VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
STUDIJŲ PROGRAMA: INFORMATIKA

Transferring Knowledge for Reinforcement Learning Domains via Generative Models

Apmokytų žinių perdavimas tarp skatinamųjų modelių naudojant generatyvius modelius

Master's thesis

By: Ilja Jurcenko
VU email: ilja.jurcenko@mif.stud.vu.lt

Supervisor: P'Ship Assoc. Prof. Rokas Masiulis
Reviewer: Dr. Linas Petkevičius

Vilnius
2022

Santrauka

Idėja, kad mes mokomės sąveikaujant su mūsų aplinka, tikriausiai pirmiausiai atsiranda, kai galvojame apie mokymosi procesą. Kai kūdikis žaidžia, moja rankomis ar žiūri, jis neturi aiškaus mokytojo, bet turi tiesioginį ryšį su savo aplinka. Naudojant šį ryšį gaunama daug informacijos apie veiksmo priežastį ir pasekmes ir apie tai, ką daryti siekiant tikslų. Visą gyvenimą žmogus mokosi iš sąveikos su aplinka. Nesvarbu, ar mokomės vairuoti automobilį ar parengti pokalbį, mes puikiai suprantame, kaip mūsų aplinka reaguoja į tai, ką mes darome, ir siekiame koreguoti savo veiksmus priklausomai nuo atgalinio aplinkos ryšio ir tai leidžia keisti ne tik mūsų supratimą, bet ir mūsų pasaulį. Mokymasis iš sąveikos yra pagrindinė idėja, kuria grindžiamos beveik visos mokymosi ir intelekto teorijos. Vienas iš tokių teorijų yra skatinamasis kompiuterinis mokymas.

Dėl to, kad modelis yra apribotas vienos aplinkos aibėje, atsiranda klausimas, dėl savo ar eksperto patirties panaudojimo iš jau išspręstų uždavinių. Todėl, šis darbas bando nagrinėti skatinamojo mokymo problematika iš sekos apdorojimo paradigmos. Toks uždavinio formulavimas leidžia panaudoti perkėlimo strategijas ir mokytis ir skirtingos aibės žinių. Išeina, kad modelis tampa plečiamas, todėl panaudoti eksperto žinias ir integruoti į savo mokymą iš kitų uždavinių gali būti naudinga. Naujas metodo nagrinėjimas yra aktualus, nes gali būti taikomas įvairiose srityse, kur įprastai naudojamas skaistinamasis kompiuterinis mokymas: roboto-technikoje, medicinoje, chemijoje, ekonomikoje, inžinerijoje.

Abstract

Reinforcement learning is a very rapidly growing field of research, where a lot of new innovations, state-of-the-art methods and technics are emerging because of never ending challenges for the artificial intelligence. RL gives an opportunity for researchers to investigate how an agent can learn a specific task by simply interacting with the environment. Step by step, after each positive and negative feedback, after millions or trillions of trials and errors in a closed environment, as a result the agent becomes an expert in the field of training. Consequently, the agent is capable to outperform human mind in a restricted domain.

From this follows a problem about context switch. After learning to complete one task the agent needs to be retrained from the start again on the new environment. But is there a way to make an agent generalize the knowledge from previous challenges? To answer the question, the RL problem is transitioned to sequence analysis problem. Thesis emphasizes the problem and revolves around the reinforcement learning, transferring and deep learning of sequence analysis.

Contents

1. Introduction

Artificial Intelligence is one of the most important topics of research of the 21st century. Robotics, medicine, chemistry, physics, economy, engineering – it is only a tip of an iceberg when it comes to application of AI in our modern age. Due to increase of the computational capability of an ordinary computer, it is now possible to construct sophisticated AI models and apply them to solve even more complex problems. One of such problems to solve are computer and board games from academical standpoint. In this paper, the term game AI is considered from the academic perspective. The main question is why it is important to apply AI in games? Let us look at the history of game AI.

The history of game AI starts from a battle between a human mind and a computer, which had taken place in 1997, where Garry Kasparov was fighting against an AI opponent called Deep Blue on the battlefield of chess board [Dec97]. One of the reasons for creating AI in computer games is competition, it serves as an engine for research. The role and utility of "competitions" in driving and evaluating AI research seems to be getting increased acceptance in recent years [Dec97].

In addition, the confrontation on the GO board in 2016 between world champion Lee Sedol and AlphaGo. The game of GO has long been viewed as the most challenging of classic games for AI due to its enormous search space and the difficulty of evaluating board positions and moves [Shm+16]. From development of AlphaGO researchers came up with a novel combination of supervised learning from human expert games, and RL from games of self-play [Shm+16]. The games in general serve as a simulation in which researchers can hone their skills of applying AI models for a specific problem, and to learn more about AI while searching for a solution.

Furthermore, not only board games were the center of attention in research, but also computer games. First deep learning model to successfully learn control policies directly from high-dimensional sensory input using RL was created in 2013, which was implemented in seven Atari 2600 games [Mks+13]. The research found that it outperformed all previous approaches on six of the games and surpassed a human expert on three of them [Mks+13]. Knowledge obtained from solving Atari games was implemented later in development of self-driving cars.

Finally, computer games served as a steppingstone for collaboration between different AI systems. The recent research on topic of AI team play was successfully applied in Dota 2 and StarCraft 2 games On April 13th, 2019, OpenAI Five became the first AI system to defeat the world champions teams of 5 at an esports game Dota 2 [BBC+19]. In the same year, AlphaStar AI system was used to play against players all around the world and became better than 99,8% of officially ranked human players

in the game of StarCraft 2 [VBC+19]. The communication between AI systems in StarCraft 2 took place between different layers of AI in order to achieve victory in sophisticated real time strategy.

The research achievements of the AI in computer games show that this field of study is very peculiar and significant for the development of intelligent systems. The controllable environment provides ease of tuning, experimenting, and evaluating the AI models. The models are training on different environments, which have contrasting rules**. The main research topic** of this paper revolves around the question of how well models from distinct environments can be transferred.

2. Problem Definition

To proceed the topic of research, the core of the thesis should be defined. **The problem**: transfer learning in the field of deep RL agents.

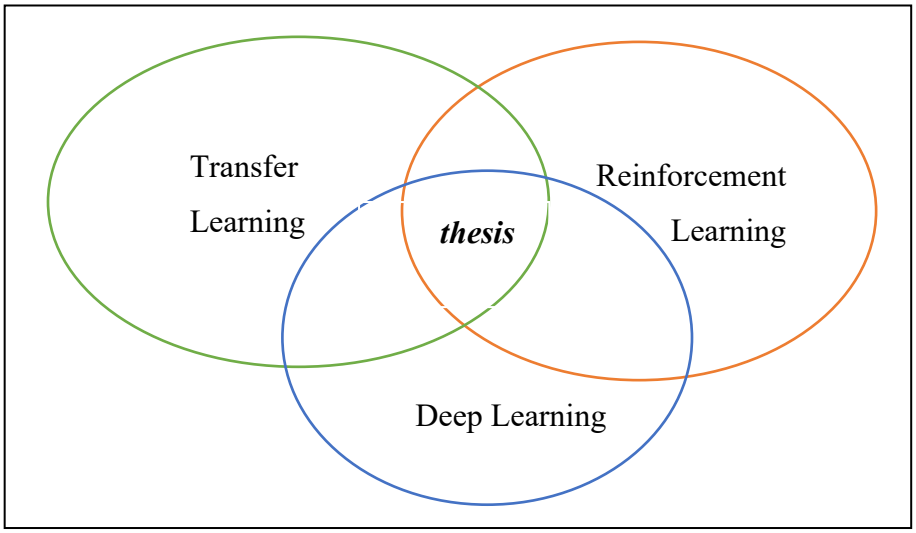


Figure 1. Scope of the thesis [ZLZ09].

As the Figure 1 suggests the main topic of this paper lies in the fields of RL, TL and DL. The problem with transfer learning is that it is hard to find what exactly needs to be transferred and how it effects results. To proceed with the research, a brief description of each related field must be given.

A typical RL problem can be considered as training an agent to interact with an environment that satisfies the criterion of a Markov Decision Process (MDP) [Roz82] [Zlz09]. From each interaction with the MDP, the agent starts with an initial state and performs an action accordingly, which yields a reward to guide the agent actions. Once the action is taken the MDP transits to the next state by following the underlying transition dynamics of the MDP. The agent accumulated the time-discounted rewards along its interactions with the MDP [Zlz09]. In short, the agent learns about the environment by taking actions and receives output from the environment as a reward, the end goal of the agent is to improve the quality of the rewards in the system it interacts. To train the agent to operate in the

MDP special algorithms are used such as Q learning (QL), Deep Q Learning (DQL), Double Deep Q Learning (DDQL) and Actor Critic methods (A2C) [Sb16].

TL on the other hand, operates on the source and target domains. Given a set of source domain and a target domain, TL aims to learn an optimal policy for the target domain, by leveraging exterior information from source domain, as well as interior information from target domain [ZLZ09]. In other words, by choosing source and target models some information, configuration or a policy is moved from the source to the target model to improve the target domain.

Despite the success of RL in solving games, there is a problem that the agents share, they fail to generalize, due to overfitting on the environments [Pag19].

3. Research Description

The research object is representation of RL problem as a sequential decision-making problem to apply the successful generalization strategies of the NLP to the RL problems**.**

The research goal is to train a target model to play Atari 2600 game, based on similar Atari 2600 source RL environments, where the agent will use generative model to compute proper action on target domain while training on offline knowledge space of Markov trajectories from source and target environments, instead of using real time environment feedback loop. The expected outcome is to quantify the impact of discrete source environment data and justify the impact on the target domain.

The research tasks:

- To analyze the theoretical parts of RL, generative models, and TL.
- To train the generative model in RL domain, with parameter tuning in a specific environment.
- To train the generative model with mixture of source and target data.
- To fine-tune the new model, evaluate the results and approaches, as well as quantify the benefits of pretraining and most impactful parameters.

The novelty of approach is to merge natural language processing solutions with RL tasks. The generative model is a Transformer [Vsp+17], which has capability to generalize from big sequencies of data. It is a special sequence to sequence neural architecture relying entirely on an attention mechanism to draw global dependencies between input and output [Vsp+17]. The input to the Transformer will be trajectories generated from source models. As a result, generative model will be tuned to interact within the target game. Trajectory in RL is a sequence of states and actions of a given agent.

4. Reinforcement Learning

RL is a subfield of ML, which addresses the problem of automatic learning of optimal decisions over time [Lap18]. In other words, it is a branch of ML, which learning occurs via interplay with an environment. Also, RL is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment [Klm96]. Consequently, it is a discipline of learning, where an agent can learn from success and failure, from reward and punishment [Rnd16].

In general, RL revolves around the idea of capturing the most important aspects of the real problem in the specific domain to achieve a long-term goal. This goal is stimulated by reinforcement technic, a feedback of reward signals. The essential part of learning in RL problems is to map situations to actions to maximize a numerical reward signal [Sb16]. These kinds of problems are identified as *closed-loop* problems, where the actions influence its later inputs [Sb16]. The agent must discover which actions yield the most reward by trying them out and not by having direct instructions. The primary characteristics of RL problems are closed-loop, not having direct instructions, consequences of actions, which play out over extended time period. Clearly, such an agent must be able to sense the state of the environment, take actions, and must have a primary goal. Sensation, action and goal are building blocks of RL models.

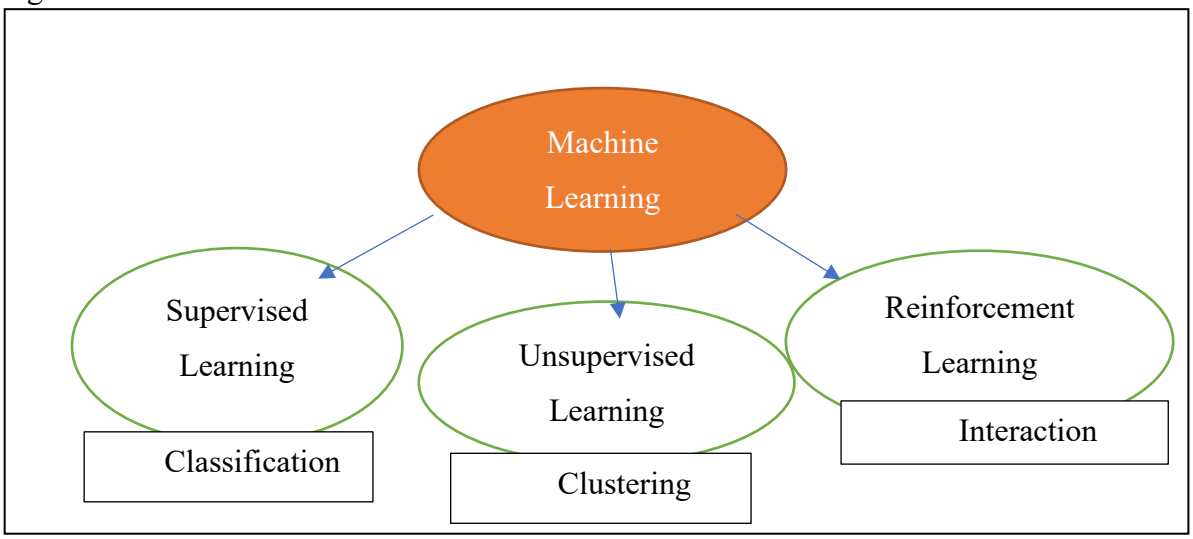


Figure 2. Branching of disciplines.

In contrast with supervised learning (Figure 2), where an agent learns from a knowledgeable external teacher, which is represented by labeled examples, the agent is capable to solve interactive problems [Sb16]. Often it is impractical to label all the states to appropriate actions by some external supervisor and to maintain correctness and representation across of all the situations. As the [Sb16] states – in uncharted territory – an agent must be able to learn from its own experience. Finally, the supervised method is used to map examples of input and desired output, and we want to learn how to

generate the output for some future, currently unseen inputs [Lap18]. In other words, the agent in supervised learning tries to extrapolate or generalize the knowledge in the already seen data for the new instances of data. RL uses such technics as well to apply them for approximation of states and rewards, but the learning is enforced by experience [Rnd16].

In addition, RL could be falsely identified as a kind of unsupervised learning. Because unsupervised learning is about finding structure hidden in collections of unlabeled data, while the RL is trying to maximize a reward signal instead [Sb16]. Uncovering the structure in an agent's experience can certainly be useful in RL, but by itself does not address the reinforcement learning agent's problem of maximizing a reward [Sb16].

RL is the third camp and lays somewhere in between full supervision and a complete lack of predefined labels [Lap18]. On the other hand, it uses many well-established methods of supervised and unsupervised learning such as DL methods.

4.1. RL Application

In order to understand RL as a structured system of learning, some examples and practical implications can be formulated. Firstly, RL has been used to solve the "curse of dimensionality problem" [Jjw17]. It is the problem where the increase of features leads to exponential growth for data samples to distinguish the target function. The problem is stated as solving high-dimensional partial differential equations, which are combined as a control theory problem and the gradient of the unknown solution is approximated by neural networks using RL, where gradient is acting as the policy function [Jjw17]. In addition, the author [Sb16] stated that the ability of some RL methods to learn with parameterized approximators addresses the classical "curse of dimensionality" in operations research and control theory. Furthermore, the RL methods are applied to optimize neural network parameters and search for best network configuration [Zl16].

RL has many examples. For instance, RL system can be associated with dopamine system in a brain. The biological limbic system of our brain has a reinforcement mechanics as a dopamine system, which toggles positive feedback to our brains and stimulates pleasure as a result [Sdm98]. Computer games, financial trading, web navigation, robotics are practical implications of RL methods. For understanding RL system in practice its core components should be defined.

4.2. RL Components and Relations

4.2.1. Reward

Reward is the building block of RL system. It is the bridge, which connects environment and agent together, giving the agent ability to comprehend the cause of its actions. This kind of feedback is called a reinforcement [Rnd16]. It is the mechanism, which is very similar to the experiences of pleasure and pain in human brain. The magnitude of the reward depends on the current state of the environment and action taken. For example, the biological human body receives rewards from of the brain based on its current hunger level, the psychological state, and the state of the environment that the subject operates in (temperature, safety and more) and action chosen (eat, sleep, drink and more). The key aspect of the agent is to maximize the reward in the long run (Equation 1). The only way the agent can influence that is to choose proper actions and yield higher rewards. The actions that agent picks form so called *policy* or in other words it is a mapping from perceived states of the environment to actions to be taken when in those states (in psychology would be called a set of stimulus-response rules or associations) [Sb16]. The agent's job is to find a policy such that maximizes some long-run measure of reinforcement [Klm96].

$$G_t = R_{t+1} + R_{t+2} + \cdots = \sum_{k=0}^{\infty} R_{t+k+1} \quad (1)$$

4.2.2. Value function

The reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in the long run.

$$V(s) = \mathbb{E}[\, G | S_t = s \,] \quad (2)$$

In contrast, the rewards determine the immediate, values indicate the long-term desirability [Sb16]. In other words, it is a problem of a delayed gratification. For example, a dilemma with a delayed gratification was studied in social experiment with marshmallows serving as a reinforcement model in preschools [Smp96]. The candidates, who neglected current reward, received more in the future, showing that receiving negative feedback in current situation could be a winning strategy. This refined and farsighted judgment could be expressed as a value function, which takes the state as an argument and gives its value, based on values of future states. The value of a state is the total amount of reward an agent can expect to accumulate from a given state (Equation 2) [Rnd16]. RL methods iteratively approximate value function for every state under certain policy. As the [Sb16] author states that the most important component of almost all RL algorithms is a method for efficiently estimating value functions.

4.2.3. Environment

The environment is everything outside of the agent, the external model, which returns rewards for fired actions and its state. In other words, given a state and action the model might predict the next state and next reward. The environment can have many variations: *deterministic, stochastic, fully observable, partially observable, discrete, continuous, episodic and non-episodic, single and multi-agent* [Klm96]. During the learning task the agent interacts with the model of the environment. The model (if it is given beforehand) is used for planning – any way of deciding on a course of action by

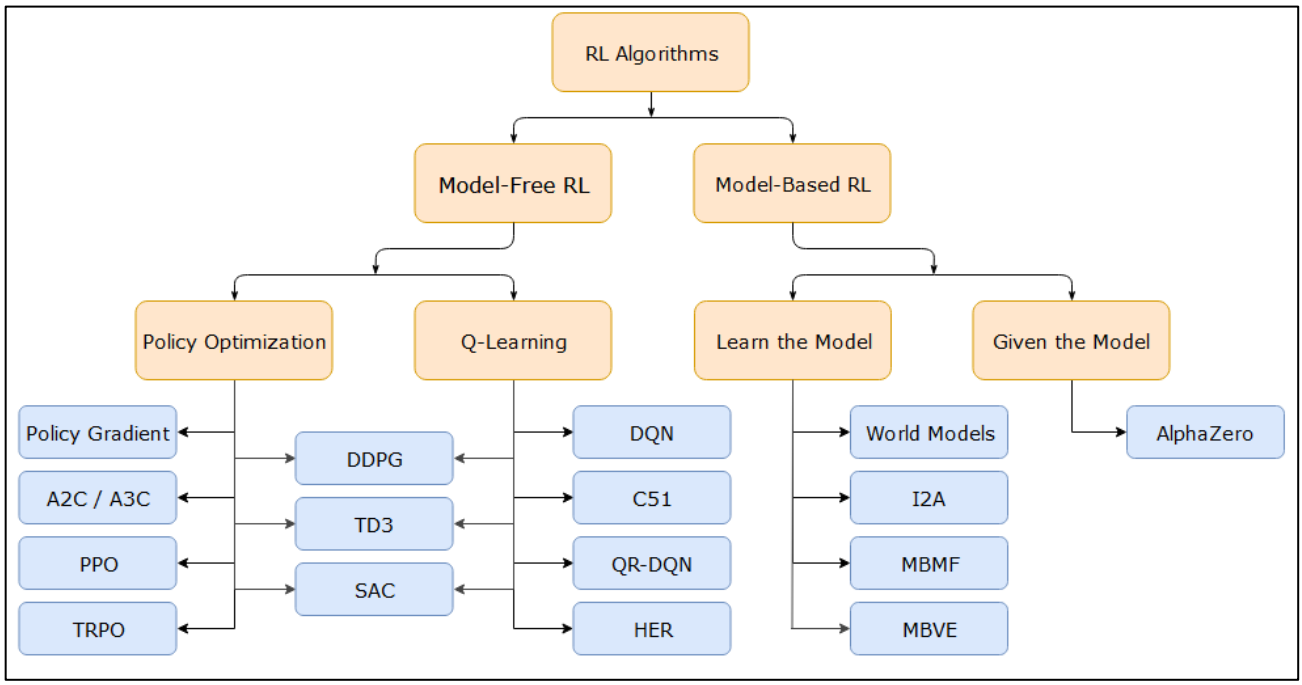


Figure 3 RL methods model-based, model-free
https://spinningup.openai.com

considering possible future situations before they are experienced [Sb16]. This strategy is called model-based learning. RL methods based on environment model split into two categories model-based and model-free (Figure 3). Model-free deep reinforcement learning algorithms have been shown to be capable of learning a wide range of robotic skills, but typically require a very large number of samples to achieve good performance [Nkf+17]. Model-based algorithms, in principle, can provide for much more efficient learning, but have proven difficult to extend to expressive, high-capacity models such as deep neural networks [Nkf+17]. In other words, model-free methods are simpler, only requiring agent to understand the environment explicitly by trial-and-error.

4.3. Discount Factor

To enable agent to control the foresightedness a *discount factor* should be added inside the accumulated rewards (Equation 3).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3)$$

The discount factor decides how much importance we give to the future rewards and immediate rewards. The value of the discount factor lies within *[0…1]* interval. If gamma equals to 1, then return just equals a sum of all subsequent rewards and corresponds to the agent with perfect visibility of any subsequent rewards and corresponds to the agent with perfect visibility [Lap18]. On the other hand, if the gamma value is close to 0, then the agent considers only the immediate rewards – short-sightedness. As a result, a discount factor gives control over the importance of reaching a sub-goal or of learning to reach the main objective.

4.4. Markov Decision Process

Markov Decision Process (MDP) can be expressed as a tuple of 4 elements (***S, A, P_a, R_a***).

- ***S*** is a set of available states that the agent can be in
- ***A*** is a set of actions that an agent can choose from
- $\boldsymbol{P_a(s, s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)}$ a transition probability, which is the probability of moving from one state s to another s' by performing some action *a.*
- $\boldsymbol{R_a(s, s')}$ is a probability of getting a reward from transitioning from a state s to another s'

$$\tau = (s_0, a_0, s_1, a_1, \dots) \quad (4)$$

The name MDP refers to the fact that the system obeys the Markov property – transitions only depend on the most recent state and action, and no prior history. In addition, a trajectory is a sequence of states and actions in the world (Equation 4).

4.5. Explore or Exploit problem

The exploration-exploitation dilemma occurs when the agent needs to find an optimal policy by choosing between the two strategies. The agent must *exploit* what it already knows to obtain reward, but it also must *explore* in order to make better action selections in the future [Sb16]. The problem is that neither exploration nor exploitation can be pursued exclusively without failing at the task [Sb16]. In other words, agents must try each action empirically to progressively favor most appealing strategy. If the environment is stochastic the agent must approximate the expected reward by exploiting every action. Exploitation is the right thing to do to maximize the expected reward on the one step, but exploration may produce the greater total reward in the long run [Sb16]. To solve this dilemma several exploration strategies could be applied: epsilon-greedy policy, softmax exploration, upper confidence bound algorithm, Thomson sampling technique and optimistic initialization of the states [Lap18].

4.6. Learning loop

For the agent to start approximating and playing with the environment, the main reinforcement learning algorithms are wrapped around learning loop, where the agent collects environment's response (states and rewards) inside its memory, while using exploration or exploitation strategies. This technic allows agent after reaching so called threshold value of the transitions, to sample randomly the states from memory inside a batch, which is used as an input for the RL algorithm.

4.7. RL algorithm

The main objective of RL algorithm is to find an optimal policy $\pi: S \mapsto A$, which includes best reward accumulation during the experiment.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{5}$$

This property can be expressed in several value functions under certain policy of learning. The value function as stated above described the value of a state, whereas the function can be adjusted as a state action pair $Q: S \times A \longmapsto \mathbb{R}$, which is formulated as a quality function $Q(s, a)$.

$$Q(s, a) = \mathbb{E}[\, G | S_t = s, a_t = a] \tag{6}$$

For finding such an optimal policy a technique of a dynamic programming can be used to approximate the value function (Equation 7).

$$NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate] \tag{7}$$

In addition, the equation 6 can be viewed as an interpolation rule between the target value and the starting point and expressed as follows (Equation 8).

$$NewEstimate \leftarrow (1 - StepSize) * OldEstimate + StepSize * Target \tag{8}$$

Furthermore, an update rule could be formulated as an interpolation between a previous state and a target state and greedily choosing the most rewarding actions. As a result, the update rule will be called a temporal difference learning rule, where Q greedy function update (Equation 9) is called *Q-learning* [Wd92] and *Sarsa* as state value iteration [Ss96] (Equation 10).

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha (r_t + \gamma \max_{a} Q(s_{t+1}, a)) \tag{9}$$

$$V(s) \leftarrow (1 - \alpha)\, V(s_t) + \alpha * (r_t + \gamma V(s_t)) \tag{10}$$

4.8. Q-learning

The quality of a state and action is updated using Q-learning algorithm. The Q function or value function can be implemented as a simple table of states. In addition, the environment's states and rewards can be processed by deep neural network, when the environment state space is too large. The

neural network provides the program with the ability to generalize from its experience, so that in new states it selects moves based on information saved from similar states faced in the past, as determined by its network [Sb16]. As a result, the value function adds a new parameter θ, which represents neural network weights $Q(s, a, \theta)$, in order to specify the approximation via network. As a result, a larger state space can be processed by using neural network as a function. In addition, the learning value of alfa is integrated inside the network and predefined beforehand. This configuration allows the agent to perform update of the model using its target state, which is calculated by the same model. When learning in a large environment space, it could cause instabilities in learning, may be harmful to training performance and sometimes lead to suboptimal policies. That is why, small improvement can be made by introducing a double deep q learning strategy [Hgs15]. The authors proposed an idea to modify the equation, by introducing a second neural network called target network. After some iterations, the target network is updated by original trained network (Figure 4). The authors of the paper proposed choosing actions for the next state using the trained network but taking values of Q from the target net [Hgs15].

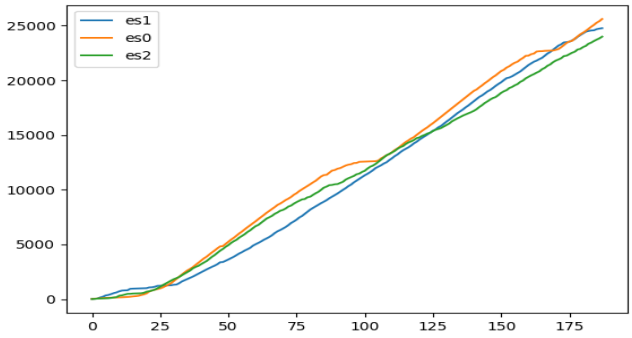


Figure 4 Results obtained from training on the Cart Pole environment, by launching small experiments by thesis author. Y-axis score, X-axis epoch. Es0 – DDQL, Es1- DQN, es2 – DDQL with target update on every iteration. Orange line (DQN) is compared with Blu line(DQN).

The examples above were practically implemented and results gathered using a Cartpole environment (https://gym.openai.com/envs/CartPole-v0/). Figure 4 represents accumulation of

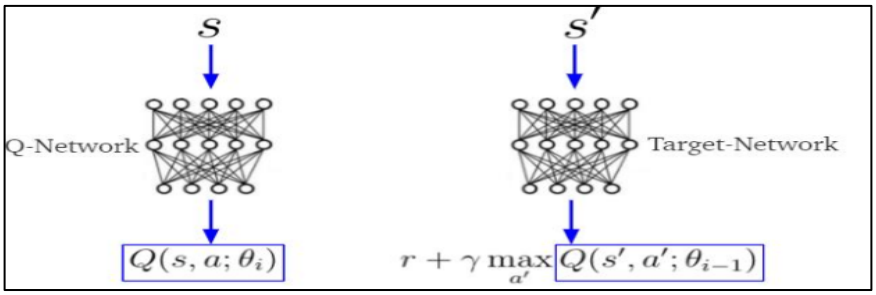


Figure 5. A dueling network architecture for double Q learning approach, where final quality value is calculated [Hgs15].

rewards obtained during the training of the agent. Figure 5 depicts the target network usage. From the graph es0 represents DDQL algorithm, es1 simple DQL, while es2 DDQL with target update on every iteration. All the runs used epsilon decay strategy for the explore and exploit problem. The orange line had the higher reward values during training epochs, which shows that the double network architecture can improve RL agent training performance.

5. Transfer Learning in RL

The previous sections described core components of the RL paradigm and algorithms. Using RL learning models as a source and target domains, the question about improving the performance by similar model is defined. The core idea of transfer is that experience gained in learning to perform one task can help improve learning performance in a related, but different, task [Ts09]. In other words, the transfer learning goal is to apply knowledge from previous environments to learn faster and more efficiently our target domain. Given a set of source domain M_s and a target domain M_t, TL aims to learn an optimal policy π^* for the target domain, by leveraging exterior information D_s from M_s as well as interior information D_t from M_t and inject both inside a deep learning model $\varphi_{dl}(D_s \sim M_s, D_t \sim M_t)$ [Zlz09].

5.1. Transfer Measurement

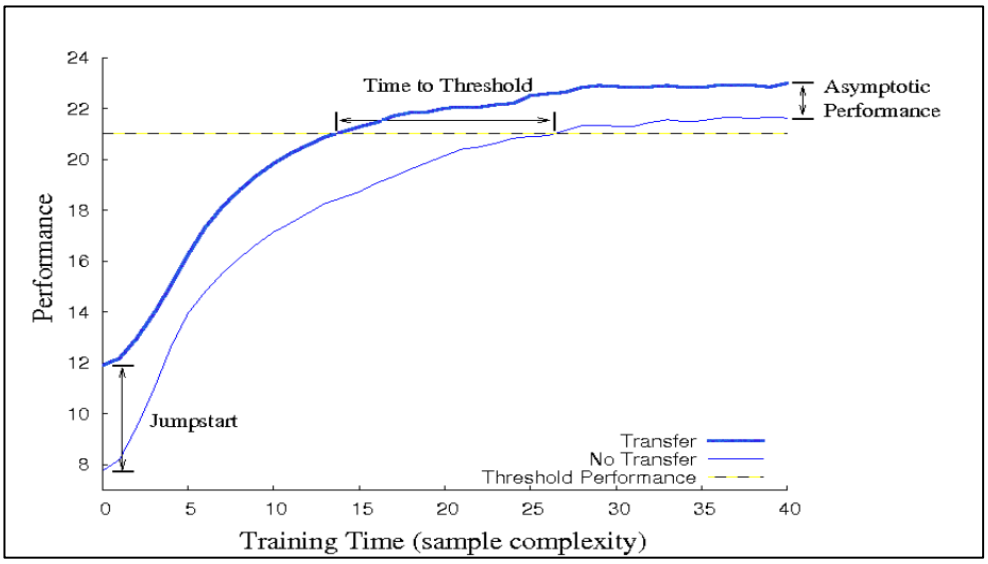
For the transfer learning approach to be useful, it needs to answer certain questions:

- What are the goals of the transfer method? By what metric(s) will success be measured?
- What assumptions, if any, are made regarding the similarity between the tasks?
- How does a transfer method identify what information can/should be transferable?
- What information is transferred between tasks?

A key challenge in TL research is to define evaluation metrics, precisely because there are many possible measurement options (Graph 1). First, in order to evaluate the performance of a transfer learning method several metrics are considered [Ts09].

1. *Jumpstart*: The initial performance of an agent in a target task may be improved by transfer from a source task (Graph 2).
2. *Asymptotic Performance*: The final learned performance of an agent in the target task may be improved via transfer (Graph 2).
3. *Total Reward*: The total reward accumulated by an agent, may be improved if it uses transfer, compared to learning without transfer.
4. *Transfer Ratio*: The ratio of the total reward accumulated by the transfer learner and the total reward accumulated by the non-transfer learner.

5. *Time to Threshold*: The learning time needed by the agent to achieve a pre-specified



Graph 1. Many different metrics for measuring TL are possible. This graph show benefits to the jumpstart, asymptotic performance, time to threshold, and total reward (the area under the learning curve) [Ts09].

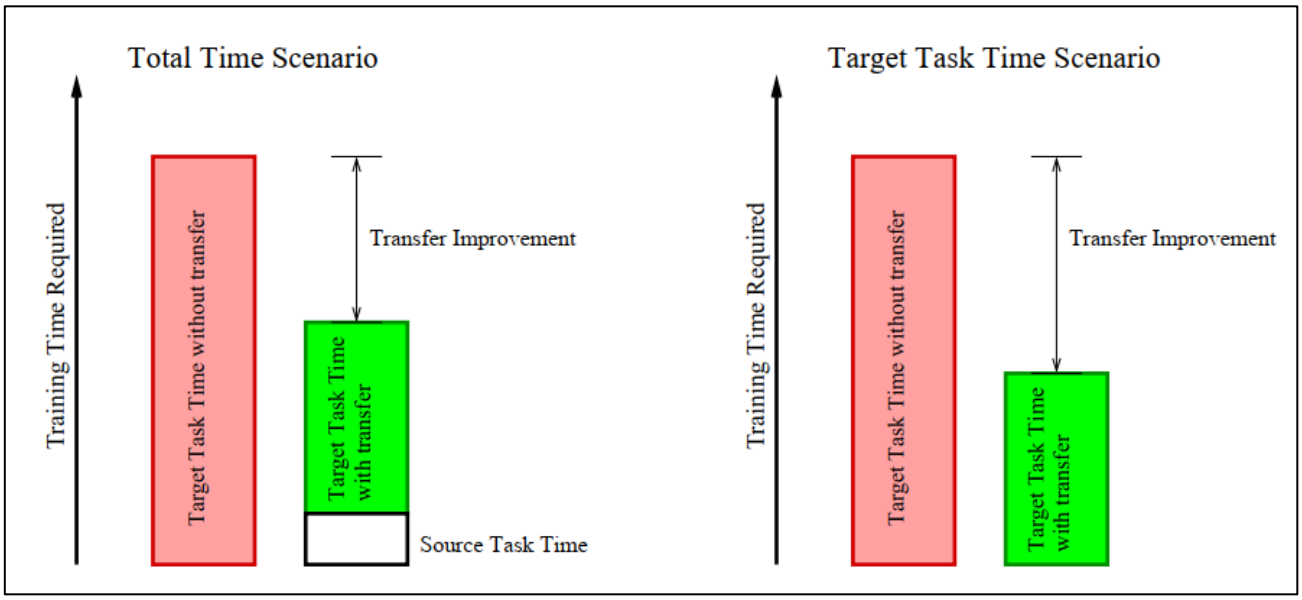


Figure 6. Successful TL methods may be able to reduce the total training time (left). In some scenarios, it is more appropriate to treat the source task time as a sunk cost and test whether the method can effectively reuse past knowledge to reduce the target task time.[Ts09]

performance level may be reduced via knowledge transfer (Graph 1).

In addition, the training time can be used as another parameter for the transfer performance measurement. To successfully transfer the knowledge, the agent would have to learn the entire sequence of tasks faster than if it had spent its time learning the final target task directly (Figure 6).

5.2. TL Categories

As the evaluation criteria is defined, it is important to distinguish several TL approaches. In order to generalize different TL methods main variance along algorithm approaches should be defined. The problem of transferring the knowledge from contrasting models has lots of configurable parameters and alternatives of the approach. The main dimensions for modifications are:

1. *Task difference assumptions*: What assumptions does the TL method make about how the source and target are allowed to differ? Examples of things that can differ between the source and target tasks include different system dynamics (i.e., the target task becomes harder to solve is some incremental way), or different sets of possible actions at some states. Such assumptions define the types of source and target tasks that the method can transfer between. Allowing transfer to occur between less similar source and target tasks gives more flexibility to a human designer in the human-guided scenario. In the fully autonomous scenario, more flexible methods are more likely to be able to successfully apply past knowledge to novel target tasks [Ts09].
2. *Source task selection*: In the simplest case, the agent assumes that a human has performed source task selection (the human-guided scenario), and transfers from one or more selected tasks. More complex methods allow the agent to select a source task or set of source tasks. Such a selection mechanism may additionally be designed to guard against *negative transfer*, where transfer hurts the learner's performance. The more robust the selection mechanism, the more likely it is that transfer will be able to provide a benefit. While no definitive answer to this problem exists, successful techniques will likely have to account for specific target task characteristics. [Ts09]
3. *Task Mappings*: Many methods require a mapping to transfer effectively: in addition to knowing that a source task and target task are related, they need to know how they are related. Inter-task mappings are a way to define how two tasks are related. These mappings could be implemented as a bridge or an adapter between two domains. If a human is in the loop, the method may assume that such task mappings are provided; if the agent is expected to transfer autonomously, such mappings must be learned. Different methods use a variety of techniques to enable transfer, both on-line (while learning the target task) and offline (after learning the source task but before learning the target task). Such learning methods attempt to minimize the number of samples needed and/or the computational complexity of the learning method, while still learning a mapping to enable effective transfer. [Ts09]
4. *Transferred Knowledge*: What type of information is transferred between the source and target tasks? This information can range from very low-level information about a specific task (i.e.,

the expected outcome when performing an action in a particular location) to general heuristics that attempt to guide learning. Different types of knowledge may transfer better or worse depending on task similarity. For instance, low-level information may transfer across closely related tasks, while high-level concepts may transfer across pairs of less similar tasks. The mechanism that transfers knowledge from one task to another is closely related to what is being transferred, how the task mappings are defined, and what assumptions about the two tasks are made. [Ts09]

5. *Allowed Learners*: Does the TL method place restrictions on what RL algorithm is used, such as applying only to temporal difference methods? Different learning algorithms have different biases. Ideally an experimenter or agent would select the RL algorithm to use based on characteristics of the task, not on the TL algorithm. Some TL methods require that the source and target tasks be learned with the same method, other allow a class of methods to be used in both tasks, but the most flexible methods decouple the agents' learning algorithms in the two tasks. [Ts09]

TL algorithms can be considered to differ along these five dimensions, but most important dissimilarity comes from the specific knowledge transferred. In further sections, examples of contrasting methods (sequential tasks, reward shaping, demonstrations) by transferable knowledge will be defined.

5.3. Sequence of Tasks

One of the examples of TL learning was presented in an experiment, where the agent was trained to balance a pole on a cart by changing the dynamics of the environment. As a result, presenting a TL framework [Ssb85]. The authors [Ssb85] use a learning scheme previously developed and analyzed to achieve performance through reinforcement and extend it to include changing to new requirements. For example, the length or a mast of the pole can change, the bias of the force, its strength, and so on. In this way, authors explore the learning system's ability to adapt to changes and to profit from a selected training sequence, both of which are of obvious utility in practical robotics applications [Ssb85]. The agent was first trained on a long and light pole, afterwards, when the task was successfully accomplished, the balances of the pole were made harder. The results of the experiment were that the total time training on the sequence of tasks was faster, than training on the hardest task directly.

Furthermore, [Ant+94] authors presented an experiment, where a mobile robot learned to shoot a ball into the goal by using vision-based RL. All the information about the changes of the environment

is only the image captured from a single TV camera mounted on the robot. To get towards the goal, the agent was taught incrementally, progressively to solve the problem of scoring maximum points. For instance, the authors began to learn to shoot the ball by setting the ball and the robot near the goal. After many iterations of success and failures, after the robot finally managed to score the points, the authors [Ant+94] placed the robot slightly further from the goal and repeat the robot learning again [Ant+94].

These tactics of solving incrementally simple tasks to achieve a bigger one is a way of transferring knowledge in the RL domain.

5.4. Reward Shaping

Reward shaping, derived from behavioral psychology, is a popular way of including prior knowledge into the learning process in order to alleviate this problem [Bhs+15]. It provides a learning agent with extra intermediate rewards, much like a dog trainer would reward a dog for completing part of a task [Bhs+15]. This extra reward can enrich a sparse base reward signal, providing the agent with useful gradient information (Equation 11). This shaping reward *F* is added to the environment's reward *R* to create a new composite reward signal that the agent uses for learning [Bhs+15].

$$R_f(s, a, s') = R(s, a, s') + F(s, a, s') \quad (11)$$

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \quad (12)$$

Shaping function *F* as the difference between two potentials. The function *F* is a defined as a difference between two $\Phi(s)$, or so-called potentials, where the value comes from the knowledge of expertise and evaluates the quality of a given state [Bhs+15]. Rather than taking all the information from the source or the sub tasks, the TL approach of reward shaping takes into consideration only the reward signals from the previous environments [Zlz09]. The approach provides an extra reward signal that is added to the environment's reward, making the agent learn on the combined signal. In addition to learning on the environment feedbacks, RS learns a reward-shaping function to generate auxiliary rewards, provided that the additional rewards contain prior knowledge to guide the agent for better action choices [Zlz09]. As a result, by reward shaping the agent learns its policy in a modified MDP where the only difference is the reward function [Zlz09].

5.5. Demonstrations

The external demonstrations from the previous similar environments can be useful for the agent to converge towards more efficient results. The [Tcs11] authors did introduce Human-Agent Transfer, an algorithm that combines transfer learning, learning from demonstration and RL to achieve rapid learning and high performance in complex domains. [Tsc11] showed that human demonstrations

transferred into a baseline policy for an agent and refined using RL significantly improve both learning time and policy performance. From this example, expert domain can vary from human to another agent performing a task with optimal or even suboptimal policy. The author [Zlz09] states that such demonstrations form a set, which each element is a tuple of transition, or so-called trajectory (s, a, s', r). This technic can be seen as a subset of Supervised Learning, in that the agent is presented with labeled training data and learns an approximation [Tsc11]. In general, learning from demonstrations is a technique to assist RL by utilizing provided demonstrations for more efficient exploration. Knowledge conveyed in demonstrations encourages agents to explore states which can benefit policy learning procedure [Zlz09]. The method is divided into offline and online depending on when the demonstrations are used. Offline - before the main update of the value function, so called pre-trained via initialization of policies and value functions with demonstrations. Online – when demonstrations are directly used in the RL stage to influence or bias the agent actions for efficient explorations [Zlz09]. However, this approach is limited by the quality of the information provided by the supervisor.

6. Generative models in RL

6.1. Transformer

Transformer is a deep neural network architecture introduced in 2017 by researchers at Google Brain [Vsp+17]. Like recurrent neural networks (RNNs), transformers are designed to process sequences such as natural language text and solve problems – machine translation, automatic abstraction. The RNN takes up the input of the data and the previous state. Because of this, the network is slow to train, and long sequence leads to vanishing gradient – the problem of long-term dependencies. The memory becomes weak to save information from old connection.

Another alternative is LSTM (long short-term memory) a special kind of RNN, which possesses special branching rules for passing previous state [Hos97]. This allows to skip unnecessary information and retain important features for a longer period. This rule helps to improve the situation of the vanishing gradient problem, but it all depends on the length of the input. If the input is increased drastically the LSTM will start to fail, introducing the same problem as with vanilla RNN.

The previous versions of sequence processing models were having problems with large scale of data. That is why, transformer was introduced. The new architecture tries to address this problem and generalize information despite huge amounts of data.

Comparing transformer with previous solutions, transformer does not require sequences to be processes in order. For example, if the input is text, the transformer does not need to process the end

of the text after processing the beginning. Because of this, transformers are easier to parallelize than RNS and can be trained faster. As well as special kind of memory mechanism allows to distinguish and hold information across long range of input data.

Going into the details, the building blocks of transformer architecture are encoder and decoder. The encoder and decoder consist of layers. The encoder layers sequentially pass the result to the next layer as its input (Figure 7). The architecture can contain both encoder and decoder, or only one of them. First, an input sequence of tokens is mapped to a sequence of embeddings, which is then passed to the encoder. Additionally, a positional information is provided. The encoder consists of a stack of blocks, each consisting of two subcomponents: a self-attention layer followed by a small feed-forward network. The layer normalization step is applied to the input of each subcomponent as well as dropout layer. The partial output of this structure is passed to the decoder stack (Figure 9).

The decoder receives a part of decoder output and the processed sequence as input. Decoder layers sequentially pass the result to the next layer along with the encoder result as its input.

Furthermore, each encoder consists of a self-attention mechanism (input from the previous layer) and a feed forward neural network with direct connection (input from the self-attention mechanism). Each decoder consists of a self-attention mechanism (input from previous layer), an attention mechanism to the encoding results (input from self-attention mechanism and encoder), and a neural network with direct connection (input from attention mechanism) (Figure 9).

Going further into details, the primary part of stack like architecture of transformers is the

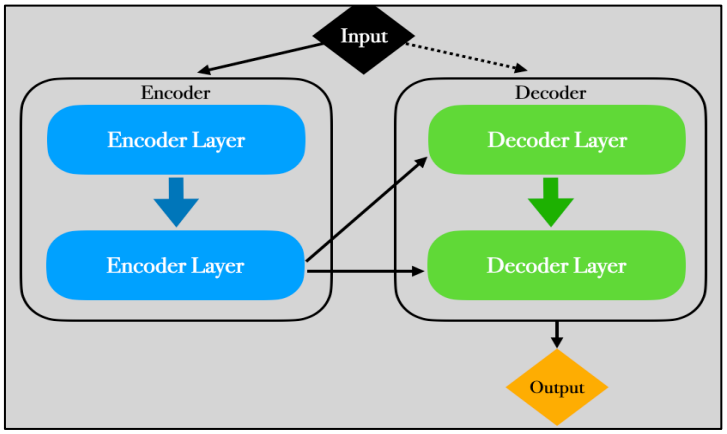


Figure 7. A top-down view of the transformer architecture [Vsp+17]

attention mechanism. From top-down perspective, it serves as a working memory that gives the ability to assess and generalize the important aspects of the information from the whole sequence. Every attention layer consists of several weight matrices: query W_q, keys W_k, and the values W_v. The

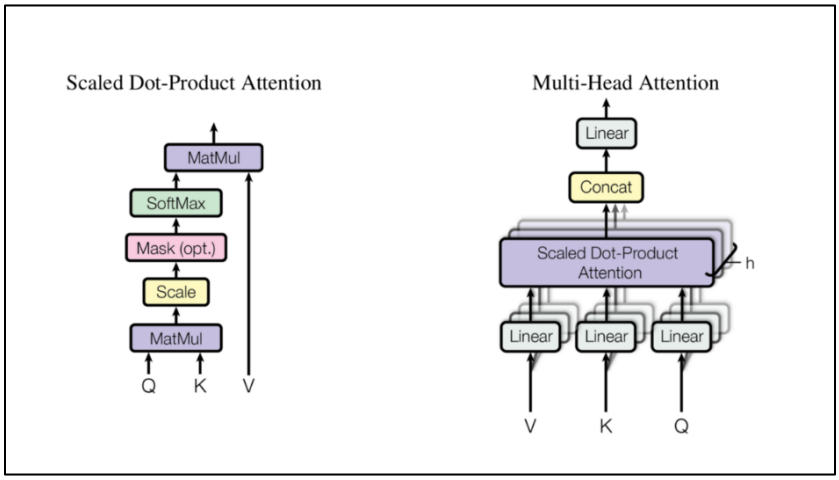


Figure 8 Attention calculation for the paper 'Attention is All You Need'. Left is scaled Dot-Product Attention. Right multi-head attention consisting of several attention layers running in parallel.

input token X is multiplied by each matrix respectively and Q, K, V values are the result. After that, the attention score is calculated as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (13)$$

From the Figure 8 and the equation 13, Q is a matrix that contains the query (vector representation of one word in the sequence), K are all the keys (vector representation of all the words in the sequence)

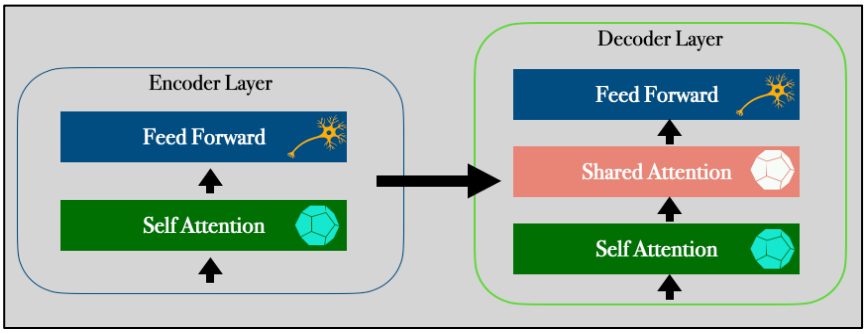


Figure 9 Building blocks of each layer [Vsp+17]

and V are the values, which are again representation of all the words in the sequence. Looking at the calculation, it can be noticed that the values in V are multiplied and summed with some attention-weights that are put inside *softmax* function, which maps distribution between 0 and 1. In other words, self-attention is a variant of attention that processes a sequence by replacing each element by a weighted average of the rest of the sequence. Consequently, the attention can be expressed as relevance between queries and keys described as a dot product (Equation 13).

Basically, if the input token is the size of 512, and the number of tokens is 9 (length of the sentence), and the number of heads is 8, then each attention head receives 9x64 (512 / 8) dimensional vector. The results of attention are spanned across the input sequence, by using the selected query and

multiplying it by each corresponding *K* of each token. The division by the square root of dimension key is used in order achieve easier gradient propagation. The gathered scores are multiplied by the value and finally the results are summed up to pass to the further layers. Finally, the attention calculation is scaled across multiple attention heads, leading to a vector of corresponding matrices and values. Introduction of heads gives the ability to comprehend the sequence in several dimensions and extrapolate more information.

The Figure 11 represents a bird eye view on the multi-head attention between different layers. The lines in each cell represent the attention from one token (left) to another (right), with line weight proportional to the attention value (ranges from 0 to 1). A more zoomed in example of such model is presented in Figure 10. In this example the input consists of two sentences: “the cat sat on the mat”

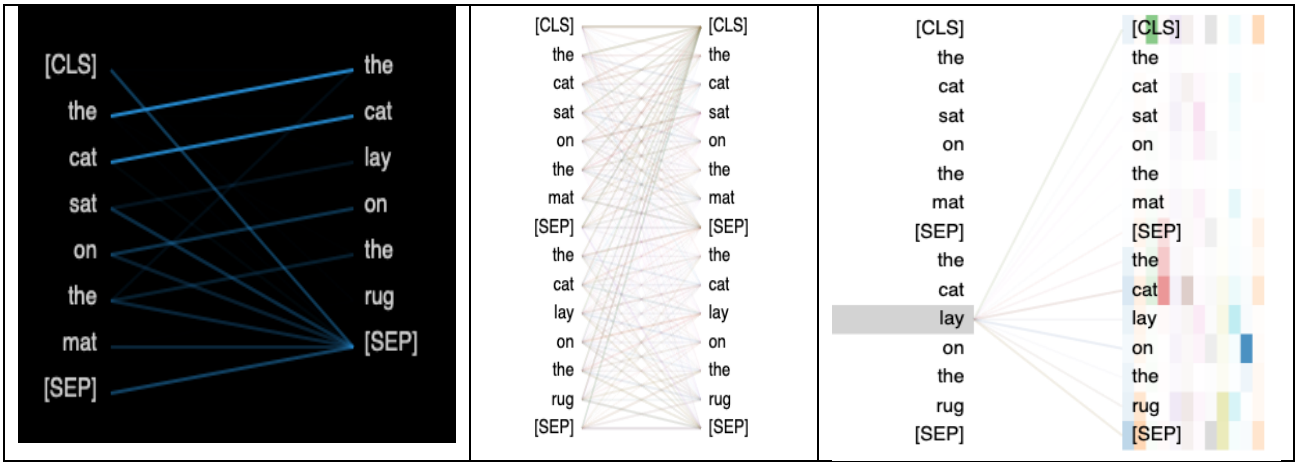


Figure 10 Example of visualization of the attention heads between different tokens. The left is the input to output, middle and right is the visualization of combination of input and output. The right one is the visualization of the selected word only. From https://github.com/jessevig/bertviz.

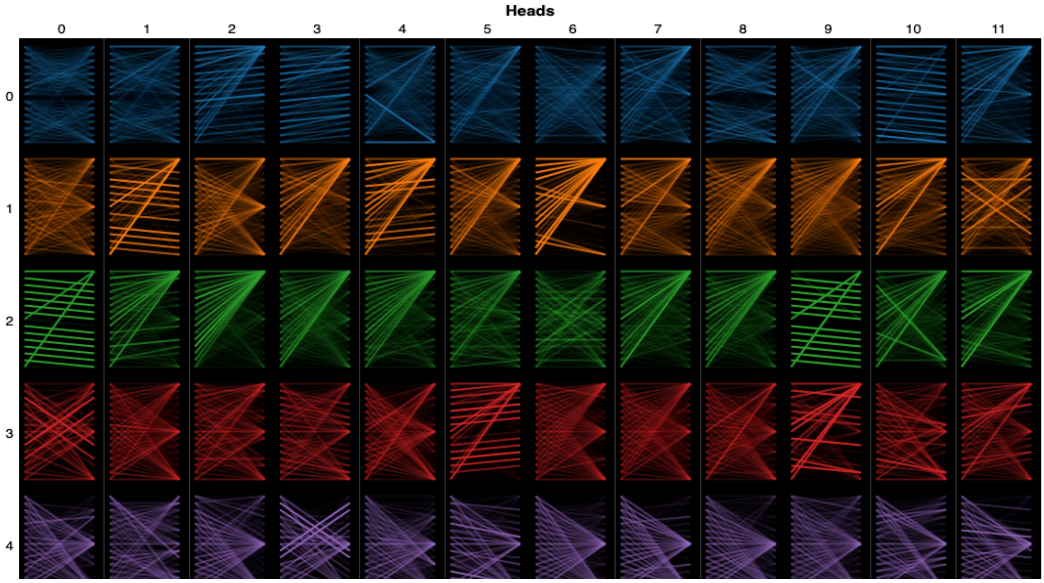


Figure 11 the view of attention throughout the model. The rows are the layers and the heads as the columns. From https://github.com/jessevig/bertviz

and “the cat lay on the rug”. The [SEP] and [CLS] symbols are special kind of symbols to indicate a sentence boundary and [CLS] is a symbol appended to the front of the input that is used for classification tasks, it is used to summarize the sentence and indicate the end of the sequence.

From the figures, it can be noticed, which parts of the sentence information the model is focusing on in the given layer and the head. Roughly speaking, attention could be comprehended to assign scores for the input features based on their importance on the context. Basically, the model will pay attention to some parts of the sentence containing more valuable information for predicting the next word in the sentence. In the Figure 10, the figure on the left has more attention on the word pair of “the” and “cat” indicating that these words are more meaningful for the prediction. It is that understanding the relationship between these words might help the model determine that this is a description of a cat and grab the context.

6.2. The Width or Height

One of the problems for transformer neural models is how to choose the right architecture. In the paper [Wlj+21] the author is describing the problem of choosing the proper architecture for the transformer. As the author stated, there is a choice between the width and the height of the transformer architecture depending on the input data. The number of self-attention layers is described as the depth of the model and the dimension of the internal representation as the width, also known as number of embeddings and the vocabulary size. The main question is, does the optimal depth-to-width ratio depend on the data modality or is it a consequence of architecture expressivity considerations? In other words, does the data modality influence the best ratio for main parameters for transformer architecture. [Wlj+21] authors identify a vocabulary bottleneck in self-attention, proving that the rank of the input embedding matrix caps the network width’s contribution to expressivity.

As the use of Transformer architectures was extended to different modalities, the effect of this architectural element was overlooked. For instance, in the bioinformatics domain [Wsl+17] the RaptorX benchmark for protein long-range contact precision with a Transformer model that has width 1280, but a vocabulary size 33, equivalent to a character-level model. The experimental results [Wlj+21] indicate that this is very sub-optimal, as the width is severely capped by the low rank of the embedding matrix.

Furthermore, [Wlj+21] shows that when the width exceeds the input embedding rank, deepening is exponentially favorable over widening. The experiments show by the authors that when decreasing the vocabulary size below the value of the network width, depth becomes more important than width also depending on data modality. All in all, the fine tuning of the transformer on the data should

consider architectures which prioritize width or height, there is a possibility that in this way an optimal result could be obtained. The author [Wlj+21] empirically demonstrate the problem of not properly choosing the architecture by:

1. A degradation when the input embedding rank is smaller than the network width [Wlj+21].
2. An advantage of depth over width when the input embedding rank is smaller than the network width [Wlj+21].
3. A degradation when the network width is smaller than the internal attention representation [Wlj+21].

6.3. Decision Transformer

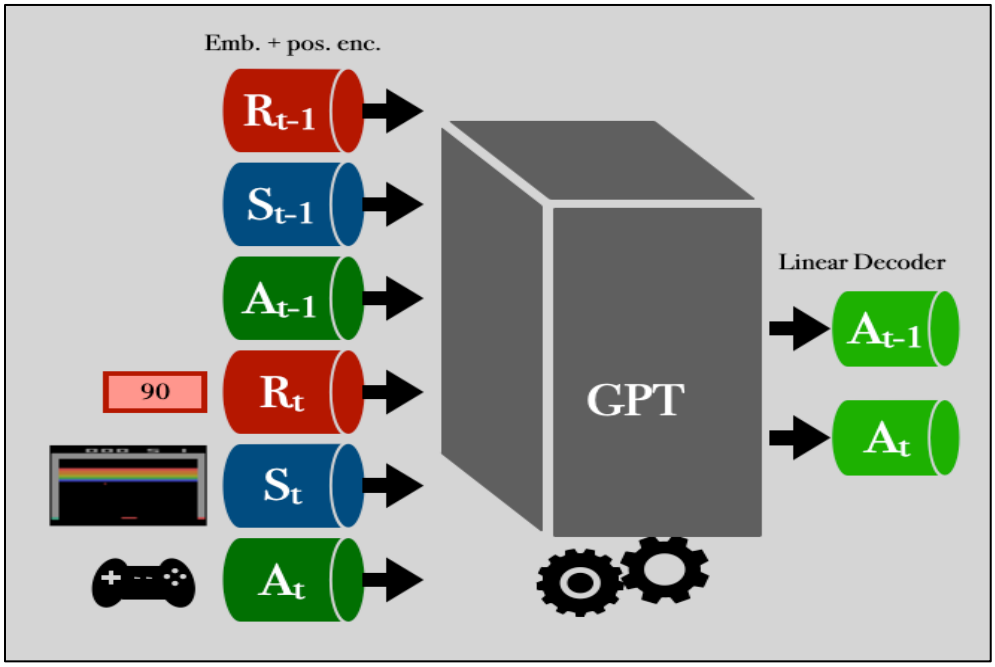


Figure 12 Architecture of a DT presented in paper [Clr+21]

To solve RL problem with a sequence modeling approach an architecture of DT was presented in recent research [Clr+21]. Figure 12 shows an example of such solution which takes as input the trajectories generated by offline RL agent environments. While conventional work in RL has utilized specialized frameworks relying on Bellman backups, [Clr+21] propose to instead model trajectories with sequence modeling, enabling to use strong and well-studied architectures such as transformers to generate behaviors. This is done in **offline RL** domain, where model is trained from a fixed dataset rather than online feedback experience in the environment. This enables to train RL policies using the NLP solutions with minimal changes.

The input trajectories are first passed to convolutional encoder for images and linear layer for continuous states. The embeddings are then processed by an autoregressive transformer model

([Clr+21 uses GPT), trained to predict the next action given the previous tokens using a linear output layer. What authors are suggesting predicting is a **desired target return** by initializing with a starting state of the environment and unrolling the sequence (the same approach with standard autoregressive generation in language models). As a result, it yields a sequence of actions to execute in the environment. On the other hand, if the model does not unroll the desired target return than, the mode is called **behavioral cloning**, which simply clones the actions and rewards of the dataset.

6.4. Approach

As stated above the main solution is to change the overall approach of the RL training algorithm. By using a generative model, the most important distinction from a casual RL algorithm with bellman equation is the **conditioning on the reward**. In this setup, the DT with given target reward, current and previous state is trying to predict the action that leads to this target reward. In contrast, the main RL algorithms such as TD and others are trying to maximize the reward by going into the target state (Equation 9). As a result, the discount factor is not needed. The suggested approach is discussed in more detail [Sch20], where author suggests turning RL task into a form of SL.

Additionally, the [Clr+21] states that it is nontrivial to model rewards since the main priority is to model based on future desired returns, rather than past rewards. That is why, instead of modeling the rewards directly, the returns-to-go are used:

$$R_t = \sum_{t'=t}^{T} r_{t'}, \quad \tau = (R_1, s_1, a_1, R_2, s_2, a_2, \dots, R_T, s_T, a_T) \tag{14}$$

The transformer is trained by using the context length parameter *K,* which defines the number of timestamps for each parameter, in total of *3K* for each parameter. To obtain token embeddings, a linear layer is presented for each input, which projects raw inputs to the embedding dimension, followed by layer normalization. For visual inputs, a convolutional encoder is used as mentioned earlier [Clr+21]. Additionally, positional embedding is added as a timestamp number. As a result, the tokens are then processed by a GPT model, which predicts action tokens via autoregressive modeling.

6.5. Credit assignment

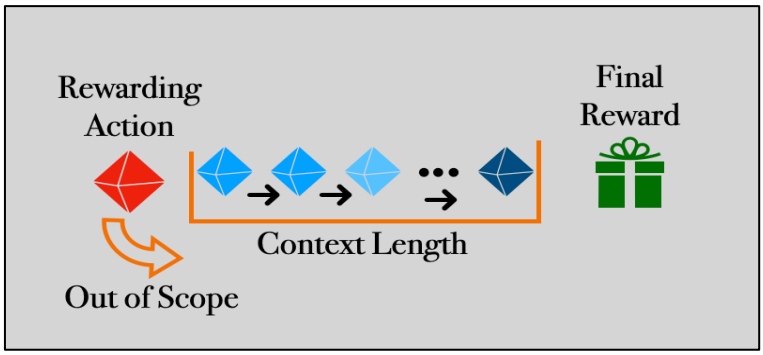


Figure 13 Example of limitation of context length of generative model.

Transformers can perform credit assignment directly via self-attention, in contrast to Bellman backups which slowly propagate rewards and are prone to "distractor" signals [Hl19]. The (temporal) credit assignment problem (CAP) is the problem of determining the actions that lead to a certain outcome (in case of RL, the higher rewards). In other words, the model is trying to evaluate the contribution of actions and to determine the most essential actions or states lead to more optimistic future. This can enable transformers to still work effectively in the presence of sparse or distracting rewards [Clr+21]. Furthermore, a transformer modeling approach can model a wide distribution of behaviors, enabling better generalization and transfer [Clr+21].

The Figure 13 shows the problem with credit assignment in transformers. The problem lies in the size of the context window, when important action or state is outside of the context length, that is why the model will miss the game changing action and continue to work with not optimal actions. The authors [Clr+21] empirically stated that the increase in the window size improves the quality of the results.

6.6. Key Takeaways

The authors [Clr+21] present important aspects of introduced architecture and several conclusions:

- Using sequence modeling the longer context length improves performance.
- DT can perform long-term credit assignment for sparse reward tasks.
- Unlike other offline RL methods, no regularization is used.
- Return-to-go conditioning avoids need for discounting.

Additionally, [Clr+21] stated, that this solves the following problems:

- Poor modeling of large distributions – catastrophic forgetting [Clr+21].
- Nonstationary cause by learning of both actor & critic networks.

The cons of this approach:

- The performance does not align with the state-of-the-art online RL solutions

7. Online Decision Transformer

The improvement of DT is presented in paper [Zlz22], where authors provide an online setup for transformer training. The team of researchers from Facebook AI Research, UC Berkeley and UCLA present an approach of improvement of offline decision transformer, by introducing data from the real time environment. This enables the transformer, or in this case an agent, get a real time feedback from the training environment allowing it to perform additional exploration and thus improving results. It incorporates both offline pretraining and online finetuning in one single framework and achieves performance competitive with state-of-the-art models on the D4RL benchmark [Fkn+20].

RL policies trained purely on offline datasets are typically sub-optimal as the offline trajectories might not have high return and cover only a limited part of the state space [Zlz22]. Online interactions are thus essential for improving model performance. Unfortunately, the introduction of online signal is not so simple and straightforward for improvement of results.

The problem is that introduction of a naïve application of offline or off-policy RL methods to the offline pre-training (simply adding an additional signal from the real time environment) and online finetuning regime often does not help, or even hinders, performance [Zlz22]. This poor performance in off-policy methods can be attributed to off-policy bootstrapping error accumulation [Klf+19]. In offline RL methods, poor performance in the online finetuning regime can be explained by excess conservatism, which is necessary in the offline regime to prevent value overestimation of out-of-distribution states [Zlz22]. Authors [Ngd+20] were the first to propose an algorithm that works well for both the offline and online training regimes [Zlz22]. Recent work, [Kft+21] proposes an expectile-based implicit Q-learning algorithm for offline RL that also shows strong online finetuning performance, because the policy is extracted via a behavior cloning step that avoids out-of-distributions actions [Zlz22].

The ODT builds on the DT architecture and incorporates changes due to the stochastic policy. The authors [Zlz22] predict the policy mean and log-variance by two separate fully connected layers at the output. The basic building blocks are described in Algorithm 1, which summarizes the overall finetuning pipeline in ODT, where the detailed inner training steps are described in Algorithm 2.

7.1. Training pipeline

The default DT architecture is then modified with several changes in training pipeline as well as the introduction of a new replay buffer, called hindsight experience replay (HER) buffer. Due to problematic online fine tuning, the generalized stochastic learning objective is introduced in ODT [Zlz22].

In the probabilistic setup, the goal is to learn a stochastic policy that maximizes the likelihood of the dataset. The authors introduce a special stochastic selection for policy or in other words extraction of next action is made probabilistic (Algorithm 1). The results are obtained and compared with

Algorithm 1: Online Decision Transformer

Input: offline data $\mathcal{T}_{\text{offline}}$, rounds R, exploration RTG g_{online}, buffer size N, gradient iterations I, pretrained policy π_θ.
Intialization: Replay buffer $\mathcal{T}_{\text{replay}} \leftarrow$ top N trajectories in $\mathcal{T}_{\text{offline}}$.
for *round* $= 1, \ldots, R$ **do**
    `// use randomly sampled actions`
    Trajectory $\tau \leftarrow$ Rollout using $\mathcal{M}$ and $\pi_\theta(\cdot|\mathbf{s}, \mathbf{g}(g_{\text{online}}))$.
    $\mathcal{T}_{\text{replay}} \leftarrow \{\mathcal{T}_{\text{replay}} \setminus \{\text{the oldest trajectory}\}\} \bigcup \{\tau\}$.
    $\pi_\theta \leftarrow$ Finetune ODT on $\mathcal{T}_{\text{replay}}$ for I iterations via Algorithm 2.

Algorithm 2: ODT Training

Input: model parameters θ, replay buffer $\mathcal{T}_{\text{replay}}$, training iterations I, context length K, batch size B
Compute the trajectory sampling probability $p(\tau) = \text{length}(\tau) / \sum_{\tau \in \mathcal{T}} \text{length}(\tau)$.
for $t = 1, \ldots, I$ **do**
    Sample B trajectories out of $\mathcal{T}_{\text{replay}}$ according to p.
    for *each sampled trajectory* τ **do**
        `// Hindsight Return Relabeling`
        $\mathbf{g} \leftarrow$ the RTG sequence computed by the true rewards: $\mathbf{g}_t = \sum_{j=t}^{|\tau|} r_j,\ 1 \leqslant t \leqslant |\tau|$.
        $(\mathbf{a}, \mathbf{s}, \mathbf{g}) \leftarrow$ a length K sub-trajectory uniformly sampled from τ.
    $\theta \leftarrow$ one gradient update using the sampled $\{(\mathbf{a}, \mathbf{s}, \mathbf{g})\}$s.

deterministic variant (Figure 14), where the stochastic policy selection obtains better results. After shifting from deterministic to stochastic policies for defining exploration objectives during the online phase, the [Zlz22] develops a novel replay buffer that stores trajectories and is populated via online rollouts from the ODT. The buffer is populated with trajectories generated from the online environment during online training episodes.

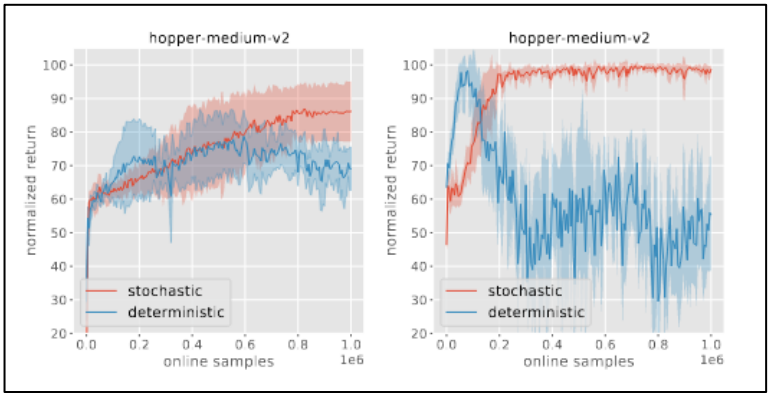


Figure 14 The results obtained by the authors [Zlz22] presents a comparison of ODT (red) with a deterministic variant (blue) in terms of training in Hopper benchmark. Left is small and Right is large architectures. For both, ODT is stable, whereas the performance of the deterministic policy declines and exhibits high variability.

Authors [Zlz22] use a replay buffer to record past experiences and update it periodically. For most existing RL algorithms, the replay buffer is composed of transitions [Zlz22]. After every step of online interaction within a rollout, the policy or the Q-function is updated via a gradient step and the policy is executed to gather a new transition for addition into the replay buffer [Zlz22]. For ODT however, the replay buffer consists of trajectories rather than transitions [Zlz22]. After offline pretraining, it initializes the replay buffer by the trajectories with highest returns in the offline dataset (Algorithm 1 line 2). Every interaction with the environment, it completely rollout an episode with the current policy, then refresh the replay buffer using the collected trajectory in a first-in-first-out manner [Zlz22]. Afterwards, update the policy and rollout again, as shown in Algorithm 1 [Zlz22]. Similar to [Hza+18] , [Zlz22] observe that evaluating the policy using the mean action generally leads to a higher return, but it is beneficial to use sampled actions for online exploration since it generates more diverse data.

7.2. Hindsight Return Relabeling

The authors present a modification for the overall reward-based replay buffer with returns-to-go as previously defined in DT. They present an extension to the returns-to-go, because during online fine tuning the overall chain of total rewards obtained from a specific state is not known. That is why, the hindsight return relabeling comes into play. It is a method for improving the sample-efficiency of goal-conditioned agents in environments with sparse rewards [Ggr+19]. The key idea here is to relabel the agent's trajectories with the achieved goal, as opposed to the intended goal [Zlz22]. In the case of ODT, we are learning policies conditioned on an initial RTG [Zlz22]. The return achieved during a policy rollout and the induced RTG can however differ from the intended RTG. Inspired by HER, [Zlz22] relabel the RTG token for the rolled out trajectory τ with the achieved returns, such that the RTG token at the last timestep $g|\tau|$ is exactly the reward obtained by the agent $r|\tau|$, see Line 6 of Algorithm 2. Like DT, Algorithm 2 uses a two-step sampling procedure to ensure that the sub-

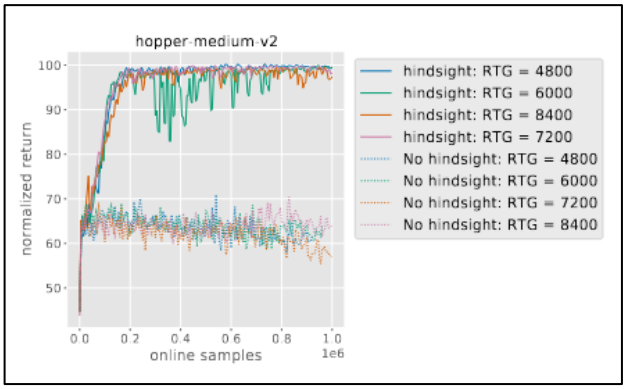


Figure 15 Results obtained by the authors [Zlz22], that indicate the improvement from previous DT, by introducing the new labeled hindsight return to go reward.

trajectories of length K in the replay buffer are sampled uniformly. The first step is to sample a single trajectory with probability proportional to its length, then uniformly sample a sub-trajectory of length *K*. This return relabeling strategy applies to environments with both sparse and dense rewards. The results obtained show, that there is an improvement in performance by introducing a new hindsight label (Figure 15).

7.3. Comparisons

In their empirical study [Zlz22] evaluated ODT against other state-of-the-art approaches for finetuning pretrained policies and investigated how the individual ODT components influence its overall performance. They compared ODT's offline performance with DT and implicit Q-learning (IQL) [Kft+21], a state-of-the-art algorithm for offline RL; and compared ODT's online finetuning performance to an IQL finetuning variant. For a purely online baseline, the team also reported the results of the soft actor critic (SAC) algorithm [Hza+18]. Empirically, it is shown that ODT is competitive with the state-of-the-art in absolute performance on the D4RL benchmark, outperforms DT and achieves nearly the same performance as IQL.

8. Transferring in Atari

This section describes different strategies that authors use to implement transferring in Atari domain. The introduction of the sequences and flipping reinforcement learning to a supervised learning domain to generalize knowledge is one of many ways to solve the problem.

8.1. Actor-Mimic

The primary approach of [Pbs15] is to spawn, create new network knowledge from already trained experts. The paper [Pbs15] views problem of training agent to play multiple environments by introducing a multitask and transfer learning. The authors [Pbs15] design a method called "Actor-Mimic" that leverages techniques from model compression to train a single multitask network using guidance from a set of game-specific expert networks [Pbs15]. The form of guidance can vary, and several different approaches are explored and tested empirically [Pbs15]. Given a set of source games, the primary objective is to acquire a policy network for multitask play on any given source game. To achieve this [Pbs15] divides networks into students and experts. For diminishing the impact of different scales of Q-values a standardization of guidance is needed. The mathematical notion of guidance is to present a squared loss that would match Q-values between the student network and the experts. The [Pbs15] authors decide to unify the output from networks of Q-values into softmax function, consequently mapping the results into one unified interval.

Intuitively, softmax is used from the perspective of forcing the student to focus more on mimicking the action chosen by the guiding expert at each state, where the exact values of the state are less important [Pbs15]. The authors call this method "Actor-Mimic" as it is an actor, i.e., policy, that mimics the decisions of a set of experts

The transferring part of this setup is when AMN weights are copied for the initialization of next DQN agent, that will be trained on the target task. This DQN is now pretrained to work on the next task. The author [Pbs15] presents a policy and feature initialization transfer strategies of defined networks.

This approach can be viewed as a pipeline of agents training iteratively to behave on a target environment based on the experts and using the previous agent as a weight initialization.

The overall transferring results indicate that AMN provides a definite increase in learning speed for 3 games of Breakout, Star Gunner and Video Pinball [Pbs15]. Authors indicate that the possible reasoning for such a performance is the source game Pong having very similar mechanics to both Video Pinball and Breakout.

8.2. Progressive Networks

Introduced recently by Google Deep Mind, it is architectural solution for neural networks [Rrd+16] for application in specific transfer learning domain. This approach considers the architecture of neural network and presents an interesting approach that is applicable for leveraging information across tasks.

The problem of transfer learning is the loss of information while finetuning on the next task. This phenomenon is called catastrophic forgetting [Kpr+17]. There is a problem, that while learning a new task a model can dispose information that is important or essential for the target task and simply forget learned experience.

The issue of transfer learning can be seen in biology as well. In marked contrast to artificial neural networks, humans and other animals appear to be able to learn in a continual fashion [Kpr+17]. Recent evidence suggests that the mammalian brain may avoid catastrophic forgetting by protecting previously acquired knowledge in neocortical circuits [Kpr+17]. When a mouse acquires a new skill, a proportion of excitatory synapses are strengthened; this manifests as an increase in the volume of individual dendritic spines of neurons [Ypg09]. Critically, these enlarged dendritic spines persist despite the subsequent learning of other tasks, accounting for retention of performance several months later [Ypg09]. When these spines are selectively "erased," the corresponding skill is forgotten [Cig+15]. This provides causal evidence that neural mechanisms supporting the protection of these

strengthened synapses are critical to retention of task performance [Kpr+17]. These experimental findings—together with neurobiological models such as the cascade model [Fda05]—suggest that continual learning in the neocortex relies on task-specific synaptic consolidation, whereby knowledge is durably encoded by rendering a proportion of synapses less plastic and therefore stable over long timescales [Kpr+17].

Taking biology into the account, the primary objective of strengthening the circuits in the brain is moved to architecture for progressive networks. The result is that the network itself is immune to forgetting and can leverage prior knowledge via lateral connections to previously learned features [Rrd+16]. Basically, it takes the approach of freezing weights for specific tasks while learning on the target domain. Catastrophic forgetting is prevented by instantiating a new neural network (a column)

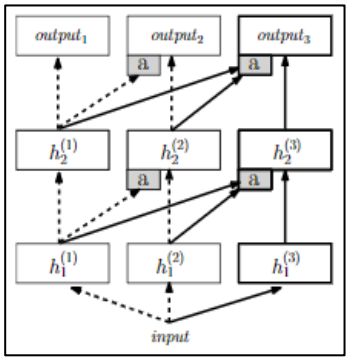


Figure 16 From [Rrd+16]. An example of progressive network consisting of three columns. The dashed arrows of first two columns on the left were trained on task 1 and 2 respectively. The grey box labelled a representation of the adapter layer. A third column is generated and added for the final task having access to all previously learned features

for each task being solved, while transfer is enabled via lateral connections to features of previously learned columns (Figure 16). From the Figure 16, the building blocks of progressive network are columns, which represent a single neural network model. Each such columns consists of L blocks (layers), with hidden activations h and lateral connections a. Each lateral connection with weights W, received input from prievious column U. The activition calculation is shown in equation 14.

$$h_i^k = f(W_i^k h_{i-1}^k + \sum_{j<k} (U_i^{k:j} h_{i-1}^j)) \qquad (15)$$

It could be noticed that first column does not have lateral connections. While training on the first task and transitioning to the second one, the weights of the first column (model, or network) are frozen. The h_0 does not have previous connections that is why the sum part of equation 14 is omitted leaving only the first step. For the incoming task k, a new column is added and lateral connections are provided. In the paper [Rdr+16] athors consider lateral connections as special adapters, which

genereliaze information presented in the previous layers of input, by introducing a dimenionality reduction.

The continuos architecture gives an ability to introduce a continual learning agent. This is possible because every block of the specific column has the information provided from the previous columns and access to the according layer index. From the Figure 16, for instance, taking the third column the output 3 block can pick information provided by $h_2^{(1)}$ and $h_2^{(2)}$. This gives an opportunity to consider the important inputs and ignore the ones that do not influence the target environment. By creating lateral adaptive connections, it allows the model to pefrorm trasferring information between the tasks and allowing to distiguish best suited input.

The practical usage of this architecture was done in several domains. The progressive networks were applied in transfering for emotinal recognition [Gka+17] and robotics, where progressive neural nets could be used to transfer between generated simulations to the real hardware [Rvr+17]. In atari domain, across the games it was observed that progressive nets result in positive transfer in 8 out of 12 target tasks, with only two cases of negative transfer [Rvr+17].

From the analysis of the results, the paper stated that progressive networks is not a silver bullet for transferring problem. The first issue is that the number of parameters of such network is very large, that is why it is parameter-wasterful. Each block requires a full layer for each column, which leads to very large scaling. Furthermore, the data feeded to the network requires labeling for specific tasks, which requires an additional step to execute. One of the most problematic issue is the sequantiality of learning. The model needs to learn tasks in a specific order (to allow the freezing of weights), and the ordering of learning affects the results.

In summary, the progressive networks provide a competative solution to the problem, by enabling the ability to distinguish information and provide correct protection of sensative source data to prevent catastrophic forgetting.

9. Object based RL

The approach of transferring can be applied by implementing a change in overall state of MDP. An integral part of every RL is MDP, where every state is encoded in some way. The shift in interpretation of state data, leads to more flexible training environment. Majority of RL algorithms in simulation domain are using an approach of parsing raw pixel data using convolutional neural networks to retrieve an essential information about the environment. Instead of using pixel array of data, the new paradigm is to use object-oriented Markov decision process (OMDP) [Dcl+08] and pass objects to the model. In this way, data will contain information about the objects that the model must

comprehend and extract knowledge from and operate on different domains. The authors [Wc18] presented a framework for the setup for object embedding network.

Current state-of-the-art DRL approaches to videogames learn directly from raw video data, using deep convolutional neural networks (CNNs) [Wc18]. Unfortunately, this approach has certain limitations. For example, many games (e.g., Starcraft) feature controllable cameras, meaning much of the game-state is obscured from the agent at any given point [Wc18]. Working around this, e.g., by putting the camera under the agent's control, adds additional complexity to the agent [Wc18]. Additionally, in many cases it may be undesirable, or even impossible to produce a video-output for the agent to consume [Wc18]. Rendering videos for multiple agents may be prohibitively expensive, and in some cases, there may be no obvious way to produce a good visual representation for NPCs (non-player characters) [Wc18]. Considering the interpretation of visual input and how agents learn, the paper [Dap+18] presented research about how humans learn to play video games and how transferring knowledge from modified environments impacts the result on the target game. The authors stated [Dap+18], in case of human players the change in textures, despite structurally the same, took twice as long to finish the game, while in comparison the performance of an RL agent was approximately the same for the modified and target game. That indicates the difference of interpretation of raw video data.

That is why, a motivation for an alternative approach rises. Comparing with other many other RL tasks, the ground-truth information about the current state of the environment is often available in videogames, although such information needs to be organized and presented to an agent in some way [Wc18]. Hence, the use of this direct information about the current game-state could be alternative to working directly on raw video data [Wc18]. Authors [Spl+15] presented a solution that has environment dependencies, a set of general features: a number of tokens collected, distance to nearby enemy and so on. Unfortunately, the instantiation of such features needs to be done prior to model as well as it has a rather strong dependency on the environment.

To overcome this problem, an object representation of the environment is presented. That means the game internals could be used as an object representation of the current state to the learning agent. In an object representation, each game-state observation is given as a list of objects, and their classes and attributes [Wc18]. For example, the state of one round of the game Pong might be represented by two objects of the class *bat*, with attributes of *x-coord, y-coord*, and player, and an object of the class ball with attributes *x-coord, y-coord, x-velocity*, and *y-velocity* [Wc18]. Most of the modern applications are written in object-oriented paradigm, that is why, this interpretation of data could be applied to any application domain.

The main advantage of object world is that agents can reuse and share the information of objects between each-other as well as the objects provide a good steppingstone for strategic thinking and planning for agent. Despite the benefits, the environment internals need to be accessible to the learning model.

The building blocks of object reinforcement learning are classes, objects, and attributes [Dcl+08]. In the OMDP domain there is a set of classes which has their own set of attributes, a set of corresponding objects [Dcl+08]. The relations between objects are mapped in a relation class, or a function. In recent studies, the object relations could be formulated as graph networks [Wmb+19].. It is an interesting solution of the problem needs future research.

The authors presented a practical solution as OEN (Figure 17). This network can not only take a list of object-feature vectors of arbitrary lengths as input to produce just a single, yet unified, fixed-length representation of all the objects within the current game-state, but also be trained on a given task simultaneously [Wc18]. Hence, OEN-based approach provides an alternative way to apply DRL algorithms within videogames, based on object information [Wc18].

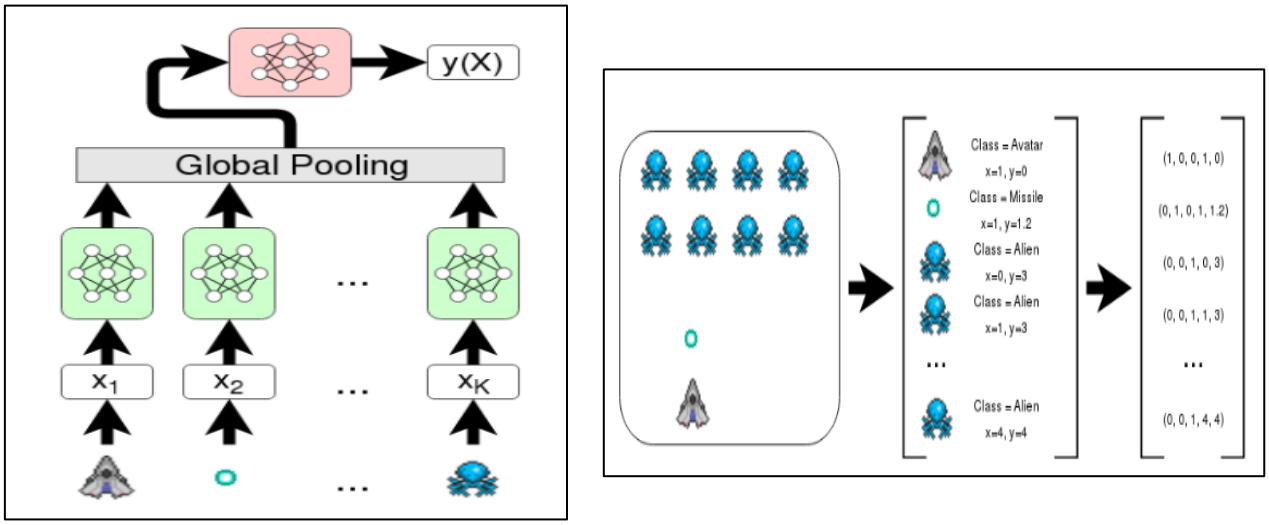


Figure 17 Object embedding network architecture on the right, on the left example of object feature extraction process [Wc18] Green networks represent embedding networks of objects and the red one is the DRL task network while global pooling is for calculating the combined environment state.

The authors use as training input the game internals, which are converted into object feature networks, presented on Figure 17 on the right. For the agent, the process of identification and extraction of objects is handled by the environment [Wc18]. That is, the object extraction can be done directly via access to the ground internals of the environment. Hence, the wide-spread use of object-oriented programming languages should help with this process, as many objects are likely to be treated as such in code [Wc18]. Thus, the description of a game state is given in an object-oriented format, that is the observation from the environment game-state is a list of objects. The process of feature

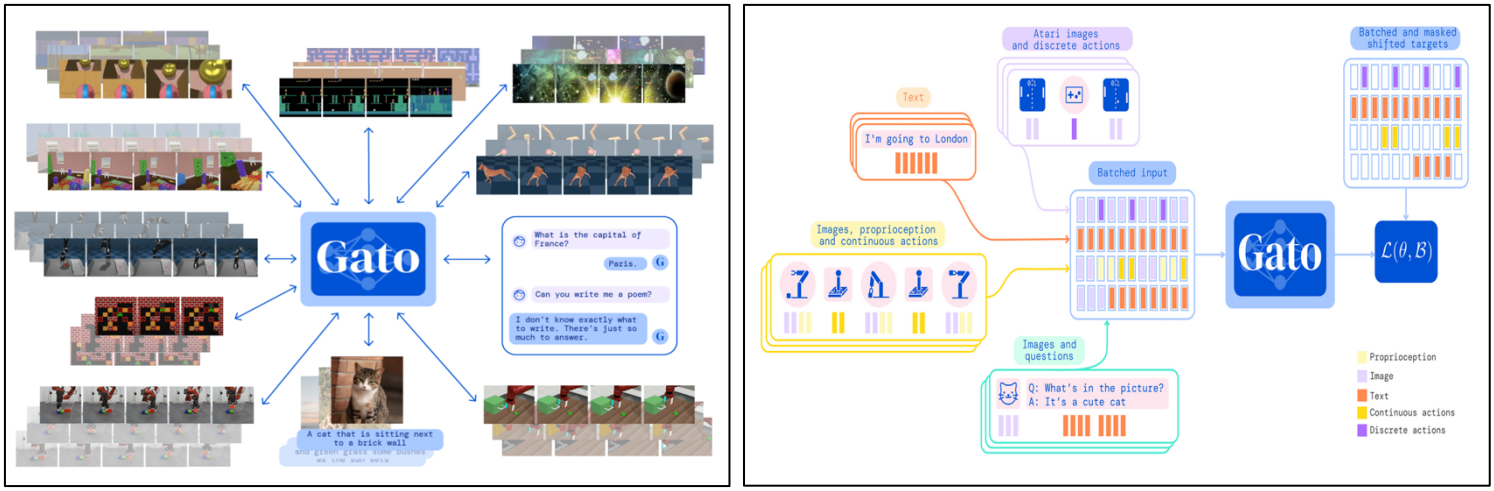
extraction needs to be performed for obtaining object feature vectors from the raw objects based on their attributes.

From the experiments [Wc18], it is observed from all the experimental results reported above that the object-based agent is capable of learning in all five games that testing was performed. Additionally, across all the games, the agent performs comparably with the feature and image representation. [Wc18] experiments demonstrate that OEN-based DRL agent can be an effective alternative to the existing agents for playing general video games.

Due to flexible object representation the naïve transferring algorithm due to simple addition of previously generated object knowledge. The transferring knowledge from one task in the setup of OEN is the feature vectors of previously selected objects, that could in theory be generalized.

10. General Task Transformer

In summary of previous section about object Markov representation, the primary bottleneck of the general models is the proper representation of the data. During the writing of the thesis, a general based task solving agent GATO was introduced by Google Deep Mind, proving that the DT transformer and overall transformer models can be generalized not only across the single ALE domain, but across text, image, and robotic data [Rzp+22]. The theoretical assumption about one model to operate on several different domain was achieved.



Example 1 The GATO general model operating on different data types on the left, on the right the top down representation of general GATO model. From https://www.deepmind.com/publications/a-generalist-agent.

The primary solution is to introduce learning only on tokenized data. Different kind of raw input (text, image, game actions, robot position) was unified by passing special tokenization step on each case. To enable processing this multi-modal data, authors serialize all data into a flat sequence of tokens. In this representation, GATO can be trained and sampled from akin to a standard large-scale

language model [Rzp+22]. During deployment, sampled tokens are assembled into dialogue responses, captions, button presses, or other actions based on the context [Rzp+22].

- Text is encoded via SentencePiece [Kr18] with 32000 subwords into the integer range [0, 32000) [Rzp+22].
- Images are first transformed into sequences of non-overlapping 16 × 16 patches in raster order, as done in [Dbk+20] Each pixel in the image patches is then normalized between [−1, 1] and divided by the square-root of the patch size (i.e. √16 = 4) [Rzp+22]..
- Discrete values, e.g. Atari button presses, are flattened into sequences of integers in row-major order. The tokenized result is a sequence of integers within the range of [0, 1024) [Rzp+22].
- Continuous values, e.g. proprioceptive inputs or joint torques, are first flattened into sequences of floating point values in row-major order. The values are mu-law encoded to the range [−1, 1] if not already then discretized to 1024 uniform bins. The discrete integers are then shifted to the range of [32000, 33024) [Rzp+22].

After the tokenization part, the model with 1.2B parameters is trained via online interactions. The training of GATO is accomplished by online interaction of the environment and the interesting feature is that previously selected actions are serving as an input to the generative model.

In ALE Atari GATO achieves the average human (or better) scores for 23 Atari games, achieving over twice human score for 11 games [Rzp+22]. While the single-task online RL agents which generated the data still outperform GATO, this may be overcome by adding capacity or using offline RL training rather than purely supervised [Rzp+22]. As operating on different tasks between ALE domain, at the same time model is showing strong performance in solving 604 distinct tasks.

Another example of multitask solver was presented again, by Google Deep Mind [Aab+22]. In this paper, the agent under the hood uses transformer architecture to generalize knowledge from different task domains. Authors strengthen the performance by using hierarchical RL, that is included for better action selection. The authors presented a peculiar approach of self-supervised learning, where agent learns from a human teacher and consequently extrapolates knowledge from human representations. Slowly the model reduces the dependency on human supervision and starts to operate on the own. The results are incredible, because the agent learns to operate on virtual environment generated by Unity3D engine – recognize the environment, answer specific questions, execute commands, observe, and exploit the environment internals (Example 2). This is a small, but firm step towards the generalization of models to solve various tasks. In this approach, the model is given a set

of trajectories generated by the human teacher and the domain of RL now slowly enters imitation learning. The raw input is presented as an image array of 92x72 array values and text-based tokens.

Example 2 The multimodal environment presented as virtual generated world with human and agent interaction using actions and text. From https://www.deepmind.com/publications/a-generalist-agent.

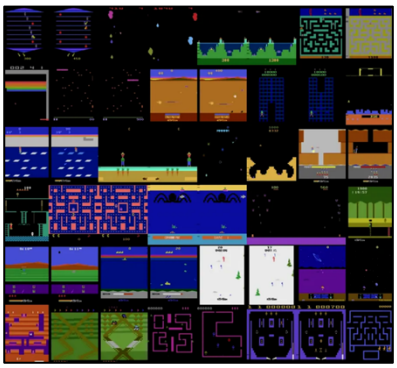
11. Practical Research

The Atari Learning Environment (ALE) is one of the most popular benchmarks with D4RL. The Atari 2600 problem set is picked due to its popularity in academia research [Hgs15], as it stands a good benchmark for the RL methods with overall number of 60 environments. Additionally, it provides with sufficient range span of simple to complex environments. Only recently, some of more complex environments such as *Montezuma Revenge* and *Pitfall* were solved, testing the novel approach of first return, then explore [Ehl+21]. Also, the Atari environments are suitable due to small frame and memory size.

The environment interpretation is perception of the input signal as a frame image without knowing anything about the Atari internals, and choosing actions like a human player, but via the programming interface. That is why, it is theoretically possible to switch the environment and use the model free nature of RL algorithms. The result of action presents itself as a new observation.

11.1. Learning dataset

For practical experiments the offline RL dataset (https://github.com/google-research/batch_rl) was used. The learning dataset is a collection of gathered trajectories from 60 Atari Environments using a standard learning DQN agent. The learning instance (trajectory) is a tuple of (state, action, reward, next state). The tuple consists of numerical data. The state is saved as a numerical representation of 160x192 frame, it is eventually resized to 84x84 as done in most of the papers [Hgs15]. The reward is a simple integer representing score, which can be normalized between all the environments using gamer score values per domain. Finally, action is a simple integer that maps to corresponding game input. The training dataset was a size of 500000 states. During the experiments different environments (21 in total) and its combinations from the Atari game collection were used: “Breakout”, “Seaquest”, “Qbert”, “Pong”, “Alien”, “Amidar”, “Assault”, “Asterix”, “Asteroids”, “BankHeist”, “BattleZone”, “BeamRider”, “Bowling”, “Boxing”, “CrazyClimber”, “Freeway”, “Frostbite”, “Gravitar”, “Kangaroo”, “MsPacman”, “PrivateEye”, “Spaceinvaders”. The



Example 3. Collection of Atari game screenshots. The primary input data of the model is the game screenshot. From https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark.

environments were picked according to the most used ones in DQL papers and referencing target environments from [Clr+21]. The total number of learning dataset was 1.7 terabyte of data.

11.2. Related Work

In order to obtain practical results, DT surfed as a baseline for experiments [Clr+21]. DT was improved to obtain results of transferring using a spectrum of environments with different action and reward space:

- The model was configured to be able to train on source and target domains. As a result, an initial transferring experiment was performed.
- The model was scaled to support training input from different kind of environments and fine-tuned on the target domains.
- The model was trained using default parameters using [Clr+21] configuration. The model architecture was slightly fine-tuned to improve the quality of results. The authors of [Clr+21] stated that there is a room for improvement, regarding the parameter tuning.

To unify the Atari game different action sizes, the vocabulary number was increased. The primary motivation for this setup was to perform the transferring experiment and create a model that could work on any given environment from set of Atari.

11.3. Training pipeline

For obtaining different results from memory heavy datasets, a training pipeline was introduced. A sequence of training on different environments, serializing the information as input due to high model RAM usage for the next process and freeing resources to be able to continue the pipeline. The DT was trained using two stages. The first phase is pretraining phase. This step launched experiments on environments to gather the source environment knowledge and the second stage was training on the target domain. The training phase was launching experiments on a parametrized number of trajectories from the offline RL datasets and at the same time on every epoch evaluating the model performance by launching the testing Atari simulation environment using “Atari.py”. The testing part was performing stationary number of episodes (referencing [Clr+21] setup, 10 testing games), or runs, on the environment and calculating the gathered score. The total score for the training epoch was the mean score across 10 testing episodes. Additionally, model was trained using 3 separate seeds and testing scores were calculated among them. The primary metric of evaluation of model performance in case of RL is the agents’ acquired score on a specified environment.

11.4. Models

During practical experiments the baseline *minGPT* model with several primary parameters was used as suggested by the author [Clr+21].

Default	
Number of layers	6
Number of attention heads	8
Embedding Dimension	128
BatchSize	128
Context Length K	30
Learning rate	6*10-4

Depth Oriented	
Number of layers	16
Number of attention heads	12
Embedding Dimension	128
BatchSize	128
Context Length K	30
Learning rate	3*10-4

Width Oriented	
Number of layers	2
Number of attention heads	32
Embedding Dimension	64
BatchSize	128
Context Length K	30
Learning rate	3*10-4

Small modifications were added, and several model versions were considered. The primary changes were to the model number of attention heads, number of layers and toggling of learning decay. As well as models that were pretrained on the source environment Pong vary in number of pretrained epochs.

11.5. Workstation

For these experiments, a workstation with 32 GB of RAM, Intel Core i7-9700K CPU 3.60GHZ and Nvidia RTX 2080 was used.

11.6. Results

11.6.1. Decision Transformer

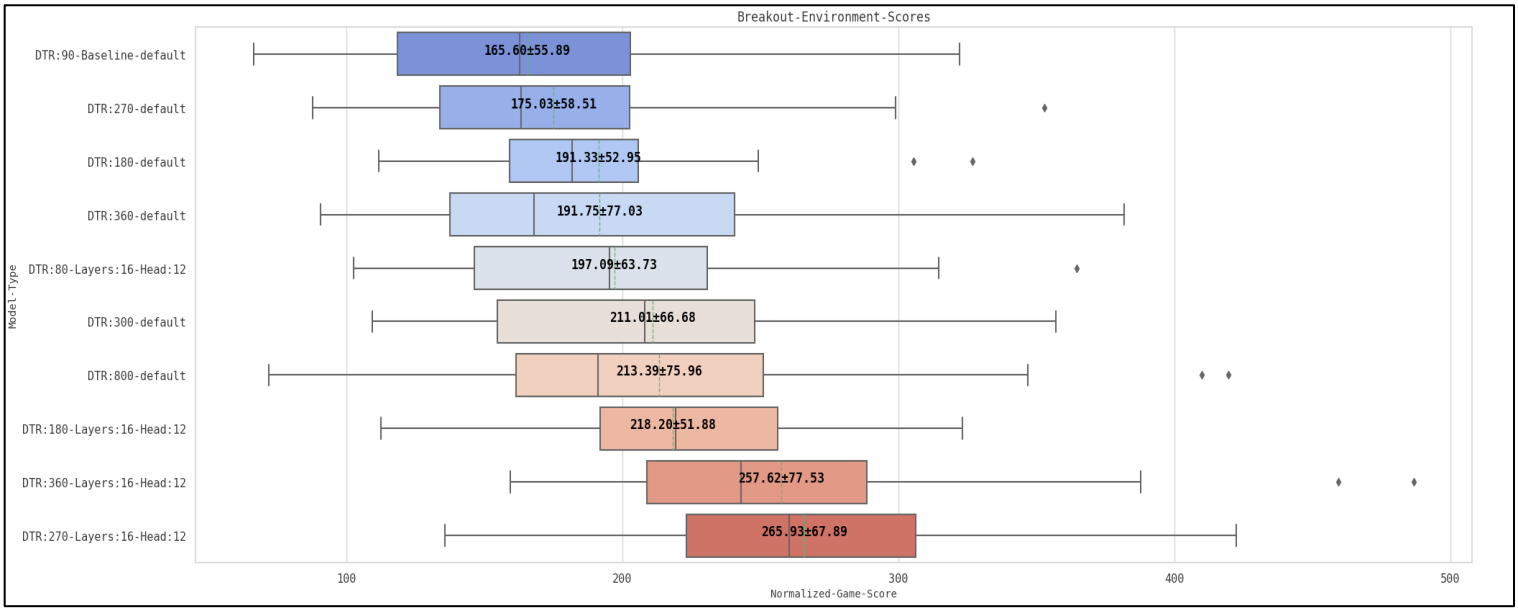


Figure 18 Results of DT models on the target environment Breakout. The normalized score against the model type. DTR is the raw score desired target return parameter – expected reward, Layers – number of transformer layers, Head - number of attention head respectively. Numbers in bold indicate mean and standard deviation respectively. The values which do not map to distribution are marked as simple dots. Baseline at the top, results sorted by mean. Best result at the bottom of y axis.

The Figure 18 presents the results obtained from training DT on 500000 transitions from offline dataset, with different model configurations, operating on the target environment Breakout. The Model-Type consists of DTR (desired target return) number of layers and attention heads. Keyword "default" refers to architecture authors [Clr+21] used, described in tables above in previous section. Boxes at the bottom have higher means then at the top of y axis. The results were obtained by training each model for 12 epochs across 3 seeds. The authors [Clr+21] used to train for 5 epochs, but to gather more data the number of epochs were increased. After each epoch the testing phase was performed and model tested on 10 online episode interactions, calculating total scores, and taking mean values from 10 episodes. To align with the authors [Clr+21], the score normalization step was performed. It is a simple calculation, where the overall score (raw score) from the environment was gathered and applied formula [Hln+21]:

$$\frac{raw\_score - random\_score}{professional\_gamer\_score - random\_score} * 100 \qquad (16)$$

The table [Hln+21] provides the proper professional, random scores for calculation. For Figure 18 values *{random_score: "2.0", professional_score "30"}* used. The median is drawn as a solid line on the box and mean values as dotted green line. The default keyword means that this configuration used the default DT architecture presented by the authors [Clr+21]. The DTR is the hyperparameter for the

DT, called desired target reward, or return to go. This score is passed to the transformer to perform autoregression and predict the future action. From the Figure 18, it can be noticed that the boxes shifted to the right have higher scores compared to the others. The bottom red box DTR-270 showed better results.

11.6.2. Transferring experiment

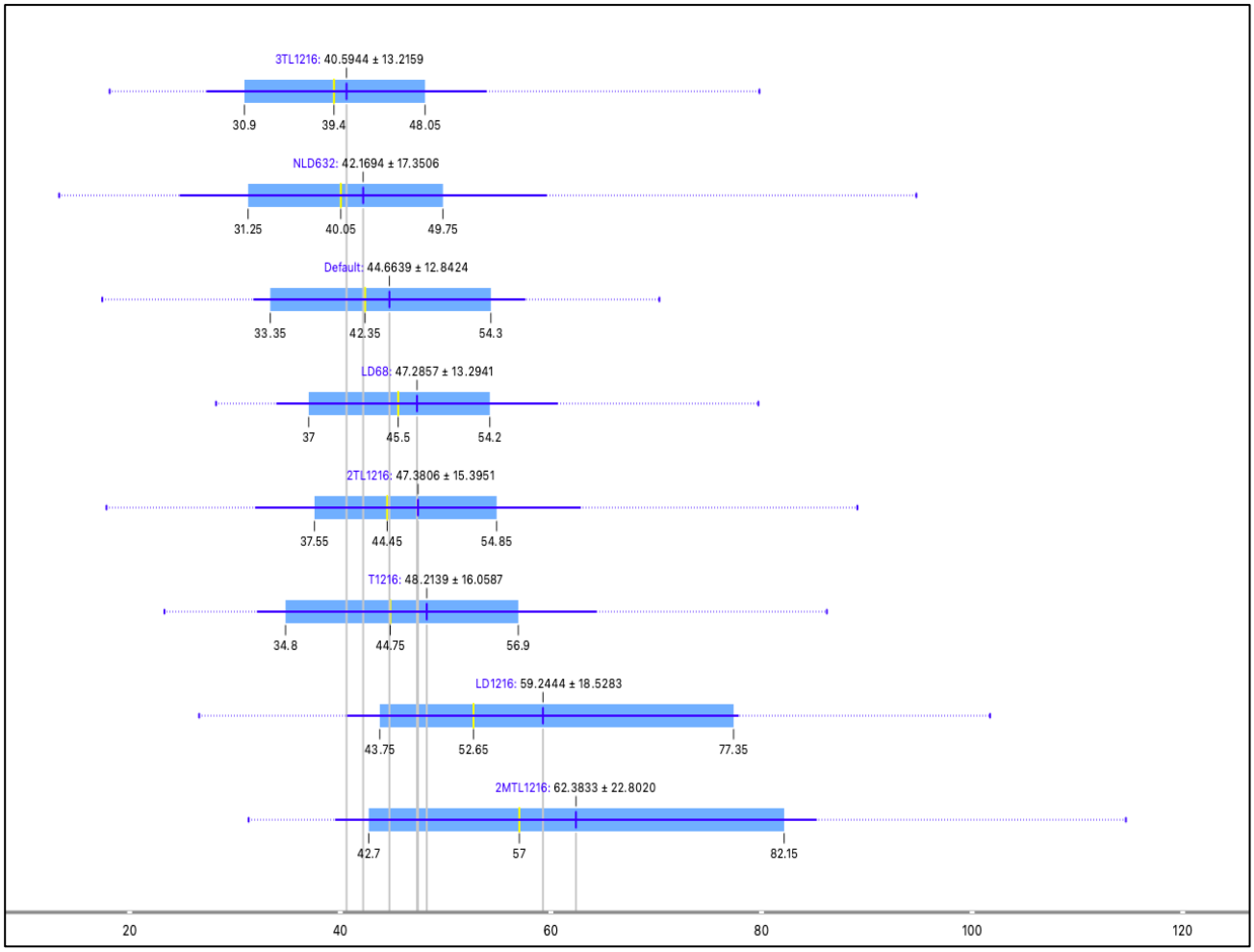


Figure 19 An example of raw scores obtained from number of modified environments. X-axis represents raw scores. These scores are not normalized. In the table below, the details of modified architectures are presented. Target domain Breakout. Results shifted to the right resents better results.

Model Index	Learning Decay	Attention Heads	Layers	Transfer pre training	Learning Rate	Epochs	Data ratio	Source Env	Desired Target Reward
NLD632	False	32	6	None	6*10-4	12	100%	-	90
Default	True	8	6	None	6*10-4	12	100%	-	90
LD68	True	8	6	None	3*10-4	12	100%	-	90
LD1216	True	16	12	None	3*10-4	12	100%	-	90
TL1216	True	16	12	1 epoch	3*10-4	12	20/80	Pong	90
2TL1216	True	16	12	2 epochs	3*10-4	12	20/80	Pong	90
3TL1216	True	16	12	3 epochs	3*10-4	12	20/80	Pong	90

2MTL1216	True	16	12	2 epochs	3*10-4	12	10/90	Pong	90

Model Index	Mean Values
NLD632	42.16 +- 17.35
Default	44.66 +- 12.84
LD68	47.29 +- 13.29
LD1216	59.24 +- 18.52
2MTL1216	**62.3833 +- 22.80**
TL1216	48.21 +- 16.05
2TL1216	47.38 +- 16.06
3TL1216	40.59 +- 13.22

Figure 19 displays architectures mean and standard deviation scores on the Breakout target domain with some of transferring experiment results on source Pong domain. The encoding for the architectures is NLD, LD, TL, MTL. The TL represents runs with transfer learning, MTL represents run with transfer and modified pretraining data split, LD architecture with modified learning decay, where NLD modified architecture with increased number of attention heads. The number on before the encoding represents 2TL, 3TL number of pretrained epochs on the pretraining step.

The primary addition to Figure 19 is runs that incorporate transferring experiment results. The results were obtained the same way as in the Figure 18, but with addition of transferring runs. The pretraining part was trained on different data split ration and with different pretraining epoch number. The primary model was then initialized the weights of the transferring model.

In summary of Figure 19, the model 2MTL1216 with transferring experiment showed better results.

11.6.3. Increasing number of environments

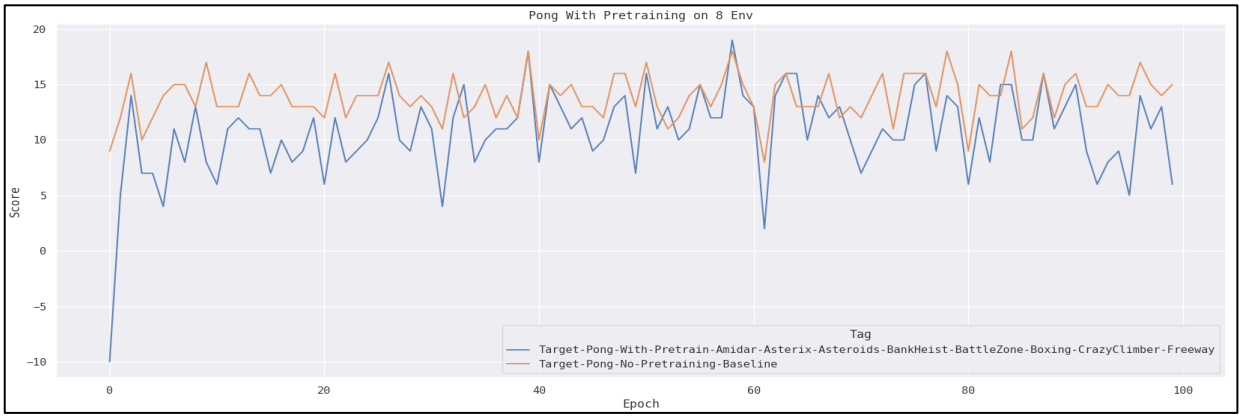


Figure 20 shows the results of training 100 epochs. Indicating of no need to train for more than 10 epochs. The target environment is Pong. X-axis is epoch number and y-axis the overall score obtained after the training epoch. The first epoch is labeled as 0.

During this experiment phase the model is modified to play on most of the environments from the subset. It is a different model compared with section above. The difference is in vocabulary size and maximum possible timestamps. The Figures 20 and 21 show a line plot on pretraining on target environment Pong with several environments as Source.

Figures 20 and 21 show negative and positive starts respectively. Likewise, Figure 20 shows that training on higher number of epochs is not necessary. For the reason of scaling across number of domains, the vocabulary size is increased as well as the maximum number of maximum timesteps is increased. From the Figure 20, the orange line represents the baseline training part and the blue line below the training with weight initialization after pretraining on 8 games (Amidar, Asterix, Asteroids, BankHeist, BattleZone, Boxing, CrazyClimber, Freeway). The higher the scores the better. From the Figure 20, the first pretraining step on the (epoch 0) receives negative result and as the epoch are

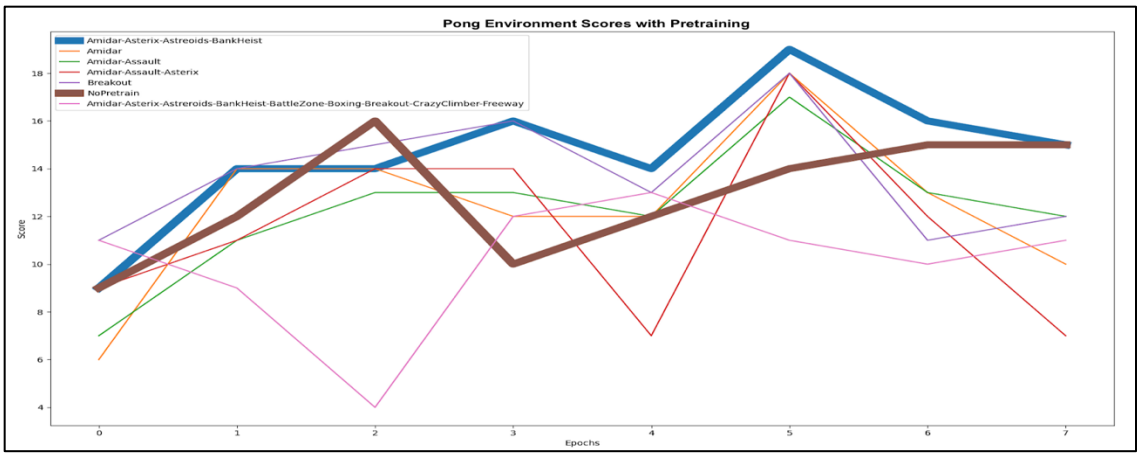


Figure 21 Results obtained on training with different pretraining environments. Brown line indicates baseline and blue line indicates positive pretraining result.

increasing the blue line does not surpass the baseline.

From the Figure 21, training on 8 epochs, the optimistic start was shown by the blue line, while baseline result is shown in brown. The combination of *Amidar-Asterix-Asteroids-Bankheist* is shown in blue. For the reminder, the score for each epoch is calculated across 3 seeds with mean over 10 testing episodes.

In order to see the overall score of pretraining on number of different environments an index encoding in figures was used to indicate the number of games that were included into pretraining phase. From the Figure 22, the y-axis graph contains the encoded values of overall scores of the model on the target environment breakout.

Nr	Name	W_19	W_18	W_17	W_16	W_15	W_14	W_13	W_10	…	W_1
1.	Amidar	+	+	+	+	+	+	+	+		+
2.	Asterix	+	+	+	+	+	+	+	+		-
3.	Asteroids	+	+	+	+	+	+	+	+		-

4.	Bankheist	+	+	+	+	+	+	+	+		-
5.	BattleZone	+	+	+	+	+	+	+	+		-
6.	BreamRider	+	+	+	+	+	+	+	+		-
7.	Bowling	+	+	+	+	+	+	+	+		-
8.	Boxing	+	+	+	+	+	+	+	+		-
9.	Breakout	+	+	+	+	+	+	+	+		-
10.	CrazyClimber	+	+	+	+	+	+	+	+		-
11.	Freeway	+	+	+	+	+	+	+	+		-
12.	Frostbite	+	+	+	+	+	+	+	+		-
13.	Gravitar	+	+	+	+	+	+	+	-		-
14.	Kangaroo	+	+	+	+	+	+	-	-		-
15.	MsPacman	+	+	+	+	+	-	-	-		-
16.	Pong	+	+	+	+	-	-	-	-		-
17.	PrivateEye	+	+	+	-	-	-	-	-		-
18.	SpaceInvaders	+	+	-	-	-	-	-	-		-
19.	Venture	+	-	-	-	-	-	-	-		-

The Figure 22 indicates the results of training on the accumulated environment weights from different source environments. The x axis shows raw scores of the target environment. The results shifted to the right of the x axis have higher means. The boxes are sorted out according to mean values. The box on the bottom of y axis have higher results compared to box on the top. The baseline is the model that had no initial weight injection, it is simple default test run on the environment without

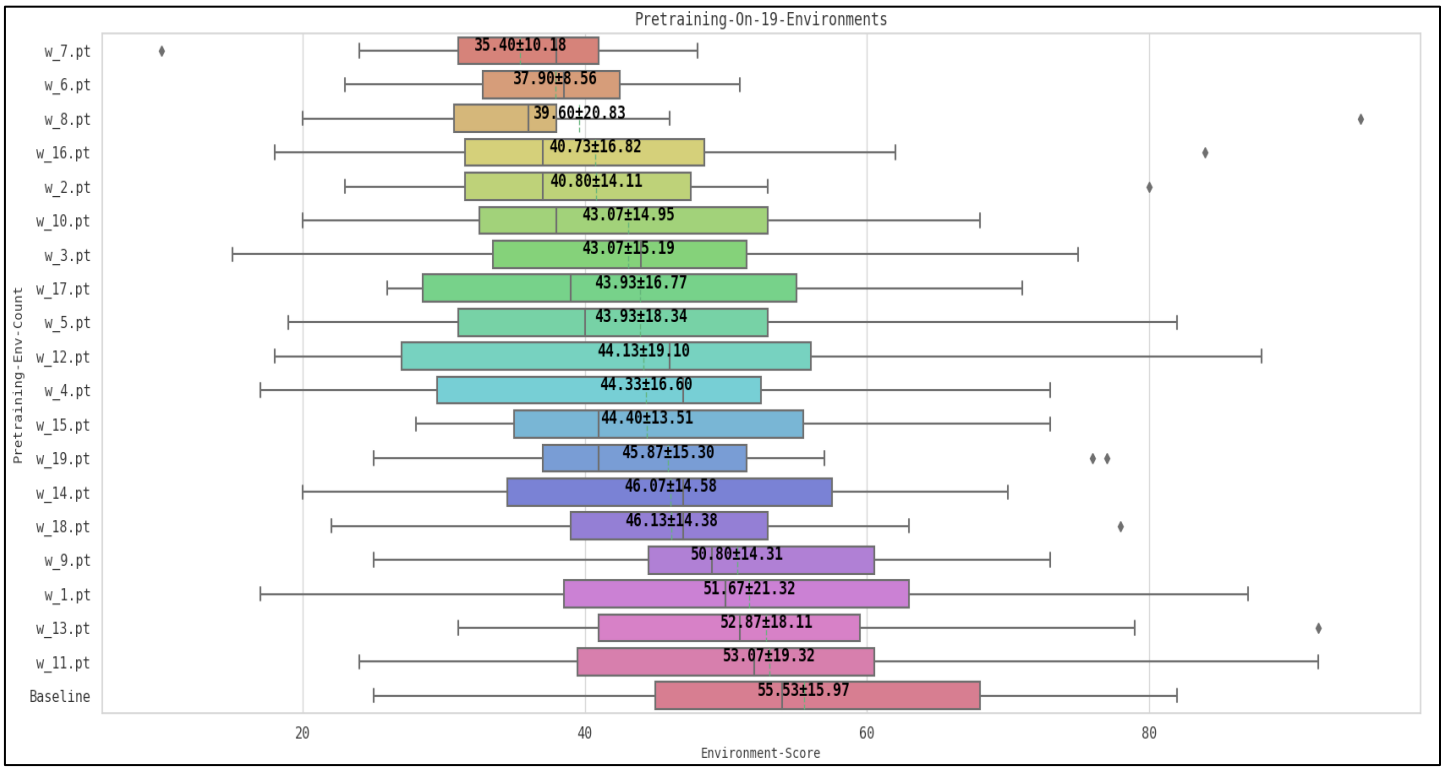


Figure 22 Target environment Breakout, pretraining on 19 different games. "w_1.pt" is the encoding for initialization weights. The number indicates how many environments were used for pretraining. The weights were gathered by accumulating training weights by the number of environments included. Numbers in bold indicate mean and variance. Green dotted line on the box is the mean, while the solid line is the median.

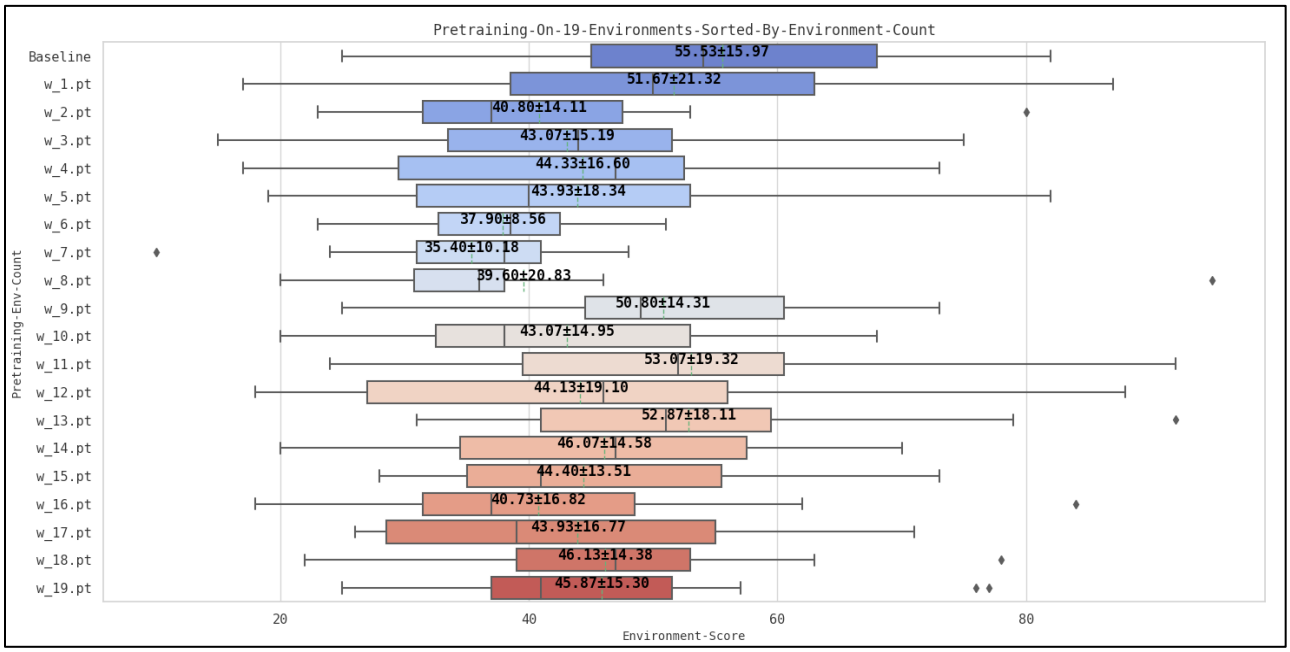


Figure 24 Sorted by accumulated environment count based results showing in gradient between blue from 1-10 and 10-19 red, pretraining split based on environment count. Score is not normalized. Baseline is on the top colored in blue.

pretraining. The maximum and minimum values are whiskers and dots on the graph indicate the out of the distribution scores. Results obtained from Figure 22 are using 6 layers, 8 attention heads and 128 embeddings. The same architecture as presented by authors [Clr+21].

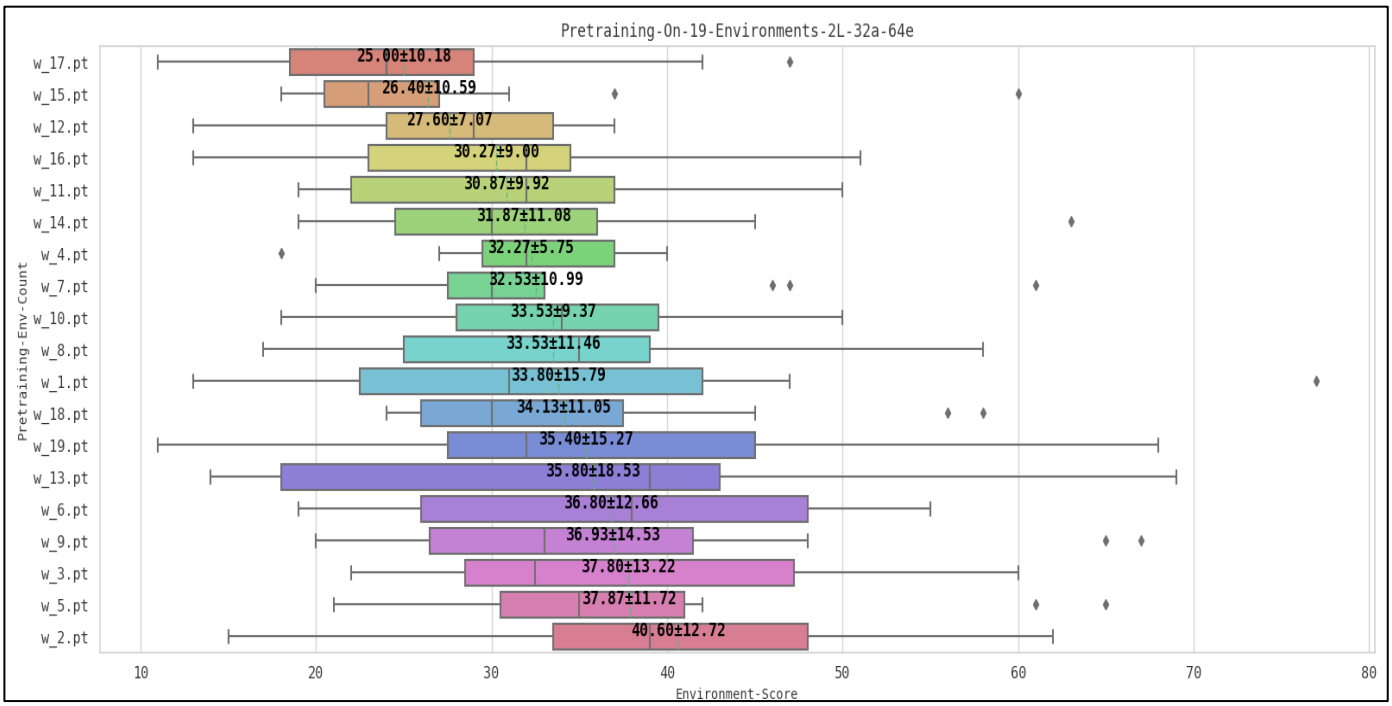


Figure 23 Results obtained with width-oriented architecture. 2 layers, 32 attention heads, 64 embeddings. Number of pretrained environment weights on the y-axis and the x-axis represents raw score. The results shifted to the right have higher scores.

Figure 23 represents the same multiple environment weight-oriented results but using a different width-oriented architecture. Comparing to height-based architecture, the mean results are smaller.

Figure 24 shows example of values sorted by environment count for the Figure 22, where the color indicates the two clusters (1-10 and 10-19).

Additionally, the important factor is to train on proper number of pretraining iterations. Lowering, the number of iterations in pretraining stage on source environments can lead to blending of results.

Furthermore, Figure 25 indicates results on pretraining with 1000 iterations compared with 10000 iterations on Figure 22. During the calculations, it was noticed that the results are not drastically different but lowering the number of pretraining iterations on source datasets, can lead to lowering the impact of results. This is an intuitive feature, but when transitioning to more light architecture, the

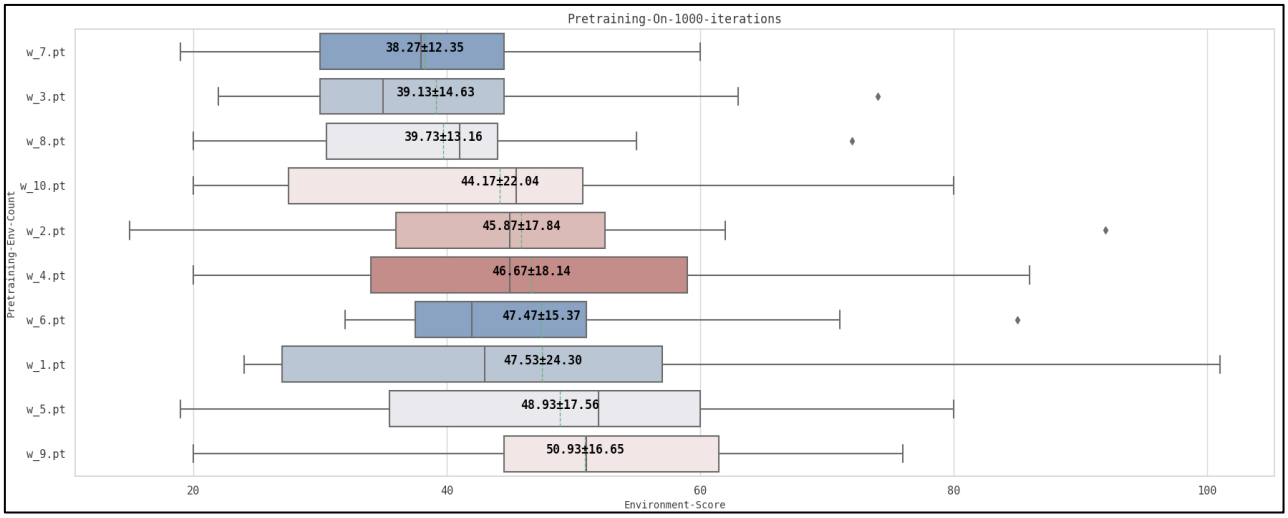


Figure 25 Results obtained using a depth-based architecture with pretraining step training only on 1000 number of environment steps.

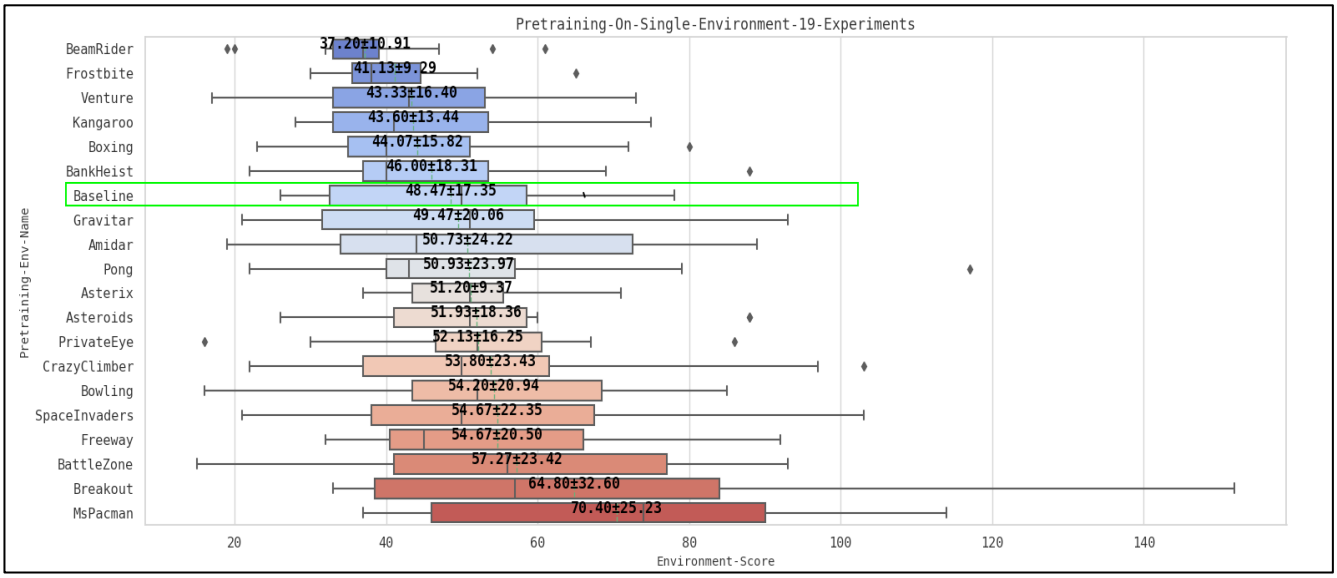


Figure 26 Results obtained from pretraining on single environment. Depth based architecture 6 layers-8 attention-128 embedding. DTR is set fixed set to 90. The y-axis indicates the environment name and the x-axis the scores. The baseline is selected in green box. Target Breakout, raw scores.

lower number of pretraining iterations makes greater impact on width-based architecture, compared to a more depth oriented.

The results (22-24) were obtained by using accumulated knowledge from the 19 environments. From the Figure 24 the maximums from pretraining on accumulated values are higher, but the mean score from transferring is lower the baseline. That is why, pretraining on single environment without accumulation can be considered. Consequently, the impact of each environment alone without any additional accumulation was calculated in Figure 26. The baseline without the pretraining is selected in a green box. The pretraining on single environment data gave a positive increase in results, leading the “MsPacman” environment most favorable compared to others in “Breakout” target domain.

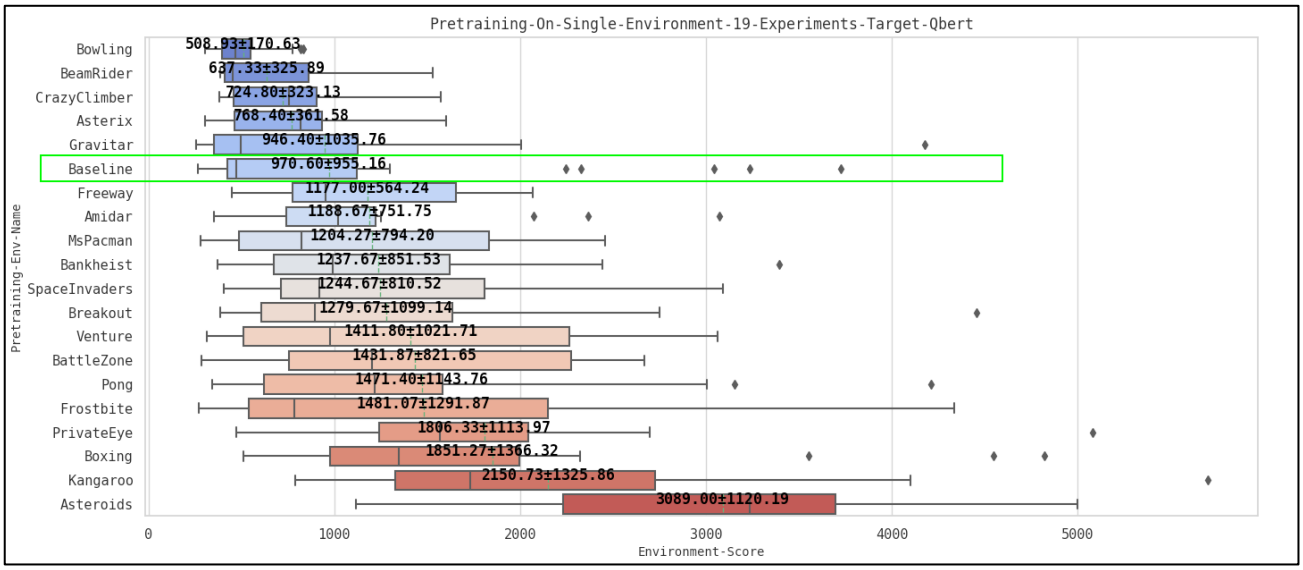


Figure 27 Results obtained from training on Qbert target environment with transferring approach. Raw scores. Results are not normalized.

To compare the transferring on another target domain, the “Qbert” was picked as a target. The reason was that [Clr+21] used it as a baseline. After training on Breakout environment, environment “Qbert” was picked that was not included in pretraining environment dataset. The results from the Figure 27 indicates increase in results obtaining best raw scores *3089+-1120.19* with initial weights from “SpaceInvaders”. With introduction of transferring, these results from Figure 27 are higher than provided by the authors [Clr+21] (Appendix B), comparing with initial DT approach.

11.7. Analysis

Due to the stochastic nature RL the observed variance is quite large in observed results between episodes and between average of different test runs. Additionally, due to time and computational constraints only several target domains were chosen aligning with [Clr+21] authors. Nevertheless, due to small number of target environments certain patterns can be observed. This section summaries the results to distinguish the dependencies and highlight vital observations.

11.7.1. Finetuning on single source

The Figure 18 indicates the training using different value of DTR hyperparameter (expected reward) as well as changing the architecture. The results of Figure 18 indicate that increasing the DTR can lead to better results. It should be noticed that it cannot be set to a maximum arbitrary number. The maximum possible score in Breakout environment 896. Comparing runs with 800 and 300 DTR values and default architecture, there is no major impact in scores. However, finetuning on 270 and increasing the depth of the architecture increases the results reaching [Clr+21] authors best results and improving the standard deviation.

Figure 19 indicates the results obtained by implication of transferring experiment. In practice, the [Clr+21] default model was used and obtained mean values. Afterwards, small fine tune of the model (or in other words mutation) was done in order to gather practical results for comparison purposes for transfer models. The LD1216 model showed better results in this experiment domain. Consequently, the code was modified to work with mixed data of several environments. As a result, the transferring approach was made using the same architecture of LD1216 as base, but small differences and configuration specific details are defined in table above in model description section. As mentioned before the primary part of the evaluation criteria was the score obtained in the Atari *Breakout* environment. The experiments in Figure 19, are containing results with static DTR parameter set to 90. The baseline result is 44.66 +- 12.84 with DTR set to 90**.** Taking this into account, the transferring experiment presented an increase in mean performance with Pong environment dataset. The interesting part is that Pong and Breakout are almost the same, but Pong has horizontal orientation vs Breakout vertical one. Despite the differences on observed data, with small finetuning the model with initial weight trained on Pong improved the results. The data split of 10% gave better results, as well as pretraining with 2 epochs improved the overall picture.

Additionally, the important parameter of Decision Transformer is the context length which was set as constant as well with a value of 30, according to [Clr+21] conclusions. These parameters and impact could be a future scope of analysis.

11.7.2. Scaled transferring

To see how the scaled model operates on number of environments, long distance experiment with 100 epochs was run. Figure 20 showed pessimistic start due to pretraining on larger number of environments as well as showing a problem that increase of pretraining environment on target domain could cause a slight degradation even in a longer run of 100 epochs. Also, the figure shows no need to train on big number of epochs, because values span across the same range in longer run.

For reflecting the overall pretraining combination impact of environments, the accumulative experiment was run. The scaling of DT on 19 source environments, was presented in Figure 22 and 23. In summary, the increase in source environments did not drastically improve the mean values of results comparing with baseline, but there is a positive change in achieving more maximum results across the distribution in Figure 22, with weights 11, 13 and 1. The interesting observation is that it was expected to see a degradation of mean results, when increasing number of additional environments (so that weights with 19 should be at the top). The empirical results indicate that the number of environments is not impacting the transferring results in negative way literally. Even with all 19 environments included the run with 19 is in the middle of score chart. Taking this into account, the game mechanics and the similarities between the input could be the possible highest impact on scores compared to the number of environments. The run with 8 environments did not contain the Breakout pretraining, but the 9th presented the Breakout data for target domain and improved the results.

Also, the motivation for fine tuning was the [Clr+21] authors' statement about future work. In the domain of RL, where the input data is dependent on actions and states, the increase of attention heads and a small number of layers increasing the depth of the architecture could be an intuitive approach, but also the alternative approach of widening the architecture. By introducing the higher number of attention heads and lowering the number of embeddings and layers could provide a better architecture, but empirical results indicate the depth-based approach is more favorable (comparing results 23 and 22).

Likewise, a split is presented in Figure 24. The figure using color splits data into two sectors 1-10 and 10-19, showing several details. It can be noticed that 10-19 environment pretraining gives higher maximum results and more likely to appear around higher pretraining environment numbers. The 11 pretraining achieved highest reward from the environment after the baseline.

Moving from the accumulation pretraining strategy to single source environment resulted in improvement of baseline mean scores. The increase in 20 score points for baseline was accomplished by pretraining on single "MsPacman" environment. From the Figure 24, The "Breakout" pretraining on same environment data intuitively gave higher results, proving that the generalization was happening. The peculiar result is that "MsPacman" presented increase in mean score. One of explanations could be that "Breakout" environment and "MsPacman" are horizontally oriented as well as the movement from left to right is present in both games. Additionally, the breakout paddle board and "MsPacman" ghost board have square shape. Furthermore, the pretraining on more complex environment, but with important similarities could lead to increase in results in general. Likewise, the

"BattleZone" environment was the third from the top to improve the results. The interesting fact is that game is 3D oriented and moving to 2D space presents a simplification, which leads an increase in mean scores. Additionally, same behavior is noticed with target environment "Qbert". From Figure 27 the "Qbert" environment best performed on "Asteroids" pretraining environment, obtaining higher scores. The possible explanation could be related to previous target environment. "Qbert" is about jumping around pyramid tiles across horizonal and vertical span. "Asteroids" have a higher range span of controls moving horizontally and vertically, introducing higher complexity, which could be useful in transferring domain. Finally in Figure 21, the Pong environment is having a positive increase during transferring.

12. Completed work

- The generative model was successfully trained and evaluated to play in RL domain using offline data.
- The generative model was successfully scaled across number of 19 Atari environments.
- Transformer model was able to perform transferring experiment using single and accumulated knowledge from 19 environments.
- Transformer width and depth-based architectures were tested in operating for offline domain.

13. Results

- Results of the paper [Clr+21] were improved in scope of this work.
- The pretraining on single environment improved baseline results.
- Transferring on higher number of environments achieved lower baseline results.
- The accumulative approach for pretraining environment data led to lower results (Figure 24).
- Target environment "Breakout" had best transferring on "MsPacman" environment from a set of 19 environments (Figure 26).
- Target environment "Qbert" had best transferring on "Asteroids" environment from a set of 19 environments (Figure 27).
- In case of "Qbert" with transferring improved baseline as well as results in [Clr+21] obtained.
- Applying transferring with finetuning on similar game-based datasets mean results were improved from 44.66 +- 12.84 to 62.38 +- 22.80 (Figure 19).

14. Observations

- The interpretation of RL task as sequence-to-sequence domain practically allows to use the scaling nature of generative approach. Unfortunately, due to lack of online interactions the generative approach can only compete with offline RL domain, achieving lower score values compared to DQN agents. After observing how the agent interacts with the environment, it can be stated that more exploration of the environment is needed.
- The ability to pretrain on multiple and different source environments grants the model to operate on all 19 environments at the same time. Taking this into account, the approach of using Transformer for solving RL domain can be used to learning multiple domains at the same time using same network. This ability is lacking in most of DQN agents.
- The knowledge accumulation experiment could be researched further by introducing deeper architecture. There is a probability that the model architecture was too shallow to handle 19 environments. Additionally, special inner task mapping could be implemented to improve the result, but this tactic is forcing model to attach to environment specifics, resulting in loss of generality.
- The number of pretrained environments does not influence intuitively the mean results, (in other words - the higher count leads to better, or the worse performance). The higher number of environments between 10-19 had better results than 1-10 (Figure 24). The possible explanation could be presence of “Breakout” data in the second half. In other words, the similarities between the target and source environment have higher impact on transferring compared with the number of sources.
- From experiments, the data split of 10% of observations from foreign environment and 90% of observations from target could lead to better transferring results (Figure 19).

15. Research Summary

The view of the RL problem as a sequence-to-sequence paradigm is quite novel. The primary goal of the thesis was reached, by training a generative model to operate on RL domain and to perform transferring experiment on multiple source environments. The tasks formulated beforehand were accomplished:

- The thesis presents a theoretical overview of the problem and description of several state-of-the-art solution approaches.
- The training and parameter tuning of generative model was performed.

- The scaling of the generative model on several source environments was performed.
- The quantification of pretraining and the benefits were displayed in obtained scores.

16. Conclusions

Summarizing the results obtained, several conclusions can be formulated:

- Pretraining on single - more complex environment (Figure 26, Figure 27), but with one important aspect of similar mechanic, orientation, or visual input surpasses mean baseline results and makes transferring successful.
- From the experiments, the accumulation transferring on 19 environments showed lower baseline scores. Despite the fact, the higher number of environments positively influences the higher maximums (Figure 24), resulting in improvement.
- From experimental results, the depth-based architecture is more favorable compared to width based (Figure 23).
- Pretraining on higher number of environments, can lead to lower performance in first epochs (Figure 20).
- Large number of environments in epochs 5+ is not making a significant impact on results.
- Larger offline pretraining dataset gives positive transferring impact more for depth-based comparing with width-based.

17. Future Work

This part of work initialized the main experiment of transferring, allowing broader and more detailed research in future stage.

- The generative transformer could implement an online signal from the environment to increase the exploration of the data.
- The data from the environments, can have additional transformations introducing unification across domains for improvement of transferring and operation results.

Some interesting experiments can be made:

- Introducing GAN networks to generate the random transferring states of Atari 2600 screenshots.
- Learning from demonstrations, or a supervisor. The model can be improved to include the offline or online input from the human player and use them as source. Moving the problem into self-supervised domain.

18. References

[Aab+22] J. Abramson, A. Ahuja, A. Brussee and more. Creating Multimodal Interactive Agents with Imitation and Self-Supervised Learning. arXiv preprint arXiv:1511.06342. 2022.

[Ant+94] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based behavior. Acquisition for a Shooting Robot by Using a Reinforcement Learning. In Proceedings of IAPR/IEEE Workshop on Visual Behaviors-1994, pages 112–118, 1994.

[Bbc+19] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak and more. Dota 2 with Large Scale Deep Reinforcement Learning, OpenAI Five, 2019.

[Bhs+15] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor and A. Nowe. Reinforcement Learning from Demonstration Through Shaping. Proceedings of the 24th International Conference on Artificial Intelligence, pp 3352-3358, 2015.

[Cig+15] J. Cichon, W. B. Gan. Branch-Specific Dendritic ca2+ Spikes Cause Persistent Synaptic Plasticity. Nature 520, 2015, pp. 180–185.

[Clr+21] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A Glover and more. Decision Transformer, Reinforcement Learning via Sequence Modeling. UC Berkeley, Facebook AI Research, Google Brain, Advances in neural information processing systems.

[Dap+18] R. Dubey, P. Agarawal, D. Pathak, T. L Griffiths and more. Investigating human priors for playing video games. 2018. Proceedings of the 35th International Conference on MachineLearning, Stockholm, Sweden, 2018, PMLR 80.

[Dbk+20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D.Weissenborn and more. An image is worth 16x16 words: Transformers for image recognition at scale. Preprint arXiv:2010.11929, 2020.

[Dcl+08] C. Diuk, A. Cohen and M. L. Littman. An Object-Oriented Representation for Efficient Reinforcement Learning. Proceedings of the 25th international conference on Machine learning, 2008.

[Dec97] D. DeCoste. The Future of Chess-Playing Technologies and the Significance of Kasparov Versus Deep Blue, 1997, pp. 9-13.

[Ehl+21] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley and J. Clune. First Return Then Explore. Nature 590, 2021, pp. 580-586.

[Fkn+20] J. Fu, A. Kumar, O. Nachum, G. Tucker and more. D4RL: Datasets for Deep Data-Driven Reinforcement Learning, 2020, arXiv preprint arXiv:2004.07219.

[Fda05] S. Fusi, P. J. Drew, L. Abbort. Cascade Models of Synaptically Stored Memories. Neuron 45, 2005, pp. 599–611.

[Ggr+19] D. Ghosh, A. Gupta, A. Reddy, J. Fu and more. Learning to Reach Goals via Iterated Supervised Learning. arXiv preprintarXiv:1912.06088, 2019.

[Gka+17] J. Gideon, S. Khorram, Z. Aldeneh, D. Dimitriadis and more. Progressive Neural Neworks for Transfer Learning in Emotion Recogntion. arXiv preprint arXiv:1706.03256. 2017.

[Hln+21] D. Hafner, T. Lillicrap, M. Norouzi and J. Ba. Mastering Atari with Discrete World Models. Published as a conference paper at ICLR, 2021.

[Hza+18] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In International conference on machine learning, 2018, pp. 1861-1870.

[Jjw17] J. Han, A. Jentzen, and E. Weinan. Overcoming the Curse of Dimensionality: Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 2017, pp. 115.

[Jle21] M. Janner, Q. Li, S. Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem. NeurIPS Spotlight, 2021.

[Hos97] S. Hochreiter and J. Schmidhuber. Long Short-term Memory. Neural Computation 9: 1735-80, 1997.

[Hgs15] H. Hasselt, A. Guez and D. Silver. Deep Reinforcement Learning with Double Q-learning. Google DeepMind, 2015.

[Hla19] C. Hung, T. Lillicrap, J. Abramson, Y. Wu and more. Optimizing Agent Behavior Over Long Time Scales By Transporting Value. Nature communications, 2019, pp. 1–12.

[Klm96] L. Kaelbling, M. Littman and A. Moore. Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4, 1996, pp. 237-285.

[Kft+19] A. Kumar, J. Fu, G. Tucker and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019, pp. 11784-11794.

[Kft+21] I. Kostrikov, R. Fergus, J. Tompson and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, 2021, pp. 5774–578.

[Kpr+17] J. Kirkpatrick, R. Pascannu, N. Rabinowitz, J. Veness and more. Overcoming Catastrophic Forgetting in Neural Networks. Proceedings of the national academy of sciences 114.13, 2017, pp. 3521-3526.

[Kr18] T. Kudo and J. Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In Annual Meeting of the Association for Computational Linguistics, pages 66-71, 2018.

[Lap18] M. Lapan. Deep Reinforcement Learning Hands-On. Packt Publishing, Birminghan, 2018, 546 pages.

[Mks+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wiersta, M. Reidmiller. Playing Atari with Deep Reinforcement Learning. DeepMind Technologies. 2013.

[Nkf+17] A. Nagabandi, G. Kahn, R. S. Fearing and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. University of California, 2017.

[Ngd+20] A. Nair, M. Dalal, A. Gupta, S. Levine. Accelerating Online Reinforcement Learning with offline datasets. arXiv preprint arXiv:2006.09359, 2020.

[Pag19] C. Packer, K. Gao. Assessing Generalization in Deep Reinforcement Learning. arXiv preprint arXiv:2111.09794, 2019.

[Pbs15] E .Parisotto, JL. Ba, and R. Salakhutdinov. "Actor-mimic: Deep Multitask and Transfer Reinforcement Learning. arXiv preprint arXiv:1511.06342, 2015.

[Roz82] Y.A. Rozanov. Markov Random Fields. Springer Science & Business Media, New York, 1982, 196 pages.

[Rnd16] S. Russell, P. Norvig and E. Davis. Artificial Intelligence: A Modern Approach. 3rd ed, Prentice Hall, Upper Saddle River, NJ, 2016, pp. 649.

.[Rrd+16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins and more. Progressive Neural Networks. Google Deepmind, 2016, arXiv preprint arXiv:1606.04671.

[Rvr+17] A. A. Rusu, M. Vecerik, M., Rothorl and more. Sim-to-real robot learning from pixels with progressive nets. In Conference on Robot Learning PMLR, 2017, pp. 262-270.

[Rzp+22] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo and more. A Generalist Agent. arXiv preprint arXiv:2205.06175, 2022.

[Sb16] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. The MIT Press. Cambridge, Massachusetts, 2016.

[Sch20] J. Schmidhuber. Reinforcement Learning Upside Down: Don’t Predict Rewards – Just Map Them to Actions. The Swiss AI Lab, IDSIA, USI & SUPSI NNAISENSE, Lugano, Switzerland, 2020.

[Spl+15] S. Samothrakis, D. Perez-Liebana, S. M. Lucas and M. Fasli. Neuroevolution for general video game playing. Computational Intelligence and Games (CIG), IEEE, 2015, pp. 200-207.

[Ss96] S. P. Singh and R. S. Sutton. Reinforcement Learning with Replacing Eligibility Traces. Machine Learning, 1996, pp. 123-158.

[Sdm98] W. Schultz, P. Dayan, P. R. Montague. A Neural Substrate of Prediction and Reward. Journal of neurophysiology, 1-27, 1998.

[Shm+16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre and more. Mastering the Game of Go with Deep Neural Networks and Tree Search, 2016, Nature 529, pp. 484-489.

[Smp96] Y. Shoda, W. Mischel, and P. K. Peake. Predicting adolescent cognitive and self-regulatory competencies from preschool delay of gratification: Identifying diagnostic conditions. Developmental Psychology, 1996, 26*(6),* 978–986.

[Ssb85] O. G. Selfridge, R. Sutton and A. G. Barto. Training and Tracking in Robotics. Proceedings of the 9th International Joint Conference on Artificial Intelligence, pages 670-673, 1985.

[Ts09] M. E. Taylor and P. Stone. Transfer Learning for Reinforcement Learning Domains: A Survey. Journal of Machine Learning Research 10, 2009, pp. 1633-1685.

[Tsc11] M. E. Taylor, H. B. Suay and S. Chernova. Using Human Demonstrations to Improve Reinforcement Learning. The AAAI Spring Symposium, Help Me Help You - Bridging the Gaps in Human-Agent Collaboration, Palo Alto, 2011.

[Vbc+19] O. Vinyals, I. Babuchkin, W. Czrnecki, M. Mathieu, A. Dudzik and more. Grandmaster Level in StarCraft 2 Using Multi-Agent Reinforcement Learning, 2019, Nature 575, 350-354.

[Vsp+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones and more. Attention is All You Need, Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp 6000-6010.

[Wc18] W. William, K. Chen. Learning to Play General Videogames via an Object Embedding Network. 2018 IEEE Conference on Computational Intelligence and Games (CIG), 2018, pp. 1-8.

[Wd92] C. Watkins and P. Dayan. Q-Learning. Machine learning, 1992, pp. 279-292.

[Wlj+21] N. Wies, Y. Levine, D. Jannai and A. Shashua. Which Transformer Architecture Fits My data? 38th International Conference on Machine Learning (ICML), PLMR 139, 2021.

[Wmb+19] N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess and more. COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration. arXiv preprint arXiv:1905.09275*,* 2019*.*

[Wsl+17] S. Wang, S. Sun, Z. Li, R. Zhang and more. Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model. PLOS Computational Biology, 2017, pp. 1-34.

[Ypg09] G. Yang, F. Pan, W. B. Gan. Stably maintained dendritic spines are associated with lifelong memories. Nature 462, 2009, pp. 920–924.

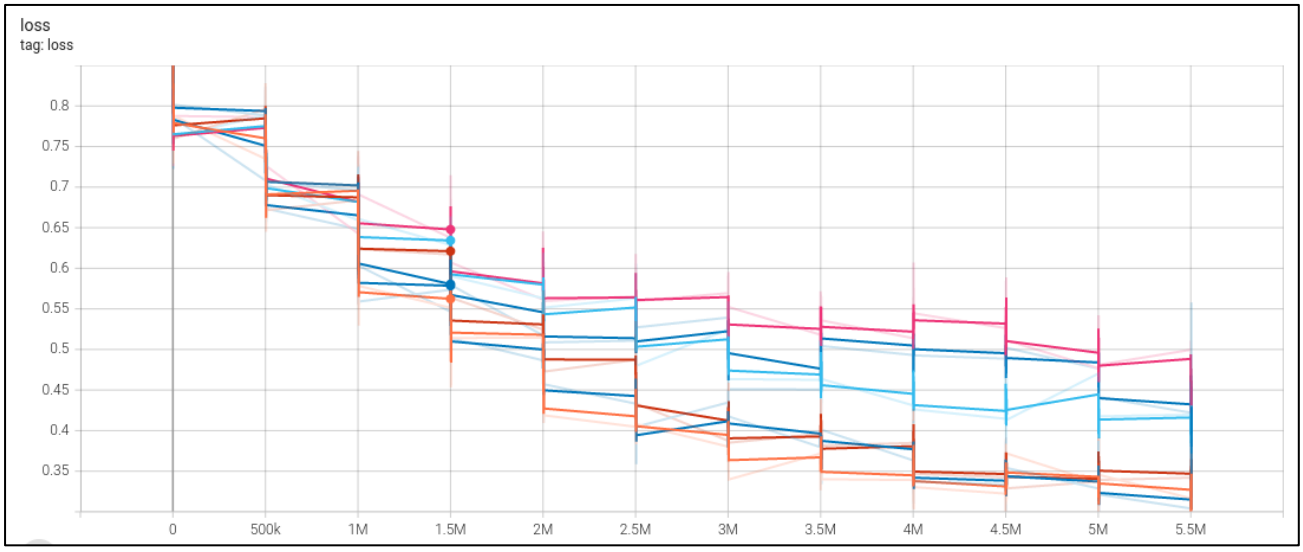
[Zl16] B. Zoph and Q. V. Le. Neural Architecture Search with Reinforcement Learning. Google Brain, ICLR conference, 2016.

.

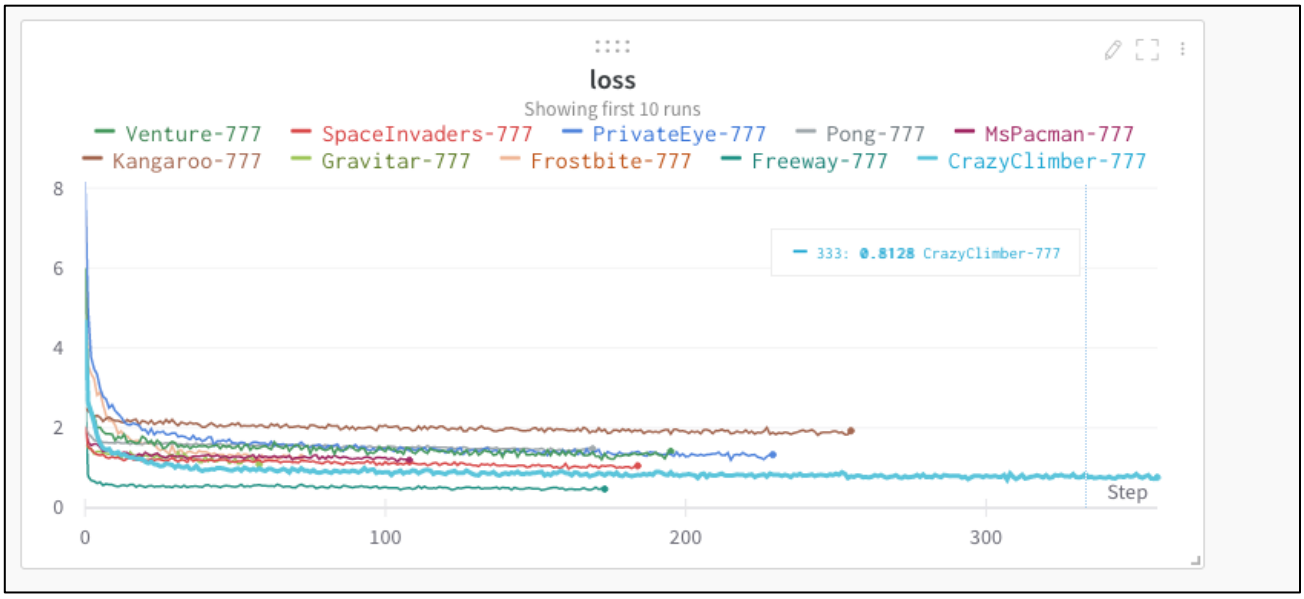
[Zlz09] Z. Zhu, K. Lin and J. Zhou. Transfer Learning in Deep Reinforcement Learning: A Survey, 2009.

[Zzg22] Q. Zheng, A. Zhang, A. Grover. Online Decision Transformer. Facebook AI Research. University of California, Berkley 2022.

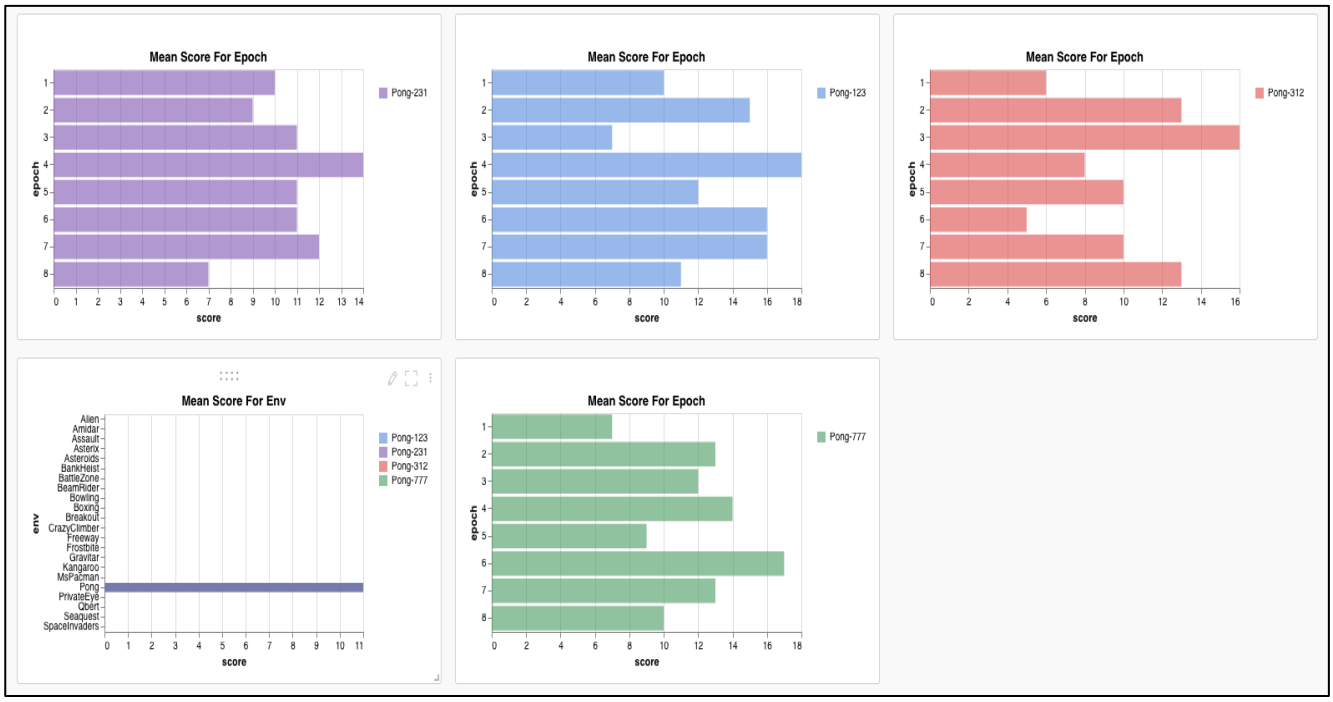
19. Appendix



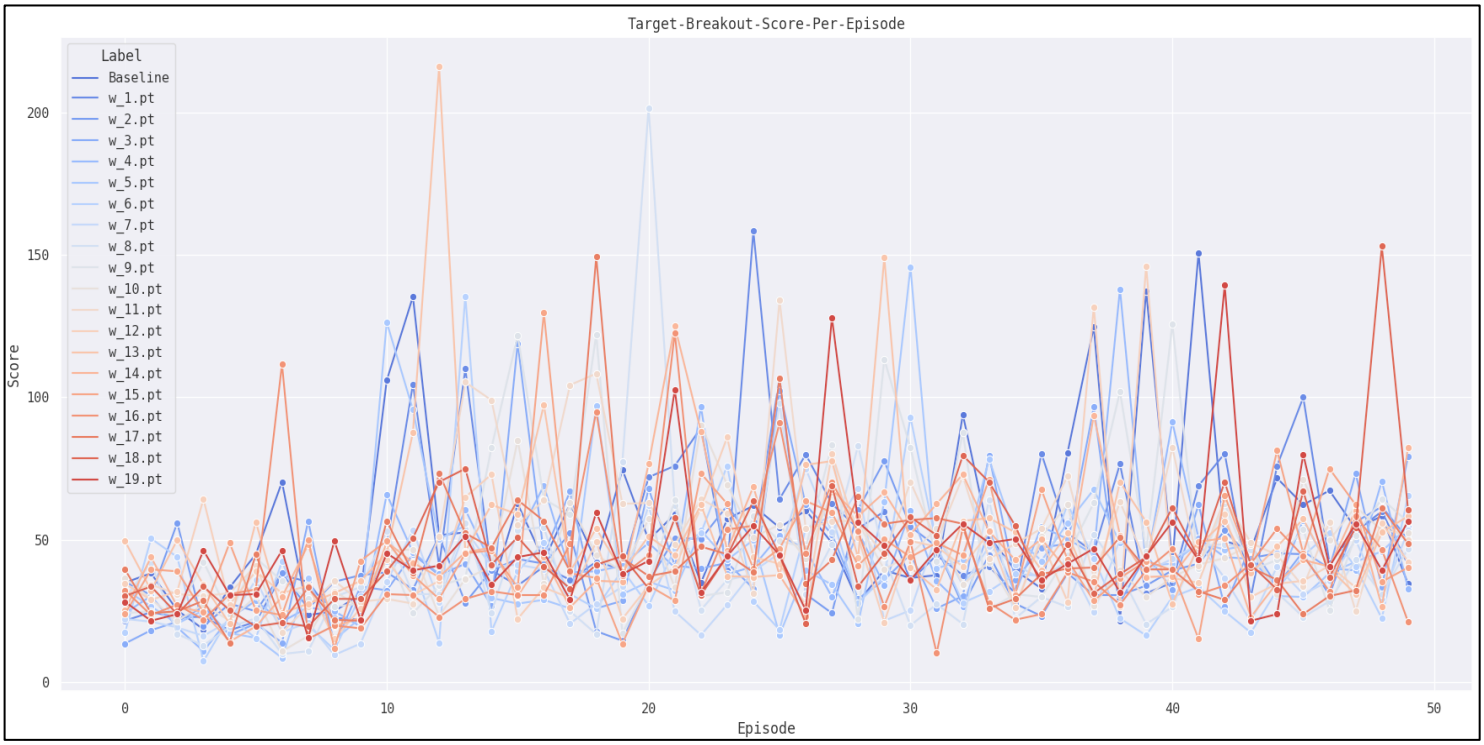
Example of loss values on target domain Breakout.



Example loss function on several pretrained environments on 777 seed.



Example of training across 4 seeds on the Pong environment



Example of score per episode in knowledge accumulation experiment. Color indicates division into two groups (1-10) (10-19) environment count. Red lines have more positive start and higher maximum results.

20. Acronyms

AI	Artificial Intelligence
A2C	Actor-Critic Methods
DL	Deep Learning
DT	Decision Transformer
DTR	Desired Target Return
DQN	Deep Q Network
DDQN	Double Deep Q Network
HER	Hindsight Experience Replay
IQN	Implicit Q Network
MDP	Markov Decision Process
NLP	Natural Language Processing
ODT	Online Decision Transformer
OMDP	Object Markov Decision Process
RL	Reinforcement Learning
RTG	Returns to go
RNN	Recurrent Neural Networks
SL	Supervised Learning
TL	Transfer Learning