t. $f^l(\mathbf{h}_{l-1}) = \sigma_l\left(W_l^\top \mathbf{h}_{l-1} + \mathbf{B}_l\right)$, where $W_l \in \mathbb{R}^{h_{l-1} \times h_l}, \mathbf{B}_l \in \mathbb{R}^{h_l}$ are the paremeterization of the l'th layer. The arrows visualize all entries of the previous hidden state vector (or input) coming into each node and being multiplied by a weight (the arrow) followed by the addition of a bias, the affine transformation is followed by a nonlinear activation as usual.

A visualization of a deep neural network is given in Fig. 3. Note, that the input is not counted as a layer but marked as (0). The network consists of four layers, the input layer $f^1 : \mathbb{R}^N \rightarrow \mathbb{R}_1^h$ is the layer mapping from the input to the hidden representation, the two hidden layers (layers between the input and output layers) $f^1 : \mathbb{R}_1^h \rightarrow \mathbb{R}_2^h$ and $\mathbb{R}_2^h \rightarrow \mathbb{R}_3^h$ maps between hidden representations, lastly the output layer $f^4 : \mathbb{R}_3^h \rightarrow \mathbb{R}^M$ maps from the last hidden representation to the output representation.

To emphasize the expressiveness of deep neural networks, the universal approximator theorem states that a neural network of at least one hidden layer can approximate any function to an arbitrary degree of accuracy given enough hidden units [48, 49]. However in practice deeper networks are preferred and as mentioned in Goodfellow et al. [45], a deep architecture can bring exponential gains computationally.

3.4. Deep Reinforcement Learning (DRL)

The following will discuss broadly the two branches of policy optimization and Q-function-based methods in the setting of model-free RL [2]. "Deep" in DRL refers to the use of deep

[2]Model-free RL denotes the algorithms which do not use a model of the environment dynamics or learn a model of the dynamics to plan the actions

neural networks to represent the policy, the Q-function, or the state-value function depending on the particular DRL method.

3.4.1. Q-learning

Q-learning-based methods use a neural network to represent the q-function. The neural networks are trained to approximate the Q-function of the Bellman optimality equation, by minimizing Eq. (15). Here α denotes the step size, r_t denotes the actual reward the agent received. The target consists of the actual reward added with the expected discounted rewards of following the policy associated with the Bellman optimality equation, which is given by $\gamma \max q_t(s_{t+1}, a)$. This approach to estimating Bellman equations is known as temporal-difference learning. [4]

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha \left(\underbrace{r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)}_{\text{target}} - q_t(s_t, a_t) \right) \tag{15}$$

Some practical implementations of Deep Q-learning methods are DQN, DDQN, and Rainbow [1, 50, 51]. DQN is recognized for being the catalysator of recent DRL methods and successes. DQN was the first method to obtain super-human performance on several Atari games.

3.4.2. Policy Gradient Optimization

Policy gradient optimization is a class of methods that applies gradient-based methods to learn a policy. The policy is represented by a deep neural network $\pi_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$. In a sense, policy-based methods are more principled as they directly optimize the policy to maximize the expected returns Eq. (2), as opposed to Q-learning methods. Q-learning methods rather aim to solve the Bellman optimality equation and indirectly lead to a policy which under certain assumptions is optimal as described in Subsection 3.1.

When improving the policy, what we want is to maximize the expected discounted cumulative reward, and given we will not apply Bellman equations, it is required to represent the joint probability of the transition dynamics and the policy more explicitly. Specifically, we introduce the notion of the probability of a trajectory given transition dynamics $\mathcal{T}$, policy $\pi_{\boldsymbol{\theta}}$ and initial state probabilities ρ by Eq. (16).

$$p(\tau|\boldsymbol{\theta}) = p(s_0) \Pi_{t=0}^{T} \pi_{\boldsymbol{\theta}}(a_t|s_t) \mathcal{T}(s_{t+1}|s_t, a_t) \tag{16}$$

Additionally we will represent the undiscounted reward of a trajectory $R_\tau = \sum_{t=0}^{T} \left(\mathop{\mathbb{E}}_{s_{t+1} \sim \mathcal{T}(\cdot|s,a)} [\gamma^t \mathcal{R}(s_t, a_t, s_{t+1})] \right)$. Now the objective to maximize can be represented by Eq. (17)

$$J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathbb{E}}_{\tau \sim p(\tau|\boldsymbol{\theta})} [R_\tau] \tag{17}$$

to take, hence model-free. Model-free methods can be characterized as being trial-error learners as opposed to planners [42].

Computing the gradients of the policy's parameters $\boldsymbol{\theta}$ with respect to the expected returns Eq. (17) would not yield gradients, as the returns themselves do not depend on $\boldsymbol{\theta}$. However, applying the Log-derivative trick allows modifying the expression such that the expression will depend on the policy's parameters. The derivation is given by Eq. (19).

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \nabla_{\boldsymbol{\theta}} \left[\mathop{\mathbb{E}}_{\tau \sim p(\tau|\boldsymbol{\theta})} [R_\tau] \right] && (18) \\
&= \nabla_{\boldsymbol{\theta}} \left[\sum_\tau p(\tau|\boldsymbol{\theta}) R_\tau \right] && \text{(Expand expectation)} \\
&= \sum_\tau \nabla_{\boldsymbol{\theta}} p(\tau|\boldsymbol{\theta}) R_\tau && \\
& && \text{(Bring derivative under the summation)} \\
&= \sum_\tau \frac{p(\tau|\boldsymbol{\theta})}{p(\tau|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} p(\tau|\boldsymbol{\theta}) R_\tau && \\
& && \text{(Multiply and divide by policy)} \\
&= \sum_\tau p(\tau|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \left(p(\tau|\boldsymbol{\theta}) \right) R_\tau && \text{(Log-derivative trick)} \\
&= \mathop{\mathbb{E}}_{\tau \sim p(\tau|\boldsymbol{\theta})} \left[\left(\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \left(\pi_{\boldsymbol{\theta}}(a_t|s_t) \right) \right) R_\tau \right] && \\
& && \text{(Return to expectation form)}
\end{aligned} \tag{19}$$

From Eq. (19) an algorithm can be envisioned, which performs a batch of rollouts and computes an average across the gradients of the parameters of the policy with respect to the expression given in Eq. (19).

In practice, various modifications are made to the vanilla policy gradient method to minimize the variance and in general, obtain more practical algorithms. A typical modification is to change the sum of the experienced returns R_τ with different measures of the expected returns. A particular example of such an algorithm is Proximal Policy Optimization (PPO) proposed by J. Schulman et al. [5] which changes R_τ with a measure known as Generalized Advantage Estimation. Further, PPO addresses that the vanilla policy gradient method is prone to instability due to too large changes in the policy during training. To address this issue PPO ensures that the new policy remains "proximal" to the previous policy by proposing a clipped objective.

A key distinction between Q-learning methods and policy gradient methods presented above is that Q-learning is an off-policy method, while the policy gradient method is on-policy. Off-policy refers to the fact that behavior policy, that is the policy that interacts in the environment can be different from the target policy, which is the greedy policy associated with the Bellman optimality equation. This fact is exploited in methods such as DQN and DDQN by having a ϵ-greedy policy as the behavior policy [1, 50]. On the other hand, the on-policy objective is directly related to a specific policy, thus the behavior and target policy is the same. This explains why policy gradi-

ent methods typically augment the objective by maximizing the entropy of the policy [5, 6]. [4]

3.5. Goal Conditioned Reinforcement Learning

GCRL is an extension of RL that aims to tackle problems in which an agent must adapt to achieve multiple different goals. The key idea behind GCRL is to condition the agent's policy on a specific goal, allowing it to learn how to accomplish various tasks in a more flexible and efficient manner [52, 35, 17]. In contrast to standard RL, which focuses on optimizing a single reward function, GCRL incorporates a reward function, which is conditional on the goal. This enables the agent to acquire skills that can be reused and adapted across a range of goals. The goals can be represented in various ways e.g. images, vectors, states, or even natural language descriptions [17].

GCRL can be described as augmenting the MDP with the tuple $(\mathcal{G}, p_g, \phi, \mathcal{R}_g)$ denoted Goal-augmented MDP (GA-MDP) [17], where

- $\mathcal{G}$ is the set of goals
- $p_g : \mathcal{G} \to [0, 1]$ is the distribution of goals in the environment
- $\phi : \mathcal{S} \to \mathcal{G}$ is a function which maps states to goals
- $\mathcal{R}_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ is the adjusted reward function, which besides depending on the current state, the current action, the next state, also depends on the goal given to the agent
- $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \rho, \gamma$ is defined as the default MDP formalism given earlier

In many cases $\mathcal{S} = \mathcal{G}$ and ϕ is the identity function. The goal is to learn a policy $\pi : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to [0, 1]$, which maximizes the expected cumulative returns over the goal distribution given by (20) [17].

$$J(\pi) = \mathop{\mathbb{E}}_{s_0 \sim \rho, g \sim p_g, a_t \sim \pi(\cdot|s_t, g), s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}, g) \right] \tag{20}$$

The environments considered in this work are episodic and with a fixed goal sampled initially. The reward function is typically a sparse indicator function of the type given in (21) [17]

$$\mathcal{R}_g(s, a, s', g) = \mathbb{1}(\phi(s') = g) \tag{21}$$

When $\gamma \in [0, 1)$ the optimal policy learned from such indicator reward function can be characterized as the policy, which minimizes the number of steps before reaching the goal, thus yielding the least discounted reward.

With the function ϕ it is possible to implement the key idea of Hindsight Experience Replay (HER) [35], which introduced the idea of relabelling. The motivation is that it is possible to learn from failed trajectories, that is where the agent does not reach the intended goal, but reaches another state. To learn from failed trajectories the idea is to relabel the reached state as having been the intended goal, and thus the agent will be able to learn from the trajectory. Given, that in DRL the policy is a deep neural net, the expectation is that the policy will be able to transfer the skills learned between the relabelled states and actual goals, given similarities in the environment.

Although a GA-MDP could be modified into a standard MDP, by designing a special initial state, which samples a goal, and conveying the goal information to the agent as part of its input state representation, the GCRL formulation makes explicit the communication of meaningful information in the form of goals to direct its behavior.

One proposal to learn policies in the setting of GCRL, was proposed by Schaul et al. [52] in the form of universal value function approximation (UVFA), which extends value functions to be goal-conditioned. Through such an approach existing DRL methods can be adapted to the GCRL setting. Another approach to learning goal-conditioned policies is through supervised RL.

3.5.1. Supervised Reinforcement Learning

Supervised RL denotes the learning of policies by maximum-likelihood estimation. The general appeal is the simplicity of supervised learning methods as opposed to DRL methods [19]. Recently a number of works have shown success in formulating the RL task through a goal-conditioned imitation learning task, which can be solved by maximum-likelihood estimation [18, 20, 21]. The goal-conditioning has been shown crucial to achieving good results and highlights the connection to GCRL. However, the methods typically are limited to training in the offline setting, requiring expert trajectories such as the Decision Transformer. Albeit the work of GCSL has shown that it is possible in certain environments to learn interactively. In the following the setting of supervised RL will be characterized as it provides a unified overview of the two works GCSL [18] and the Decision Transformer [20] extended in this work.

We assume to have a GA-MDP $GM = (\mathcal{G}, p_g, \phi, \mathcal{R}_g, M)$, where M is a MDP. During training we assume to have a data set $\mathcal{D} = \{\tau_1, \ldots, \tau_n\}$. Each trajectory is an episode of fixed length H, such that $\tau_i = s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_H, a_H, r_H$ for $1 \leq i \leq n$ and $s_j \in \mathcal{S}, a_j \in \mathcal{A}, r_j \in \mathcal{R}$ for $1 \leq j \leq H$. When training the policy the idea of relabelling [35] is applied. In particular in the case of GCSL the function ϕ is the identity function, $\mathcal{G} \subseteq \mathcal{S}$, and $\mathcal{R}_g(s, a, s', g) = \mathbb{1}(s' = g)$. Then for any time point t along trajectory τ_i, the tuple (s_t, a_t, g) will be sampled, s.t. g is a relabelled future state sampled uniformly, that is $g = \phi(s')$ where $s' \sim \text{Uniform}(s_{t+1}, \ldots, s_H)$. The goal-conditioned policy π_θ is then trained to maximize the log-likelihood given by Eq. (22).

$$\max_{\theta} \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{H-1} \mathop{\mathbb{E}}_{g=\phi(s' \sim \text{Uniform}(s_{t+1},\ldots,s_H))} \log \pi_{\theta}(a_t|s_t, g) \tag{22}$$

Then when using the policy, it is assumed that a goal state is available to feed to the agent, which is satisfied in the environments used for GCSL.

In the context of the Decision Transformer, $\mathcal{G} = \mathbb{R}$ and $\phi(\tau_{t:H}) = \sum_t^H r_t$, as such the goal is the quantity denoting the reward-to-go of the trajectory. Besides the goal being characterized as the return-to-go, instead of a future state, the goal-conditioned policy is trained similarly to Eq. 22. During inference, the Decision Transformer can be directed to emit actions, that result in high reward trajectories by feeding it high reward-to-go quantities as the goal.

Algorithm 1 illustrates the general training procedure of supervised RL. The algorithm receives a data set as input, a GAMDP GM, a goal sampler f, and whether the setting is offline or offline. For GCSL f takes a subsequence of a trajectory and returns a uniform distribution over the states, whereas for the Decision Transformer f takes a subsequence of a trajectory and returns the distribution, which allocates all probability mass on the returns-to-go of the subtrajectory. *setting* denotes whether the setting is offline or online, for GCSL online is required, and will ensure that the agent collects rollouts according to its policy. Further, in the online setting the initial input data set $\mathcal{D}$ is generated from random interaction in the environment, and $\mathcal{D}$ is a FIFO buffer containing a fixed number of trajectories (10,000 in the case of GCSL [18]). In the offline setting, $\mathcal{D}$ is assumed to be a large data set containing successful trajectories.

Algorithm 1 Supervised RL

Input: Data set of trajectories, $\mathcal{D} = \{\tau_1, \ldots, \tau_n\}$
Input: A Goal-Augmented Markov Decision Process GM
Input: A goal sampler f for relabelling
Input: Online or Offline setting, *setting*
Initialize policy $\pi_{\theta}(a|s, g)$
for $i = 1, 2, 3, \ldots, N$ **do**
  if *setting* is Online (for GCSL) **then**
    Sample $g \sim p_g$, collect data with $\pi_{\theta}(\cdot|\cdot, g)$.
    Insert $\tau = (s_0, a_0, \ldots, s_H, a_H)$ into D (FIFO)
  end if
  Randomly sample a batch of trajectories:
  for $j = 1$ to k **do**
    $\tau_j \sim D$
  end for
  Sample time index for each trajectory, $t \sim [1, H]$, and sample a corresponding outcome: $g = \phi(s' \sim f(\tau_{t:H}))$
  Compute loss:
$L(\theta) \leftarrow \sum_{(s_t, a_t, g)} \log \pi_{\theta}(a_t|s_t, g)$
  Update policy parameters:
$\theta \leftarrow \theta + \eta \nabla_{\theta} L(\theta)$
end for
return Goal-conditioned policy $\pi_{\theta}(a|s, g)$

3.6. The Transformer Architecture

The transformer deep neural network architecture was proposed by Vaswani et al. in the paper "Attention Is All You Need" [22], highlighting the importance of the proposed attention mechanism. What is different from the transformer architecture and many previous architectures is that it allows for varying-sized sized inputs and can be set up to output varying-sized outputs. Thus, rather than operating on fixed-sized vectors or tensors, the transformer operates on a set of inputs. Albeit, previously proposed recurrent neural network (RNN) architectures such as LSTM [53] and GRU [54] also possess the capacity to operate on varied-sized inputs, the transformer solves two crucial limitations of the LSTM and GRU networks. In particular recurrent neural network has trouble with performing on longer sequences as the distant information may fade away, on the other hand, the transformer operates on the full set of inputs, thus avoiding information loss. The other limitation that the transformer overcomes is the avoidance of sequential computation, which is typically the case for RRNs as they operate with the abstraction of hidden states $\mathbf{h}_t$, which are dependent on the previous hidden state $\mathbf{h}_{t-1}$. The transformer instead discards the sequential computation, by relying on the attention mechanism to draw local and global dependencies across the sequence, enabling more parallelization on GPUs, thus also rendering it more computationally attractive [22].

A visualization of the transformer in the original use case of natural language translation is illustrated in Fig. 4. The transformer operates on sequences of tokens $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^{d_{\text{model}}}$ for $1 \leq i \leq n$. The tokens are vectors of a particular dimensionality d_{model}, and in the original work represented a word-piece encoding of text, thus mapping sub-words to vectors. The original transformer is composed of an encoder f_{enc} and a decoder f_{dec}. The encoder maps the input sequence to a sequence of hidden states $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$, where $\mathbf{z}_i \in \mathbb{R}^{d_{\text{model}}}$ for $1 \leq i \leq n$. The hidden state is then used by the decoder. In the context of the original transformer, the task was natural language translation, thus the input to the encoder would be a vector sequence representation of English text. The decoder would initially be fed with a start-of-sequence (sos) token, and would then output the next word. At the next iterative decoding

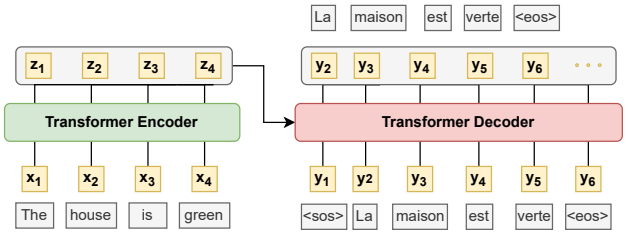


Fig. 4: A visualization of the interface of the transformer architecture in the context of the original proposed task of natural language processing. The encoder receives the English sentence "The house is green", which is converted to their respective tokens $\mathbf{x}_1, \ldots, \mathbf{x}_r$ before being fed to the encoder. The decoder receives the processed English sentence and is trained to autoregressivly predict the next French token, hence the leftwards shift in the output of the decoder.

it would be fed the sequence "sos word1" and would emit the second word and so on, until it has emitted an end-of-sequence (eos) token, at which point the decoding would be terminated. The decoder is then conditioned on the output of the encoder $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ and its decoder input sequence $\{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$, where $\mathbf{y}_i \in \mathbb{R}^{d_{\text{model}}}$ for $1 \leq i \leq k$.

3.6.1. Attention Mechanism

The key operation of the transformer is the attention mechanism. The attention mechanism operates on three sequences $\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$, $\{\mathbf{k}_1, \ldots, \mathbf{k}_n\}$ and $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$. The queries and keys are of dimensionality d_k, whereas the values may be of a different dimensionality d_v. Note that we can represent the sequences compactly in matrices $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ s.t. $Q \in \mathbb{R}^{n \times d_{\text{model}}}$. The scaled dot-product attention proposed in [22] is now given by Eq. (23).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{23}$$

The matrix product $QK^\top \in \mathbb{R}^{n \times n}$ will contain n rows, s.t. that i-th row $QK^\top_{i,:}$ denotes the dot products of $\mathbf{q}_i$ with all the key vectors s.t. $QK^\top_{i,j}$ denotes the dot-product $\mathbf{q}_i \cdot \mathbf{k}_j$. The softmax is applied row-wise, such that afterward each of the n rows sum to 1, that is $\sum_{j=1}^{n} \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}V\right)_{i,j} = 1$ for $1 \leq i \leq n$. As such the softmax maps the dot-products to row-wise simplex vectors. The factor $\frac{1}{\sqrt{d_k}}$ ensures that the dot-product is normalized proportional to the dimensionality.

In practice MultiHeadAttention is used, that is the above attention computation is duplicated $h \in \mathbb{N}$ times, h depends on the size of the transformer, but in the original setup $h = 8$. The MultiHeadAttention is given below:

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_o \\ \text{where head}_i &= \text{Attention}(QW_{q_i}, QW_{k_i}, QW_{v_i})\end{aligned}$$

Where $W_{q_i}, W_{k_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_{v_i} \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are the projection matrices mapping the input sequences to the respective query, key, and value spaces for each head $\leq i \leq h$ (in practice a bias term is often also used). $W_o \in \mathbb{R}^{h \cdot d_v \times d_{\text{model}}}$ is the output projection matrix, that ensures that the concatenated representations from the attention heads get mapped to d_{model}. The benefit of using multiple heads is that each head will learn to attend to different semantic aspects e.g. one may focus on short-term dependencies another long term dependencies [22], in the context of images each may learn to attend to various parts of an object [30].

The name attention comes from the fact that each query will obtain a new representation in terms of a convex combination of the value vectors, exactly depending on how the query "attends" to each particular key. This implies that there is a relation between the keys and values, such that how much the query $\mathbf{q}_i$ attends to the key $\mathbf{k}_j$, should indicate the value of information in $\mathbf{v}_j$ for the query. Often the distribution of attention over the keys will either follow a power-law distribution and allocate most attention to few keys and in other cases, it will resemble closer to a uniform distribution over the keys [55].

3.6.2. Architecture overview

As mentioned earlier the transformer has an encoder and decoder (f_{enc}, f_{dec}). The encoder applies the MultiHead attention operation as a self-attention mechanism. Concretely that means that it feeds the input sequence $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ as the queries, keys, and values, thus MultiHead self-attention is MultiHead(X, X, X). In the context of X being an English sentence, this enables each word to attend to each other word in the sentence and obtain a representation based on the full sentence. The decoder on the other hand operates both with self-attention followed by cross-attention. In the context of the original transformer (visualized by Fig. 4), the decoder was fed the French sentence $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$, thus the self-attention will initially allow it to obtain a representation of each token based on the full sentence. The cross-attention on the other hand is defined by setting the queries to be the sequence of the French sentence, while keys and values will be the English hidden state sequence $Z = \{\mathbf{z}_1, \ldots \mathbf{z}_n\}$. Thus, resulting in the MultiHead cross-attention MultiHead(Y, Z, Z), enabling the French tokens to attend to the full English sentence.

The full transformer architecture as proposed by Vaswani et al. is illustrated in Fig. 5. Besides the attention mechanism embedded into the MultiHead attention layers, the architecture makes use of residual connections [56] and layer normalization. Residual connections and layer normalization both address challenges in training deep neural networks, offering benefits such as improved convergence and performance. A crucial part of the transformer is positional encoding of the input sequence, as the transformer is permutation invariant with respect to the ordering of tokens.

3.6.3. Masking

When training a transformer-based architecture the idea of masking is crucial. Masking allows to limit, which tokens each particular token can attend to. For example, the original transformer was proposed for translation, thus the decoder was trained by causal/auto-regressive masking, which ensures that each token can not look ahead of its position, but only backward. In particular, it was trained in an auto-regressive way s.t. the token at the j-th position in the decoder sequence was trained to predict the $j + 1$-th token. The encoder on the other hand was trained with no masking, which models the use case of translation, as the full input English sentence is given without constraints, but the French sentence has to be decoded iteratively from left to right.

The causal masking for the decoder can be defined as $mask \in \{0, -\infty\}^{k \times k}$, s.t. $mask_{i,j} = 0$ for $j \leq i$ and $mask_{i,j} = -\infty$

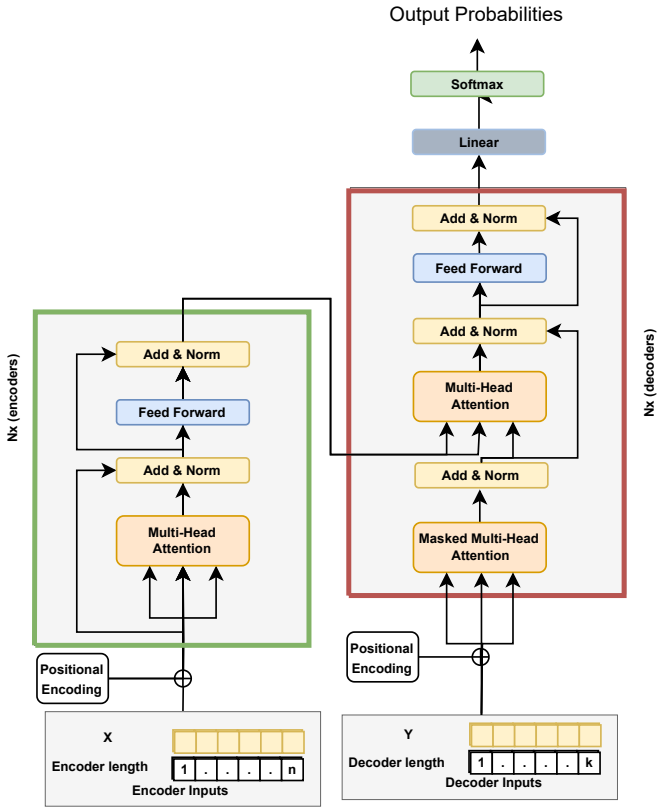


Fig. 5: The transformer architecture is composed of an encoder and a decoder. The encoder and decoder are made up of N blocks of encoders and decoders. Both the encoder and decoder apply an additive positional encoding to ensure positional information of the sequence order is embedded in the sequence of tokens. The encoder block contains a MultiHead Attention layer, where the input sequences are fed as the queries, keys, and values resulting in self-attention. It is followed by an Add & Norm layer, which applies layer normalization and adds the residual connection, which skips past the prior MultiHead Attention layer and goes directly into the Add & Norm layer. Hereafter a feed-forward network composed of 2 layers of affine transformations + nonlinear activations is applied. Lastly, this is followed by a last Add & Norm layer. The decoder block is constructed similarly, albeit it contains a MultiHead Attention layer with cross attention as the keys and values come from the encoder. Lastly, a linear transformation is applied to map the last token output of the decoder to a vector of logits, the logits are mapped to a probability distribution by the softmax operation. (Design inspired from Fig. 1 from [22])

for $i < j$. That is the mask is a lower triangular matrix, with the lower triangle filled with 0's, and the upper part (excluding the diagonal) is filled with ∞. Then the mask is added to the attention matrix from the MultiHead self-attention layer softmax($\frac{QK^\top}{\sqrt{d_k}} + mask$). This will effectively ensure that the query $\mathbf{q}_i$ can only attend to keys $\mathbf{k}_j$, where $j < i$, so that the decoder only looks backward when predicting the next token.

3.6.4. Architectural Variants

The original transformer proposed in [22] contains both an encoder and decoder, however, certain models in particular recent generative pre-training (GPT) models [57, 58, 59, 29] only uses the decoder part and its causal mask to train autoregressive-language models. It is not universally always the best approach as benefits for the original approach and other variants still exist [60], but in particular, the Decision Transformer uses a similar decoder-only setup as GPT2. The decoder-only setup is not exactly the decoder as visualized in Fig. 5, as the cross attention (second MultiHead attention block) is discarded and only the first MultiHead self-attention block with masking is maintained, as such it is more akin the encoder, but with causal masking. A third variant is a decoder-only setup, with a more flexible masking approach, known as Prefix Language Model (LM). A visualization of the variants discussed is given in Fig. 6.

3.6.5. Processing Multimodal Tokens

To process multimodal tokens multiple approaches have been proposed in previous works [23, 61, 32, 62, 63, 64]. Multimodality refers to the tokens representing data from multiple distinct modalities such as text, images, audio, etc. Four particular examples will be discussed in the following, as they will be applied in this work.

The first approach is the most straightforward approach, which feeds tokens of different modalities as a single sequence, without any further adjustments to a transformer architecture as proposed in [62]. The second approach is based on concatenation [64], which was proposed in the context of goal-conditioned supervised RL. Specifically [64] proposes to concatenate the vector representing the agent's goal with the latent vector representation of its vision-based input. This can be extended to the transformer setting by concatenating a goal vector with every input token, that is given to the transformer.

The third approach considered is that of Feature-wise Linear Modulation (FiLM) [61], which was proposed to be used in a transformer architecture to combine language instructions with an agent's visual state representation by Brohan et al. [32]. FiLM proposes to influence the output of a neural network by applying a learned affine transformation, conditioned on some input. In particular FiLM learns neural networks f and h, which outputs $\gamma_l = f(\mathbf{g}) \in \mathbb{R}^{d_{\text{model}}}$ and $\beta_l = h(\mathbf{x}_i) \in \mathbb{R}^{d_{\text{model}}}$, where l denotes the layer number. Then modulated features of the l-th layer are given by Eq. (24).

$$FiLM(\mathbf{F}_{i,l}|\gamma_l, \beta_l) = \gamma_l \otimes \mathbf{F}_{i,l} + \beta_l \tag{24}$$

In the context of the transformer $\mathbf{F}_l = \mathbf{z}_{i,l} \in \mathbb{R}^{d_{\text{model}}}$, such that the hidden representations of a token i at a particular layer l is modulated to account for additional information $\mathbf{g}$. $\mathbf{g}$ is a vector representing the information that should be embedded into the transformer, which in the context of [32] was an embedding of a language instruction.

The fourth and last form of multimodel token processing considered in this work is known as the gated fusion mechanism similar to that used in [23]. To frame the technique suppose we have a sequence of language and vision tokens: $H_{language} = \{\mathbf{z}_{l,1}, \ldots, \mathbf{z}_{l,n}\}$ and $H_{vision} = \{\mathbf{z}_{v,1}, \ldots, \mathbf{z}_{v,m}\}$ respectively. Then, we obtain a representation of the language tokens in terms of the vision tokens, by applying a single cross-attention mechanism. In particular we set $Q = H_{language}$, $K = H_{vision}$, $V = H_{vision}$ and obtain $H^{\text{attn}}_{\text{vision}} \in \mathbb{R}^{n \times d_{\text{model}}}$, by Eq. (25).

$$H^{\text{attn}}_{\text{vision}} = \text{Attention}(Q = H_{language}, K = H_{vision}, V = H_{vision}) \tag{25}$$

Then the fused output sequence $H_{\text{fuse}} \in \mathbb{R}^{n \times d_{\text{model}}}$ is obtained by Eq. (26). Here $W_l, W_v \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ are learnable weights.

$$\begin{aligned} \lambda &= \text{Sigmoid}\left(W_l H_{\text{language}} + W_v H^{\text{attn}}_{\text{vision}}\right) \\ H_{\text{fuse}} &= (1 - \lambda) \otimes H_{\text{language}} + \lambda \otimes H^{\text{attn}}_{\text{vision}} \end{aligned} \tag{26}$$

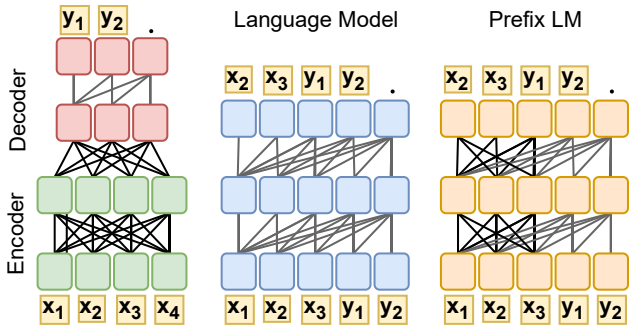


Fig. 6: Left: illustrates the original transformer architecture, the encoder receives the full sequence of tokens $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ and uses no masking (illustrated by the darker edges). The decoder receives access to the full output of the encoder, and its input sequence $\mathbf{y}_1, \mathbf{y}_2$, it is enforced a causal mask on its input (illustrated by the shaded edges) and is trained to predict the next token. Center: Illustrates the decoder-only setup used by existing GPT models. Here the decoder is fed the full sequence, the $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ denotes the prompt, and $\mathbf{y}_1, \mathbf{y}_2$ denotes its generation, it is trained with causal masking, and is trained to predict the next token. Note in the output, that the tokens are shifted left-wards, which illustrates the auto-regression, the not tagged token output will be the prediction for $\mathbf{y}_3$. Right: Denotes the decoder-only setup, with more flexible masking known as Prefix LM, that enables full visibility among the prompt tokens, while maintaining causal masking on the generative part. (Design is similar to Fig. 4 from [60])

3.7. Self-supervised learning: Variance Invariance Covariance Regularization

SSL is an approach to train DL models, wherein the data provides the supervision itself, instead of relying on human-annotated labels. This is typically done by designing a pretext task where the labels are generated automatically from the input data. [65]

In a typical self-supervised learning task, an artificial system is trained to predict or reconstruct parts of the input data. For instance, a common pretext task in computer vision is to predict patches of images given other patches [31, 30], or in natural language processing, predicting the next word in a sentence [59, 57, 58, 29, 28, 60].

The main advantage of self-supervised learning is that it can leverage large amounts of unlabeled data, which are more abundant than labeled data, to learn useful representations. These learned representations can then be used for various downstream tasks with smaller amounts of labeled data, such as classification or detection, often improving the performance of these tasks. [65]

Within the subfield of SSL, various methods exist, however in this work the method of Variance Invariance Covariance Regularization (VICReg) [40] is applied to stabilize the learning process of the transformer used in the GCSL setting. The method is versatile and flexible. VICReg was initially proposed in the context of learning representations of image data.

In the context of image data, the VICReg methods work by having an encoder network f_{enc}, which maps the image to a latent representation in the form of a vector of dimensionality d_{hidden}. To train the image encoder, an image i is sampled from the data set $\mathcal{D}$ and then two distinct image-based transformations (cropping, rotation, etc. detailed description is given in [40]) are sampled $t, t' \sim T$. Then two views of the image are obtained $x = t(x), x' = t'(x)$. To encourage that the image encoder is invariant to the transformations, the mean squared distance between $\mathbf{z} = f_{\text{enc}}(x_1)$ and $\mathbf{z}' = f_{\text{enc}}(x_2)$ is minimized. However, without further consideration, the representations will collapse, that is the network will learn to map every input to the same constant vector. The collapse and the tendency of neural networks to minimize the loss function by bypassing the intention of the designer is well known in SSL [65]. To avoid the collapse problem besides the invariance term, VICReg proposes variance and covariance terms. The variance and covariance are computed across the batch thus we define the batches processed by the image encoder as $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}, Z' = \{\mathbf{z}'_1, \ldots, \mathbf{z}_n\}$. VICReg proposes the training objective given by Eq. (27). λ, μ, ν are the respective weights for the invariance part of the loss s (s stands for similarity), variance part v, and the covariance part c

$$L(Z, Z') = \lambda s(Z, Z') + \mu \left[v(Z) + v(Z')\right] + \nu \left[c(Z) + c(Z')\right] \tag{27}$$

As mentioned earlier the invariance part is to ensure that two transformed views map to similar representations and is given by Eq. (28).

$$S(Z, Z') = \frac{1}{n} \sum_{i}^{n} \|\mathbf{z}_i - \mathbf{z}'_i\|_2^2 \tag{28}$$

The variance term is used to ensure that the representations do not collapse to a constant representation. This is implemented by using a hinge loss, that ensures that the standard deviation across the dimensions of the latent representation is above a fixed threshold γ_v across the batch ($\gamma_v = 1$ in [40]). The batches Z and Z' each contain different images, thus the variance term motivates each dimension of the learned representation to vary depending on the image. The variance term is given by (29). Here ϵ is a small scalar for numerical stabilization, and $\mathbf{z}^j$ is the vector consisting of each value at dimension j across the batch.

$$v(Z) = \frac{1}{d_{\text{hidden}}} \sum_{j}^{d_{\text{hidden}}} \max\left(0, \gamma_v - \sqrt{Var(\mathbf{z}^j) + \epsilon}\right) \tag{29}$$

It may seem that ensuring the invariance between similar images and variance between different images in the representation space may be enough to avoid collapse, but it does not work empirically as shown in [40]. The network may learn a few constant representations, which are large in terms of the absolute values and will be able to minimize the loss of the objective. To address this, it is necessary to constrain the magnitude of the learned representations. To do this VICReg proposes the covariance term as the sum of the squared off-diagonal entries of the covariance matrix $C(Z)$. More specifically given by (30).

$$c(Z) = \frac{1}{d} \sum_{i \neq j} \left[C(Z)_{i,j} \right]^2 \tag{30}$$

Where the covariance matrix is computed as by Eq. (31)

$$C(Z) = \frac{1}{n-1} \sum_{i}^{n} (\mathbf{z}_i - \overline{\mathbf{z}})(\mathbf{z}_i - \overline{\mathbf{z}})^\top \tag{31}$$

$$\text{where } \overline{\mathbf{z}} = \frac{1}{n} \sum_{i}^{n} \mathbf{z}_i$$

When minimized the covariance term encourages the off-diagonal entries to be close to 0, thus it forces the required variance from the variance term to be de-correlated from the other images in the batch. In aggregate the three terms composed in Eq. 27 are shown to be capable of learning representations competitive with existing state-of-the-art methods [40]. As a note, in practice, VICReg proposes to have an expander network, that expands the dimensionality of the latent embeddings and then applies Eq. (27) on the expanded representation, then for downstream tasks the expander is discarded. The use of an expander network is shown empirically to yield better performance.

4. Goal Conditioned Reinforcement Learning with Transformers

In the following, we aim to investigate the use of a transformer-based architecture in the setting of Goal Conditioned Supervised Learning (GCSL)[18]. In particular, we will not propose novel modifications, techniques, or similar, rather it is the combination of existing techniques covered in section 3 that is novel. Further, the key motivation is to observe whether there will be a transfer between the online interactive setting of GCSL, where the agent is given no expert trajectories, and to the offline setting with expert trajectories. The online setting in the context of supervised RL is intricate in that the model is not directly optimized to maximize a reward correlated with reaching goals. Instead, the goal-reaching behavior has to be obtained through maximum-likelihood estimation, by configuring the training procedure and architectural setup such that the model converges to goal-reaching behaviors. Thus, the GCSL setting is intended as an exploratory setting to obtain a configuration of a transformer-based architecture, that converges to such solutions in the respective environments (visualized in Fig. 1). Following this, the intention is to evaluate the obtained configuration in a more classical offline supervised RL setting by augmenting the Decision Transformer [20] with the found modifications.

The setup will be such that the experiments conducted in the setting of GCSL will be presented in section 5, followed by the evaluation in the offline setting performed in section 6.

We aim to answer the following three concrete questions:

1. What modifications are needed to adapt the transformer to work in the setting of GCSL?
2. Will there be a performance improvement on the baseline environments used in the original work? (visualized in Fig. 1
3. Will the modifications lead to improvements in the setting of offline data (Decision Transformer setting)?

5. Online data: Extending GCSL with a Transformer

We conduct experiments in two phases, with the first being experimental, and the other phase being benchmarking. The environments used in [18] are visualized in Fig. 1, for the experimental phase we conduct all our experiments in the Lunar Lander environment to enable fast iterations, in the second phase we benchmark the best-performing transformer model, and the original GCSL model on the remaining environments[3].

5.1. Experimental: Default transformer setup

To motivate the need for modifications to existing approaches of applying transformers in the context of RL, we present a default setup and show that it fails in the setting of GCSL.

The initial proposed setup of the transformer architecture is to use a decoder-only configuration, similar to GPT2 [58] and the Decision Transformer [20]. The motivation here is that it is a configuration, which is proven in the context of generating actions from a sequence of past states and actions [20, 32, 21]. Additionally, the ability to apply masking to regulate information available when generating a particular action makes the transformer suitable for a convenient implementation of relabelling. Visually the setup is illustrated by Fig. 7.

The figure illustrates that transformer receives a trajectory $\tau = s_t, a_t, \ldots, s_{t+H}$ of fixed length H. Further, it is trained autoregressively such that, the token s_k will be mapped as the prediction of a_{k+1} for $t \leq k \leq t + H$. Besides, applying a causal mask, a time point $k \in \{t, \ldots, t + H\}$ will be sampled, and then every token between s_k up to the token s_{t+H} will be masked. Thus, the only information available to the model to decode the connecting trajectory is the information before time point k and the state s_{t+H}. As such the future state s_{t+H} will act as a goal. A similar form of masking named Random Masked Hindsight Control has been proposed concurrently by [66], albeit in the

[3] We do not use the dexterous object manipulation environment (see Fig. 1), as the action space is significantly different from the remaining environments.

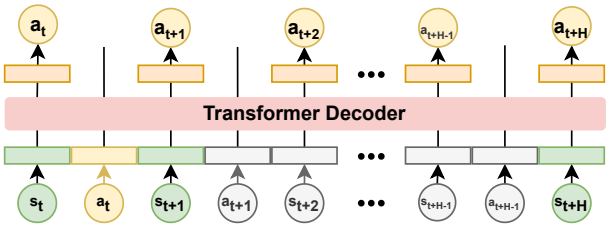


Fig. 7: The proposed default transformer configuration used in the setting of online supervised RL (GCSL). The transformer receives a sequence of H states and actions. The relabelling of a future state as a goal state is obtained, by allowing every state to attend to the last state here S_{t+H}, while masking all information between the relabelled state S_{t+H} and a randomly selected time point here $t+1$. The model is trained to autoregressively decode the next action, specifically the token s_t will be mapped to the prediction of a_t.

offline setting.

Unfortunately, our results show that in the online setting, this approach will fail, as shown by Fig A.19 available in Appendix Appendix A. The graph of this experiment is not particularly interesting as the proposed model simply flatlines in terms of learning to reach closer and closer to the goal throughout the training. However, the interesting aspect is that loss goes down rapidly throughout training, from this we deduce that the model converges, but that it is misaligned with the intention of it being capable of reaching the goal states. Upon inspection, we see, that the model will learn to always predict the same action. Given that the setting is online, the model is able to influence its own training data set, thus by learning the generated behavior of predicting the same action, it can afterward easily maximize its prediction quality. Such behavior is reminiscent of the collapse problem in SSL discussed in section 3.7. To combat this effect, we introduce the following two ideas to regularize the learning process:

- The use of a target network, proposed by Mnih et. al [1] (DQN paper)
- The use of Variance and Covariance regularization from Bardes et al. VICReg [40].

The use of a target network is known to stabilize learning in the online setting. The target network is an exponential moving average of the network. Here we propose to use the target network in the interaction with the environment, this will ensure that the target actions to predict are not generated exactly by the underlying model itself. The use of Variance and Covariance is an attempt to address the collapse into a constant action, by attempting to decorrelate the latent embeddings obtained through the transformer. Unfortunately, none of the modifications works either.

From this initial experiment, we conclude that the approaches shown to work with stable offline data do not guarantee success in the online setting. Further, we develop the intuition that the simpler baseline architecture has two direct differences as compared to the transformer setup. In particular, the network takes only a single state and the goal as input, rather than a sequence, which allows the goal information to be more discernible and avoids dilution, as when having a full sequence. The other key difference is that the transformer decodes all the actions in the sequence, thus with a sequence length of 50, it will converge much faster toward the distribution in its buffer. This will disturb the interaction between how fast the model converges toward the behavior seen in its buffer, and the collection of new data. To remedy these two aspects we propose adjustments in the next subsection.

5.2. *Experimental: Regularized Transformer*

From the intuition developed from the previous experiments, we propose to use the transformer in a highly constrained configuration. The Fig. 8 illustrates the simplified setup, where the notion of sequences is discarded and only a single state s_t and goal-state s_k, where $t < k$ is sampled from the trajectory during training. As such the setup minimizes the dilution of the goal information as the token s_t which maps to the action prediction is constrained to only attend to itself and the goal state. Additionally, the transformer only decodes the action for the state s_t, thus the interaction with the optimization frequency and the data collection is similar to the baseline.

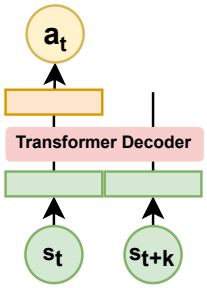


Fig. 8: A constrained configuration of a decoder-only transformer, where it is fed only state s_t and a future goal-state s_k s.t. $t < k$, and is trained to emit a single action a_t.

We maintain the modifications of adding a target network and using Variance and Covariance regularization and evaluate the various ablations with and without these modifications. As illustrated by Fig. 9 we see that models are able to avoid collapse and learn a goal-reaching behavior comparable to the baseline GCSL. The top figure illustrates the average final distance to the goal position on evaluation runs as a function of training timesteps. The bottom figure illustrates on the y-axis the proportion of rollouts, which reach within a predetermined distance of the goal state, while the x-axis similarly denotes training timesteps. Further, what we see is that the models with variance and covariance regularization (Transformer_S_Single_Dec_VR) and the model, which additionally adds the use of a target network (Transformer_S_Single_Dec_TGT_VR) performs better than the baseline ($\approx$ .1 higher success ratio on the bottom plot in Fig. 9.).

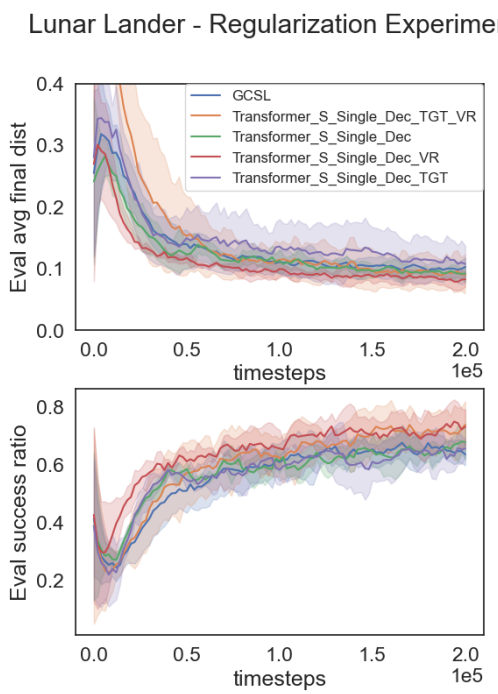


Fig. 9: Decoding a single action with a decoder-only transformer, which only receives the current state and a goal-state does not collapse. Top figure: y-axis denotes the final distance to the goal state. The x-axis denotes the number of training steps. Bottom figure: y-axis denotes the evaluation success ratio, that is the proportion of rollouts that end up within a predetermined distance of the goal state. The x-axis denotes the number of training steps. S denotes that the model size used is small. Single_Dec denotes that the model only outputs a single action. TGT denotes that a target network is used. VR denotes that variance and covariance regularization is used.

This is however a limited setting, as we do not exploit the full potential of the transformer, which was built for sequential data inputs. Thus for the next set of experiments we return to the sequential input.

5.3. Experimental: Sequential Transformer with Careful Goal Mixing

Architecturally we return the sequential configuration illustrated in Fig. 7. However, we only decode a single action. Further, we aim to explore different ways to merge the goal information into the sequence of past states and actions. As the base configuration, we use variance and covariance regularization as it showed the best performance in the previous experiment. Thus we evaluate the following configurations:

- Transformer_Base: denotes the setup shown in Fig. 7, with single action prediction. Further, it uses variance and covariance regularization
- Transformer_Base_Concat: adds to the base model concatenation-based merging of the goal token with the state and action tokens.
- Transformer_Base_FiLM: adds to the base model FiLM-based merging of the goal-state into the representation of state and action tokens.
- Transformer_Base_Gate: adds to the base model the use of a gated fusion mechanism to merge goal information into the representation of state and action tokens.

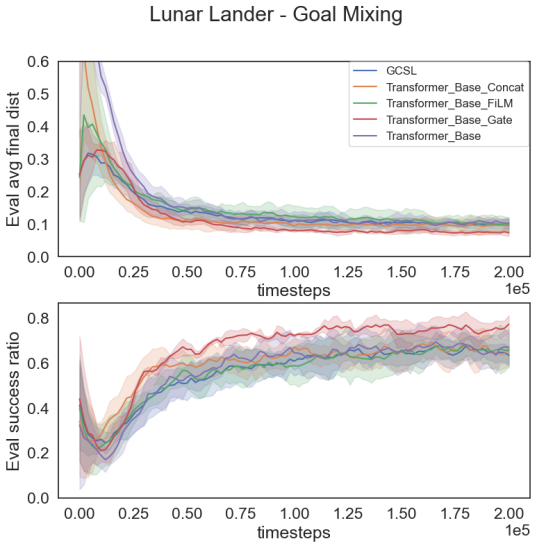


Fig. 10: Illustrates the performance of various approaches to merge the goal information with the sequence of states and actions fed to the decoder-only transformer. A description of the respective axes of the subplots is given in the description of Fig. 9

From Fig. 10 (top and bottom subfigures and axis are similar to the previously discussed Fig. 9) it can be seen that all approaches manage to avoid collapse. Further, all variants have performance similar to the baseline model GCSL, except for Transformer_Base_Gate, which has improved performance compared to the baseline ($\approx .15 - .2$ higher success ratio compared to GCSL). From this experiment we learn, that modeling a sequence of states and actions is possible in the online setting, and can increase performance with modifications. In the next subsection, we benchmark the best-performing model Transformer_Base_Gate and GCSL on the remaining environments and provide a summary.

5.4. Benchmark results and Summary

From Fig. 11 it can be seen that the proposed model Transformer_Base_Gate (denoted as Transformer in the figure), performs better than the baseline GCSL on three environments, and is on par in the last. We are now ready to answer two of the three questions posed as the motivation for extending GCSL with a transformer.

1. What modifications are needed to adapt the transformer to work in the setting of GCSL?

From the experiments, we conclude that it is necessary to ensure that the learning process does not converge too fast toward the distribution of behavior in the buffer, especially early, when the behavior is random. To combat this, we set the frequency of training updates to data collection similar to the baseline, by only training from the prediction of a single action prediction

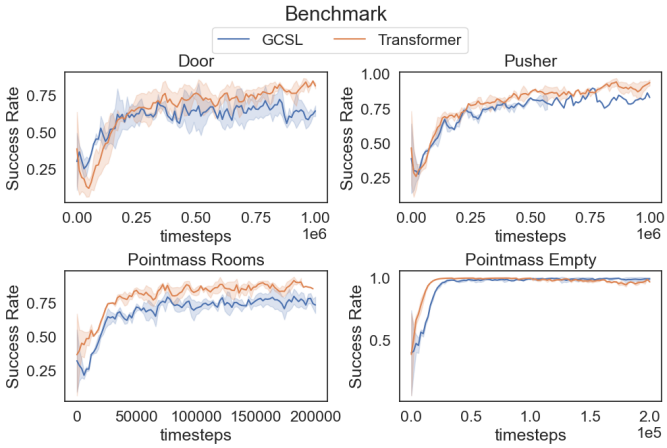


Fig. 11: Benchmarking shows the transformer with variance and covariance regularization, a fusion gating mechanism to merge goal information into the transformer, and the adjusted training procedure of only training on the prediction of a single action yields performance that is on par or better than the baseline GCSL network. The four benchmarks are the Door environment: which consists of opening a door with a Sawyer arm, Pusher environment: A Sawyer arm has to push a small object to a predetermined position, Pointmass Rooms: simple goal-reaching environment with 4 rooms, Pointmass Empty: simple goal-reaching environment with no rooms. Environments are visualized in Fig. 1. On each subfigure: the y-axis denotes the success ratio, and the x-axis denotes the training timesteps.

produced by the transformer. Further, to achieve performance beyond the baseline GCSL we find it necessary to apply variance and covariance regularization, along with using a gated fusion mechanism to merge the goal information with the sequence of state and action tokens.

- Will there be a performance improvement on the baseline environments used in the original work? (visualized in Fig. 1)

As illustrated by Fig. 11 with the above-mentioned modifications it is possible to exceed the performance of the baseline GCSL approach.

The third question to be addressed in the following section is directly aimed at evaluating whether the modifications found to work in the online setting will transfer additional performance to the offline setting.

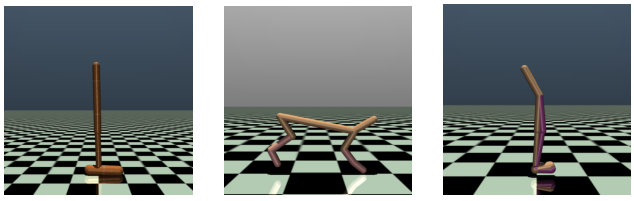
6. Offline data: Modifying the Decision Transformer

Through the experiments conducted in this section, we aim to address question 3.

- Will the modifications lead to improvements in the setting of offline data (Decision Transformer setting)?

By *modifications*, we refer to the following: variance and covariance regularization and the use of the gated fusion mechanism to merge goal information into the input sequence fed to the transformer. The effect of decoding a single action will also be evaluated, albeit intuitively it will not be beneficial as the data is stable in the offline setting.

From the work of the Decision Transformer, we select the three offline environments Hopper, HalfhCheetah, and Walker2d originally proposed in the 'Datasets for Deep Data-Driven Reinforcement Learning' D4RL [67], which contains multiple data sets for evaluating offline RL agents. The environments are visualized by Fig. 12.



(a) Hopper (b) HalfCheetah (c) Walker2d

Fig. 12: Illustration of different Mujoco environments used for training and evaluating locomotion tasks. (a) Hopper, (b) HalfCheetah, and (c) Walker2d. Each environment presents a unique challenge for the robot to learn specific locomotion skills while maintaining stability and achieving the task objective. In these environments, the robots are rewarded for effectively moving forward while ensuring stability and avoiding falls.

Each of the three environments comes in three categories: medium-replay, medium, and expert, which denotes the quality of the trajectories as compared with the performance with a state-of-the-art RL agent is capable of achieving in the respective environments. For computational efficiency, we will evaluate the ablations of the proposed modifications on the Hopper medium-replay data set. An evaluation of the proposed model consists of training exclusively on the available offline data for 100,000 training steps (similar to the Decision Transformer), and evaluation is done every 1,000 steps of training by performing 100 rollouts in the actual environment. We plot on every figure expert-normalized score $= 100 \cdot \frac{\text{score}-\text{random score}}{\text{expert score}-\text{random score}}$, as proposed in [67], similarly to the Chen et al. [20] we use the expert score and random score listed in the D4RL dataset [67]. Similarly to the previous experiments, we run each experiment for three runs. Additionally, the action space is continuous, thus the log-likelihood maximization objective is changed with the Mean Squared Error objective similar to the Decision Transformer baseline. Lastly, we highlight that the Decision Transformer, which as discussed earlier uses the returns-to-go $R_t = \sum_t^H r_t$ as the goal, will place the returns-to-go as breadcrumbs along the input sequence. In particular the Decision Transformer feeds sequences of the form $R_t, s_t, a_t, R_{t+1}, s_{t+1}, a_{t+1}, \ldots, R_H, s_H, a_T$. On the other hand, when using the gated fusion mechanism, we feed sequences of the form $s_t, a_t, s_{t+1}, a_{t+1}, \ldots, s_H, a_T$ and use R_0 as the goal token. Thus, it can be seen as a more general form of conditioning in that R_0 does not have the same dense information of the underlying MDP, which is provided in the Decision Transformer configuration.

From the ablation experiments provided in Figure 13, we observe that only the approach of using the gated fusion mechanism yields reliably better performance than the baseline. Thus, we proceed with comparing the Transformer_Gated with the baseline Decision Transformer (named DT in the figures).

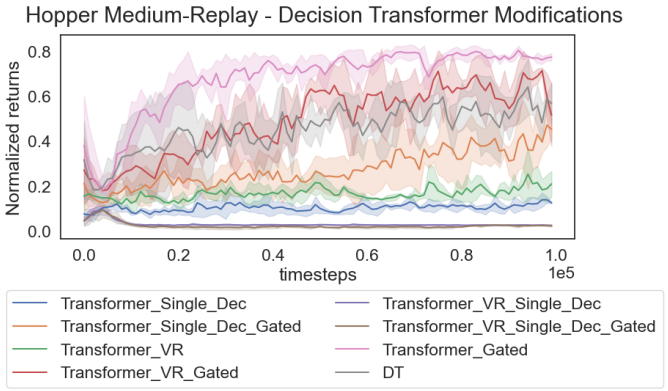


Fig. 13: The figure illustrates the expert normalized returns (y-axis) on the Hopper environment when the models are trained on the medium-replay data set. The x-axis denotes the number of training timesteps. The figure illustrates all the ablations of the Decision Transformer evaluated. DT is the baseline Decision Transformer. VR denotes that variance and covariance regularization is applied. Single_Dec denotes that the model is trained to only predict a single action. Gated denotes that the gated fusion mechanism is applied.

From Fig. 14 we see the performance in the Hopper environment, when trained on the three respective offline data sets medium, medium-replay, and expert. In general, we see that the proposed Transformer_Gated performs better, and specifically is much more stable during the training process as compared to the baseline Decision Transformer. When assessing the performance on the Walker2d environment from Fig. 15 and the HalfCheetah environment from Fig. 16, the performance difference is not as markedly discernible but is on par or slightly above in most environments.

Summarily for the answer to question 3, we find that only the gated fusion mechanism transferred performance to the offline setting successfully.

7. Goal-Conditioned Reinforcement Learning through Intrinsic Reward

This part of the work moves towards a model-based RL approach and incorporates an intrinsic reward. Many previous works exist in each of these two categories, some of which are [68, 69, 70, 71]. In this study, we aim to investigate a goal direction network's integration with an RL agent, facilitated through an intrinsic reward mechanism. Broadly, our approach entails training the goal direction network to produce a latent representation of the next state. The RL agent, in turn, strives to align the actual next state with this imagined one. We use the difference, measured as the squared L2 norm, between these two states as a negative reward. Consequently, this creates a feedback loop where the agent is incentivized to take actions that minimize this difference.

Our work presents a contrast to prior work involving Goal-Conditioned Supervised Learning (GCSL) and the Decision Transformer. While GCSL's ability to learn online seems largely dependent on the simplicity of the environments, our

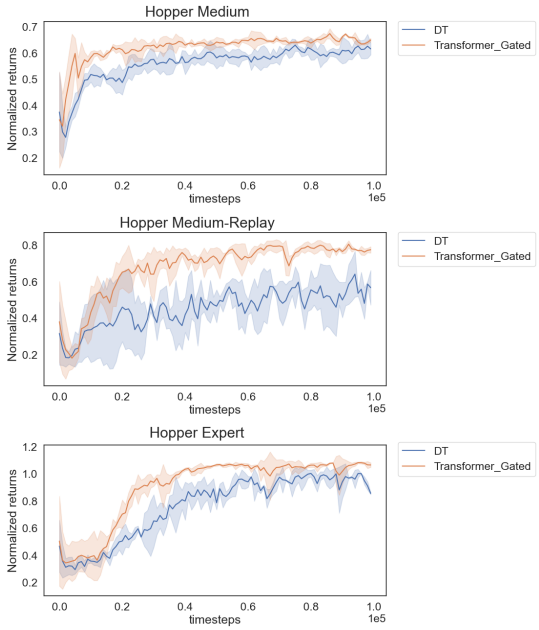


Fig. 14: Top: Hopper Medium, Middle: Hopper Medium-Replay, Bottom: Hopper-Expert. The y-axis on each subfigure denotes the expert-normalized scores evaluated on 100 rollouts every 1,000 steps of training, the x-axis denotes the training timesteps.

approach attempts to be more robust by leveraging traditional Deep Reinforcement Learning (DRL) principles. Specifically, we focus on maximizing rewards that align with specific goals. Unlike the Decision Transformer, which assumes specific types of trajectories based on state-action pairs and rewards, we consider trajectories characterized by goals and observations. We believe this shift offers potential advantages, particularly the ability to train the goal direction network from observation-based data like videos. This action-agnostic design could provide flexibility, maintaining usability even when the action space of the agent platform changes.

This exploration investigates the interaction of two distinct neural networks, the RL agent and the goal direction network, communicating via a feedback loop. This approach mitigates the need for hand-engineered, task-specific reward functions, instead proposing a universal reward function facilitated by a goal-direction network. This approach is founded on the intuition that if an agent has both the capacity to identify its goal and to evaluate the value of its actions relative to that goal, the reward function implicitly emerges. The agent would then naturally adjust its actions to maximize the likelihood of reaching its stated objective. However, this shift does not eliminate complexity; it redirects the challenge toward developing and training a sophisticated goal-direction network, capable of effectively directing the agent's actions. Despite the potential advantages, there are significant limitations to consider. The goal direction network, formulated as a dynamics model, might not always generate the most effective representation of the agent's goals. It might either over-encapsulate unnecessary details or under-specify important ones. Additionally, the

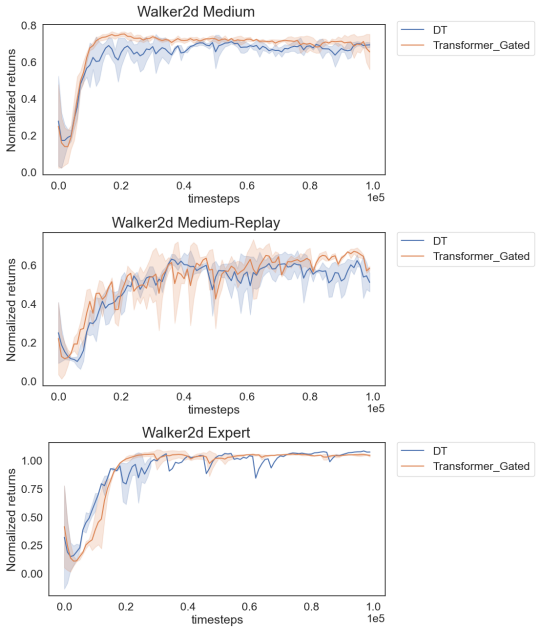


Fig. 15: Top: Walker2d Medium, Middle: Walker2d Medium-Replay, Bottom: Walker2d Expert. y and x-axis similar to Fig. 13

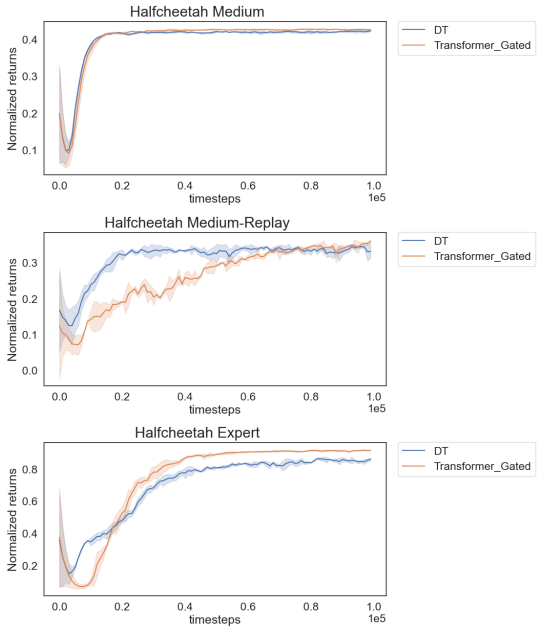


Fig. 16: Top: HalfCheetah Medium, Middle: HalfCheetah Medium-Replay, Bottom: HalfCheetah Expert. y and x-axis similar to Fig. 13

network might struggle to generate meaningful next-state representations when faced with novel states, potentially impacting its generalizability.

Our method proposes an adaptation of the existing environment model learning approach [68], incorporating specific modifications. The objective is to train a suite of neural networks using observation-based data, which could also be action labeled, structured in the form $\tau = g, s_0, s_1, \ldots, s_H$. Here, g signifies a goal that either articulates the agent's intent or defines the trajectory's quality. The neural networks in consideration are:

- A state encoder: state_encoder$(s_t) = z_t$ where $z_t \in \mathbb{R}^d$. This can be potentially trained using self-supervised learning methods such as autoencoders [72, 31].
- A goal-conditioned dynamics model: $P(z_t, g) = \hat{z}_{t+1}$ which predicts the next latent state given the current state and the goal.
- A goal-conditioned RL agent: $\pi(z_t, \hat{z}_{t+1}) = a_t$, responsible for determining the actions within the environment.

Once the state encoder and the goal-conditioned dynamics model have been trained, the next step involves training the GCRL agent within the environment, guided by the reward function specified in Eq. (32). $\hat{z}_{t+1}$ denotes the prediction of the next latent state as obtained by the goal-conditioned dynamics model P. z_{t+1} denotes the embedding of the actual next state.

$$
\begin{aligned}
r_t &= -\|\hat{z}_{t+1} - z_{t+1}\|_2^2 \\
&= -\|P(z_t, g) - \text{state\_encoder}(\mathop{\mathbb{E}}_{a\sim\pi(\cdot|z_t,\hat{z}_{t+1}),s_{t+1}\sim\mathcal{T}(\cdot|s_t,a_t)}[s_{t+1}])\|_2^2
\end{aligned} \tag{32}
$$

The RL agent can only influence the reward function through the actions it takes, as $P(z_t, g)$ is fixed when training the RL agent, and the transition dynamics $\mathcal{T}$ are given from the MDP defining the environment. Additionally, this highlights why it is not possible to simply train a neural network to minimize Eq. 32, as the gradients will not be able to be computed through the world, that is $\mathcal{T}$, instead an RL formulation is required.

To validate the proposed method, we train an agent capable of solving a 10x10 goal-reaching gridworld. Fig. 18 illustrates a gridworld with an agent and a goal position. The *state_encoder* is trained to learn the discrete cell position given the 2d grid board as input. The goal-conditioned dynamics model is trained to predict the next cell position given the current cell position of the agent and the final goal position. As the RL agent, we use a discrete PPO agent [5].

In Fig. 17 the proposed setup named Intrinsic PPO manages to solve a 10x10 goal-reaching gridworld environment, earlier than the baseline PPO. The baseline goal-conditioned PPO is trained with the sparse indicator reward Eq. (21).

Despite the encouraging result in the simpler gridworld environment, our attempts to adapt the method to more complex environments, such as the CarRacing OpenAI gym environment [73], have yet to yield successful results.

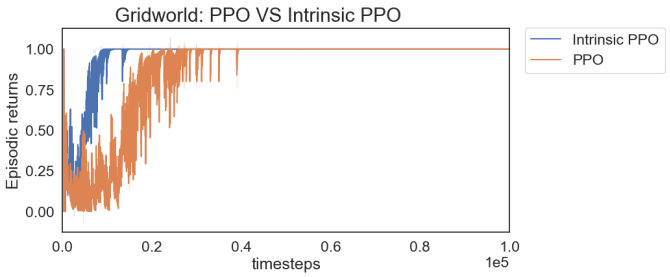


Fig. 17: Intrinsic PPO: uses the proposed goal direction network and a goal-condition PPO agent. PPO is the default discrete PPO agent. The y-axis shows the episodic return, where 1 denotes successfully reaching the goal cell. The x-axis shows the number of training steps.

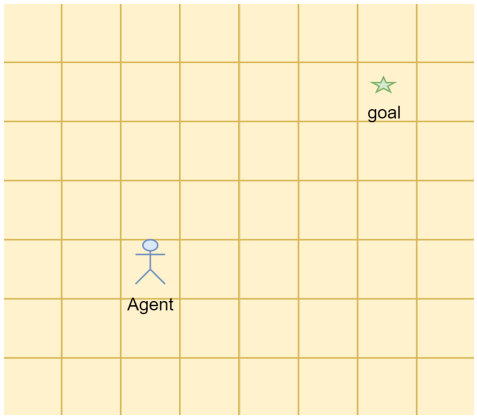


Fig. 18: Visualization of a gridworld goal-reaching environment. The agent and goal are spawned randomly at their respective cell positions. The agent can move up, down, left, and right. The aim is to reach the goal position.

8. Conclusion and Future Work

To the best of our knowledge, this work presents the first use of a transformer-based architecture in online supervised RL, by extending GCSL [18], which used an ordinary neural network. We found that it was possible only with careful modifications. In particular, we found, that it was crucial to maintain a frequency between the optimization steps and the data collection to avoid model collapse. Further, to obtain improvement beyond the baseline we found it required to use variance and covariance regularization [40], along with the use of a gated fusion mechanism [23] to merge the goal information with the sequence of states and actions tokens.

When exploring the transferability from the modifications found to work well in the online supervised RL setting to the offline setting, we found that the modifications in general did not guarantee improvements. In particular, we only found the use of the gated fusion mechanism to yield a more stable learning process with slightly higher performance compared to the baseline Decision Transformer [20], on the D4RL data sets [67] considering the Hopper, HalfCheetah, and Walker2d environments. More consequently the work presents a direction for a more discrete representation of the goal, in that we only feed a single reward token as the goal, as compared to [20], which performs best when feeding the returns-to-go at every time step. The benefit of this is simplified goal specification.

Lastly, we proposed the idea of training a goal-conditioned dynamics model coupled with a classical DRL agent, such that the agent can be trained by the intrinsic reward of maximizing the negative squared L2 norm between the representation of the actual next state and the next state representation given by the goal-conditioned dynamics model. We evaluated its performance on a simple 10x10 goal-reaching gridworld environment and found it to perform better than the baseline discrete PPO. However, despite our best efforts, it has not been possible to extend the method to the more challenging CarRacing environment.

For future work, the following directions seem promising. To evaluate the scalability of the proposed Decision Transformer architecture with the gated fusion mechanism in the setting of offline data, it would be enticing to evaluate it on the multi-task Atari data set used by the Multi-Game Decision Transformer [21]. Besides providing insights into the scalability of performance on larger data sets, the multi-task challenge is a stride toward a single agent capable of handling and reaching many diverse goals.

Another direction of work more attainable would be to continue the work on the proposed approach of GCRL through intrinsic reward, by extending and exploring it more thoroughly in more complex environments such as the CarRacing environment.

Finally, it might be worth considering the potential benefits of exploring computational forms of consciousness [37, 36, 39]. This interest arises from an observable gap in the current models – a seeming lack of intent or alignment. Our proposed goal-direction network is a step in the direction of enabling a more explicit intent. Yet, there could be more valuable approaches to this challenge. One such method might involve generating detailed expectations about the future, such as anticipated joint positions and associated sensor observations, and then using this information for action guidance. Alternatively, an approach akin to the Global Workspace Theory or what by Deheane et al. [36] is characterized as C1, could be considered. Here information from various unconscious processes is gathered, the most pertinent information is selected and funneled through a kind of 'working memory bottleneck', and then this information is broadcast to the rest of the system, such that it can guide a broad variety of behaviours [36]. Rather than solely relying on high-level latent representations of the entire state, these approaches could leverage more granular foresight or synchronized system-wide intent to navigate environments. While it's a challenging prospect, it could be an interesting direction for future research.

9. Code Availability

- The code used for all experiments GCSL, Decision Transformer, and the proposed intrinsic reward approach is available here: `https://github.com/miladsamim/gcrl`.

- The original source code GCSL is available at: https://github.com/dibyaghosh/gcsl
- The original source code for the Decision Transformer is available at: https://github.com/kzl/decision-transformer.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533. doi:10.1038/nature14236.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. e. a. Lanctot, Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., Grandmaster level in starcraft ii using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354. doi:10.1038/s41586-019-1724-z.

[4] S. Zhao, Mathmatical Foundations of Reinforcement Learning, Springer Nature and Tsinghua University Press, 2023, https://github.com/MathFoundationRL/Book-Mathmatical-Foundation-of-Reinforcement-Learning.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, CoRR abs/1707.06347 (2017). arXiv:1707.06347.
URL http://arxiv.org/abs/1707.06347

[6] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1861–1870.
URL https://proceedings.mlr.press/v80/haarnoja18b.html

[7] D. Amodei, J. Clark, Faulty reward functions in the wild, https://blog.openai.com/faulty-reward-functions/ (2016).

[8] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2002.

[9] P. Agrawal, The task specification problem, in: A. Faust, D. Hsu, G. Neumann (Eds.), Proceedings of the 5th Conference on Robot Learning, Vol. 164 of Proceedings of Machine Learning Research, PMLR, 2022, pp. 1745–1751.
URL https://proceedings.mlr.press/v164/agrawal22a.html

[10] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 41–48.

[11] T. Matiisen, A. Oliver, T. Cohen, J. Schulman, Teacher–student curriculum learning, IEEE Transactions on Neural Networks and Learning Systems 31 (9) (2020) 3732–3740. doi:10.1109/TNNLS.2019.2934906.

[12] V. Gullapalli, A. Barto, Shaping as a method for accelerating reinforcement learning, in: Proceedings of the 1992 IEEE International Symposium on Intelligent Control, 1992, pp. 554–559. doi:10.1109/ISIC.1992.225046.

[13] A. Ng, D. Harada, S. J. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: International Conference on Machine Learning, 1999, p. 278–287.

[14] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, JMLR.org, 2017, p. 2778–2787.

[15] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, R. Munos, Unifying count-based exploration and intrinsic motivation, in: D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 29, Curran Associates, Inc., 2016, pp. –.
URL https://proceedings.neurips.cc/paper_files/paper/2016/file/afda332245e2af431fb7b672a68b659d-Paper.pdf

[16] S. Levine, Understanding the world through action, in: Proceedings of the 5th Conference on Robot Learning, Vol. 164 of PMLR, Robot Learning, 2021, pp. 1752–1757, blue Sky Papers.
URL https://openreview.net/forum?id=L55-yn1iwrm

[17] M. Liu, M. Zhu, W. Zhang, Goal-conditioned reinforcement learning: Problems and solutions, in: L. D. Raedt (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 5502–5511, survey Track. doi:10.24963/ijcai.2022/770.
URL https://doi.org/10.24963/ijcai.2022/770

[18] D. Ghosh, A. Gupta, A. Reddy, J. Fu, C. M. Devin, B. Eysenbach, S. Levine, Learning to reach goals via iterated supervised learning, in: International Conference on Learning Representations, 2021, pp. –.
URL https://openreview.net/forum?id=rALA0Xo6yNJ

[19] S. Emmons, B. Eysenbach, I. Kostrikov, S. Levine, Rvs: What is essential for offline RL via supervised learning?, in: International Conference on Learning Representations, 2022, pp. –.
URL https://openreview.net/forum?id=S874XAIpkR-

[20] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, I. Mordatch, Decision transformer: Reinforcement learning via sequence modeling, in: M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, J. W. Vaughan (Eds.), Advances in Neural Information Processing Systems, Vol. 34, Curran Associates, Inc., 2021, pp. 15084–15097.
URL https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf

[21] K.-H. Lee, O. Nachum, M. S. Yang, L. Lee, D. Freeman, S. Guadarrama, I. Fischer, W. Xu, E. Jang, H. Michalewski, I. Mordatch, Multi-game decision transformers, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, Vol. 35, Curran Associates, Inc., 2022, pp. 27921–27936.
URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b2cac94f82928a85055987d9fd44753f-Paper-Conference.pdf

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 6000–6010.

[23] Z. Zhang, A. Zhang, M. Li, H. Zhao, G. Karypis, A. Smola, Multimodal chain-of-thought reasoning in language models (2023). arXiv:2302.00923.

[24] S. Levine, A. Kumar, G. Tucker, J. Fu, Offline reinforcement learning: Tutorial, review, and perspectives on open problems (2020). arXiv:2005.01643.

[25] A. Kumar, A. Zhou, G. Tucker, S. Levine, Conservative q-learning for offline reinforcement learning, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 1179–1191.
URL https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf

[26] I. Kostrikov, A. Nair, S. Levine, Offline reinforcement learning with implicit q-learning, in: International Conference on Learning Representations, 2022, pp. –.
URL https://openreview.net/forum?id=68n2s9ZJWF8

[27] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, N. Fiedel, Palm: Scaling language modeling with pathways (2022). arXiv:2204.02311.

[28] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever,

D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[29] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, R. Lowe, Training language models to follow instructions with human feedback, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, Vol. 35, Curran Associates, Inc., 2022, pp. 27730–27744.
URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

[30] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, A. Joulin, Emerging properties in self-supervised vision transformers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 9650–9660.

[31] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, R. Girshick, Masked autoencoders are scalable vision learners, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 16000–16009.

[32] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. Ryoo, G. Salazar, P. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, B. Zitkovich, Rt-1: Robotics transformer for real-world control at scale (2022). arXiv:2212.06817.

[33] A. Kumar, R. Agarwal, X. Geng, G. Tucker, S. Levine, Offline q-learning on diverse multi-task data both scales and generalizes (2023). arXiv:2211.15144.

[34] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 10684–10695.

[35] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, W. Zaremba, Hindsight experience replay, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017, pp. –.
URL https://proceedings.neurips.cc/paper_files/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf

[36] S. Dehaene, H. Lau, S. Kouider, What is consciousness, and could machines have it?, in: Robotics, AI, and Humanity, Springer, 2021, pp. 43–56. doi:10.1007/978-3-030-54173-6_4.
URL https://doi.org/10.1007/978-3-030-54173-6_4

[37] Y. Bengio, The consciousness prior (2019). arXiv:1709.08568.

[38] B. J. Baars, A Cognitive Theory of Consciousness, Cambridge University Press, Cambridge, MA, 1988.

[39] Y. LeCun, A path towards autonomous machine intelligence, OpenReview Archive Direct Upload (2022). arXiv:2022-06-27.
URL https://openreview.net/pdf?id=BZ5a1r-kVsf

[40] A. Bardes, J. Ponce, Y. LeCun, Vicreg: Variance-invariance-covariance regularization for self-supervised learning, in: ICLR, 2022, pp. –.

[41] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1587–1596.
URL https://proceedings.mlr.press/v80/fujimoto18a.html

[42] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, 2nd Edition, The MIT Press, 2018, http://incompleteideas.net/book/the-book-2nd.html.

[43] K. Nguyen, M. Samim, End-to-end autonomous driving using reinforcement learning, 9th Semester project, Aalborg University (2022).

[44] H. Sompolinsky, Lectures on reinforcement learning (2017).

[45] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[46] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola, Dive into Deep Learning, Cambridge University Press, 2021, https://d2l.ai.

[47] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[48] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals, and Systems 2 (4) (1989) 303–314. doi:10.1007/BF02551274.
URL https://doi.org/10.1007/BF02551274

[49] K. Hornik, Multilayer feedforward networks are universal approximators, Neural Networks 4 (2) (1991) 251–257. doi:10.1016/0893-6080(91)90009-T.
URL https://www.sciencedirect.com/science/article/pii/089360809190009T

[50] H. v. Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, p. 2094–2100.

[51] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: Combining improvements in deep reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, AAAI Press, Palo Alto, California, USA, 2018, pp. 3215–3222.
URL https://aaai.org/papers/11796-rainbow-combining-improvements-in-deep-reinforcement-learning/

[52] T. Schaul, D. Horgan, K. Gregor, D. Silver, Universal value function approximators, in: Proceedings of the 32nd International Conference on Machine Learning, Vol. 37 of Proceedings of Machine Learning Research, 2015, pp. 1312–1320.
URL https://proceedings.mlr.press/v37/schaul15.pdf

[53] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[54] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, CoRR abs/1409.1259 (2014). arXiv:1409.1259.
URL http://arxiv.org/abs/1409.1259

[55] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: Beyond efficient transformer for long sequence time-series forecasting, in: The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference, Vol. 35, AAAI Press, 2021, pp. 11106–11115.

[56] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
URL https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[57] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training, arXiv preprint arXiv:1801.06146 (2018).
URL https://arxiv.org/abs/1801.06146

[58] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, arXiv preprint arXiv:1901.07285 (2019).
URL https://arxiv.org/abs/1901.07285

[59] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[60] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, Journal of Machine Learning Research 21 (140) (2020) 1–67.
URL http://jmlr.org/papers/v21/20-074.html

[61] E. Perez, F. Strub, H. de Vries, V. Dumoulin, A. Courville, Film: Visual reasoning with a general conditioning layer, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, AAAI Press, Palo Alto, California, USA, 2018, pp. 3942–3951, aAAI.
URL https://doi.org/10.1609/aaai.v32i1.11671

[62] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, N. de Freitas, A generalist agent, Transactions on Machine Learning ResearchFeatured Certification (2022).
URL https://openreview.net/forum?id=1ikK0kHjvj

[63] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, L. Fan, Vima: General robot manipulation with multimodal prompts, arXiv preprint arXiv: Arxiv-2210.03094 (2022).

[64] S. Cai, Z. Wang, X. Ma, A. Liu, Y. Liang, Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction, CoRR abs/2301.10034 (2023).
URL https://doi.org/10.48550/arXiv.2301.10034

[65] R. Balestriero, M. Ibrahim, V. Sobal, A. Morcos, S. Shekhar, T. Goldstein, F. Bordes, A. Bardes, G. Mialon, Y. Tian, A. Schwarzschild, A. G. Wilson, J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, M. Goldblum, A cookbook of self-supervised learning (2023). arXiv:2304.12210.

[66] Y. Sun, S. Ma, R. Madaan, R. Bonatti, F. Huang, A. Kapoor, SMART: Self-supervised multi-task pretraining with control transformers, in: The Eleventh International Conference on Learning Representations, 2023, pp. –.
URL https://openreview.net/forum?id=9piH3Hg8QEf

[67] J. Fu, A. Kumar, O. Nachum, G. Tucker, S. Levine, D4rl: Datasets for deep data-driven reinforcement learning (2021). arXiv:2004.07219.

[68] D. Ha, J. Schmidhuber, Recurrent world models facilitate policy evolution, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 31, Curran Associates, Inc., 2018, pp. –.
URL https://proceedings.neurips.cc/paper_files/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf

[69] S. Singh, R. L. Lewis, A. G. Barto, J. Sorg, Intrinsically motivated reinforcement learning: An evolutionary perspective, IEEE Transactions on Autonomous Mental Development 2 (2) (2010) 70–82. doi:10.1109/TAMD.2010.2051031.

[70] D. Hafner, J. Pasukonis, J. Ba, T. Lillicrap, Mastering diverse domains through world models, arXiv preprint arXiv:2301.04104 (2023).

[71] L. Zhang, G. Yang, B. C. Stadie, World model as a graph: Learning latent landmarks for planning, in: International Conference on Machine Learning, PMLR, 2021, pp. 12611–12620.

[72] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, Journal of Machine Learning Research 11 (2010) 3371–3408.

[73] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym (2016). arXiv:1606.01540.

Appendix A. GCSL Experiments

Appendix A.1. Transformer experiments

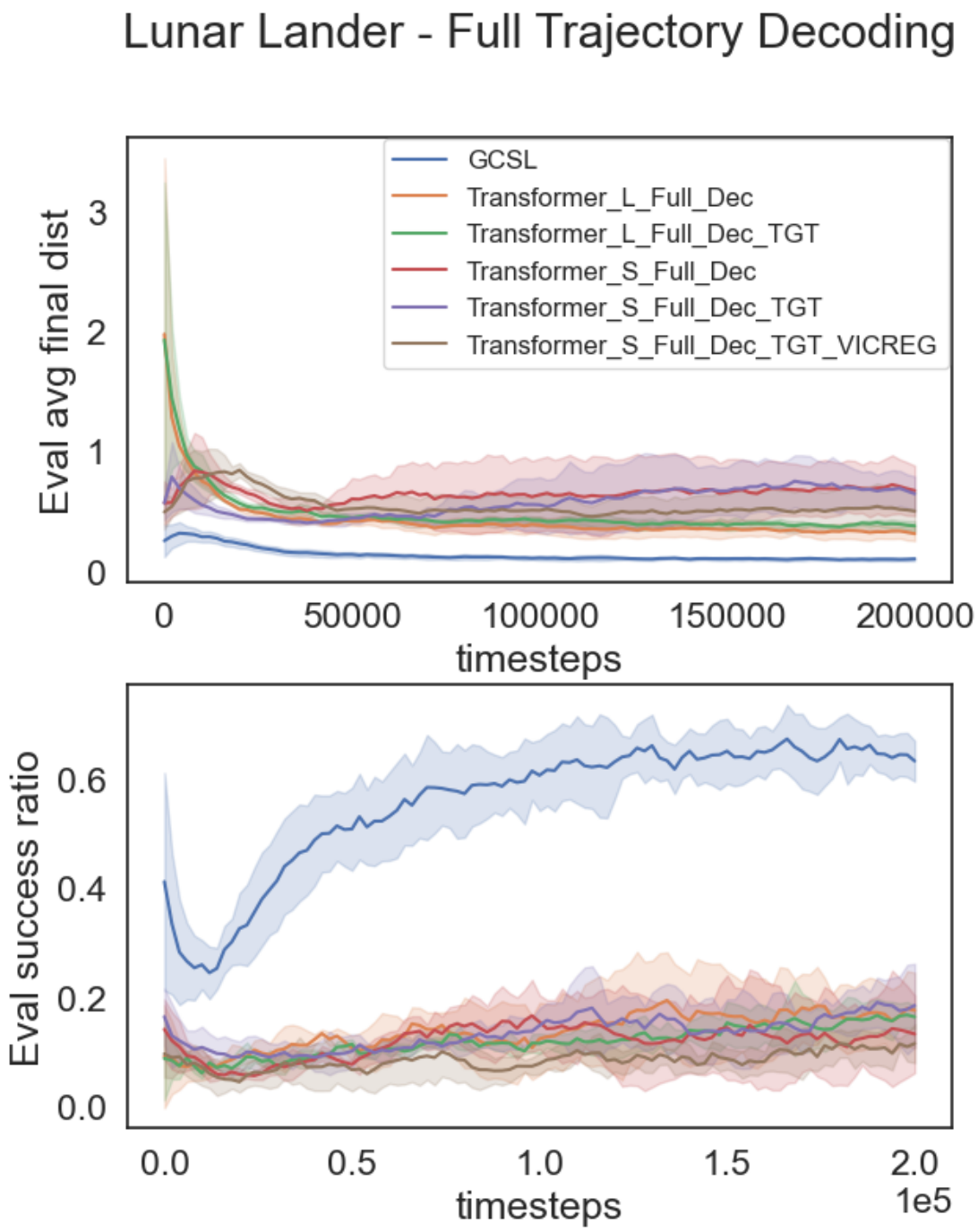


Fig. A.19: Applying the decoder-only transformer with causal and hindsight masking. L denotes a larger model (more parameters), as compared to size S. Full denotes that every action at every time step is being decoded. TGT denotes the use of a target network, similar to the notion of target network in DQN [1]. VICREG denotes, that variance and covariance are applied to decorrelate the embeddings. GCSL is the baseline comparison. Top figure: y-axis denotes the final distance to the goal state. The x-axis denotes the number of training steps. Bottom figure: the y-axis denotes the Evaluation success ratio, that is the proportion of rollouts that end up within a predetermined distance of the goal state, the x-axis denotes the number of training steps. Overall, the results show that the default proposed decoder-only transformer regardless of model size and regularization methods does not work in the online setting.