Learning to play Tetris with Hierarchical Reinforcement Learning

Sayak Dasgupta

MSc in Data Science
The University of Bath
2023

Learning to play Tetris with Hierarchical Reinforcement Learning

Submitted by: Sayak Dasgupta

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Tetris is a classic video game usually represented by a 20 x 10 grid with seven different falling pieces or tetrominoes. The pieces stack upon one another and a player earns points when all the cells in a single row are filled. Existing artificial agents have attempted to play the game with hand-crafted features that are assumed to improve the score. However, those results have been very far from the best human player performances. A common strategy followed by expert players is to manipulate the pieces ahead to stack up in a certain manner that can guarantee high scores at later timesteps. Hence it presents a great test bed for the investigation of action hierarchies in a sequential decision-making environment. An action hierarchy can be described as a set of actions stacked upon another set where the top-level action can call the actions below it. This decomposes a learning objective into different sub-goals each attended by different levels of actions in the hierarchy. The project investigated such hierarchies in Tetris using the Options framework. Options are closed-loop hierarchical actions that are extended beyond a single timestep of the game and end stochastically with some distribution. One such method is Option-Critic which learns such options end-to-end without explicitly defining the sub-goals before. The project empirically established that such hierarchies are present when experimented on a smaller version of the game using this method. The options learn to plan for interpretable sub-goals on their own without any extra reward directed towards these actions. An example of such a sub-goal from the project is an option planning to stack a well-shaped block on the board with an empty column between that is cleared by another option later on by dropping an I piece and scoring two points.

Contents

List of Figures

Acknowledgements

Acknowledgements given to my supervisor Dan Beechey for his continued support throughout the project. Special thanks also given to a Nuno Faria for the Python Tetris Framework and Bacon, Harb and Precup (2016) for their Option-Critic code made available on GitHub.

Chapter 1

Introduction

1.1 Background

1.1.1 Game of Tetris

Tetris is a classic video game that was first created in the Soviet Union in 1984 by Alexey Pajitnov. The traditional Tetris board consist of a grid of cells and pieces of different shapes that fall from the top of the grid to fill up the bottom or stack on existing cells. These gaming pieces called tetrominoes usually occur in seven different shapes selected randomly with equal probability. Tetrominoes are controlled during their fall by rotating them or moving along the columns on the left or right. When all the cells of a row are occupied that row gets cleared and the rows above drop by one position. The player gets points for clearing single or multiple lines of the grid and the game ends when there's no space for the next tetromino at the top. The

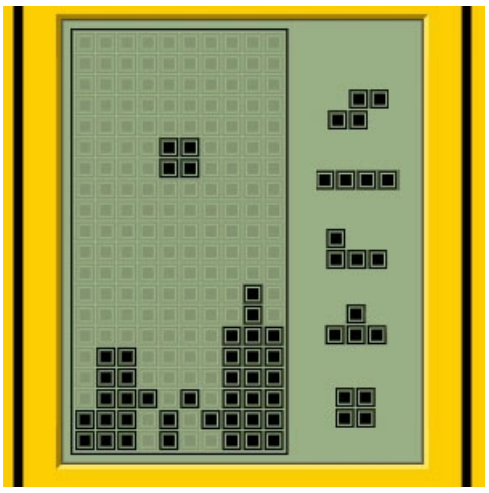


Figure 1.1: A grid showing the falling pieces. The tetrominoes on the right are the S, I, L, T and O shapes.

goal of a player is to earn maximum points by clearing as many lines as possible. It has been found that the game always ends with a probability of 1 for a certain sequence of tetrominoes (Burgiel, 1997). Even knowing all the sequences of tetrominoes for a particular trajectory, figuring out the combination that returns the maximum score is an NP-hard problem (Demaine, Hohenberger and Liben-Nowell, 2002).

1.1.2 Existing Controllers for Tetris

Tetris qualifies as a good benchmarking testbed for RL algorithms as it involves a large number of states and can be considered a sequential decision-learning problem. The success of RL agents in different well-known video games like Atari in the past decade has shown a lot of promise. However, given the large state space, these agents have struggled to solve Tetris. Additionally, an agent does not have any idea of the order of the next pieces in Tetris which makes the environment partially observable and developing effective strategies difficult. Furthermore, the game generates sparse rewards which are often difficult to track back through several timesteps to actions that led them. Currently, the most successful Tetris controllers are built using some kind of function approximators and hand-crafted features. Examples of such features can be holes which are empty spaces or gaps left behind on the board by pieces that did not fit together, landing height as in the height at which the last tetromino landed etc. Thiery and Scherrer (2009) describes controllers as either one-piece controllers or two-piece controllers and all of them rely on an evaluation function. One-piece controllers are those that are built using the information from the current board and piece whereas two-piece controllers also include knowledge about the next piece. The goal of any controller is to select actions which return the highest evaluations.

The early attempts to build controllers involving those pre-defined state features such as the number of holes(Tsitsiklis and Van Roy, 1995), the height of each column, the difference in column heights(Bertsekas and Tsitsiklis, 1996) etc. recorded the best score of 6800 lines cleared using policy-gradient methods(Kakade, 2001). All these methods required a lot of feature engineering and hand-crafted weights for those features. The best hand-crafted controller was built by Fahey (2003) and used a linear evaluation function which cleared 660,000 lines. Szita and Lörincz (2006) used cross-entropy optimization along with noise to build an agent that outperformed previous RL algorithms by almost two orders of magnitude. The success of this paved the way for incorporating genetic algorithms into solving Tetris which proved to achieve scores almost as high as that of the previous hand-crafted controllers. Many recent iterations of controllers include an evolutionary algorithm within the policy of the RL agent and were able to generate state-of-the-art results. One such method used classifiers within the agent to classify which actions would be better instead of selecting an action based on values from some other approximation methods or hand-crafted features (Gabillon, Ghavamzadeh and Scherrer, 2013).

1.2 Summary

The game of Tetris is reported to have about 7 X 2^{200} states and most of the controllers use some kind of approximating methods to evaluate the effectiveness of each action to generate policies. Creating effective controllers is also partially dependent on the implementation of the game and hence makes research comparison quite difficult. The evaluation function or score also presents a large variance between different plays of the game and it usually takes a large number of episodes to estimate the learning capability of a policy. Given the large state space, function approximators find it difficult to learn to generalise well across different situations of the game. A survey of existing literature shows that deep reinforcement learning methods have only been able to clear a few hundred lines and are sample inefficient. Gabillon, Ghavamzadeh and Scherrer (2013) also conjectured that Tetris is a game where good policies are easier to represent than approximating values indicating how good each state is. Algorta

and Özgür Şimşek (2019) suggests that Tetris should present an excellent test bed for learning the hierarchy of actions where certain actions might be optimised to achieve some sub-goals that maximize the overall average reward. One such sub-goal might be when the agent knows to stack up pieces in a configuration to clear four lines at once by dropping a single piece like I. However, handcrafting such sub-goals and including them a priori into the learning algorithm is almost impossible with such a large state space and virtually no two situations occurring twice uniquely. This project proposes that the Option-Critic algorithm from the Options framework of hierarchical reinforcement learning will be able to address these limitations. Option-Critic (Bacon, Harb and Precup, 2016) is an end-to-end approach for learning options (hierarchy of actions) and searches for the optimal policy in the policy space by optimizing a parameterized policy through stochastic gradient descent. Since the architecture is capable of learning options on its own, it does not require predefining sub-goals and can simultaneously learn these hierarchies and identify features of the state where they are applicable.

The project presented experimental results that showed empirically the existence of action hierarchies on a small version of Tetris. The agents during training have also learned to plan effective strategies around such hierarchies to extract high rewards, compared to agents planning without options. Some of the options were optimised on their own to build up certain blocks on the grid that would generate high rewards at later timesteps and all these were motivated without any extra reward functions. This showed that agents can identify useful temporally extended sub-goals in the game and utilise them to improve performance. Some of these actions are interpretable from the perspective of our intuition and have shown promise of being effectively scaled up to a bigger board.

1.3 Objectives

To summarise, this project will :

- Develop tabular and function approximation agents on a small version of Tetris and establish the idea that it is difficult to create good controllers for the game.
- Explore the application of a hierarchical RL framework called Option-Critic on the game of Tetris and how they can enhance the performance of the controllers compared to the previous agents.
- Develop effective ways to analyse and intuitively interpret the action hierarchies from the hierarchical agents.

Chapter 2

Literature Review

2.1 Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework used to model sequential decision-making in situations where the outcomes depend on past actions and environment dynamics. MDP assumes that a system follows the principle of the first-order Markov property such that the future state depends only on the current state and action taken. A finite MDP is described with: a finite set of states S, a finite set of actions A, the probability of transitioning to state s' from s as $P(s'|s, a)$ and a reward r obtained from completing action a in state s. The agent's goal is to learn an optimal policy π: S x $A \rightarrow$ [0, 1] that maximizes the expected cumulative reward over time. The expected cumulative reward starting from a state s and following policy π is defined as the value function of that policy for that state: $V_\pi(s) = \mathrm{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s]$ where γ is a constant called discount factor that depriciates the value of the reward over time, making immediate rewards more valuable than later ones. The same definition can be extended to define the action-value function when the agent in state s takes an action a and follows the policy afterwards $Q_\pi(s,a) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid \mathrm{s}_0 = s, a_0 = a]$.

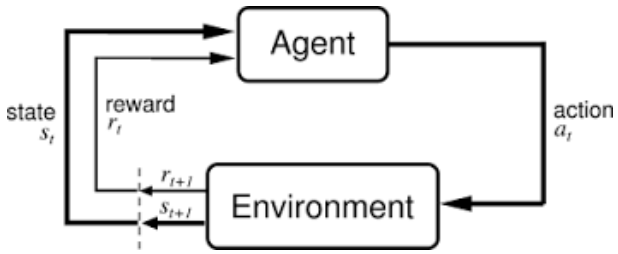


Figure 2.1: Agent-environment interaction in RL

2.2 Policy Gradient methods

Policy Gradient methods search for the optimal policy directly in the space of policies instead of trying to approximate the value functions from before. The policy is represented using some unknown function θ and methods such as stochastic gradient descent are used to approximate its value. This framework treats the problem of finding the best policy as an optimization problem by maximizing some quantity. Since the goodness of a policy can be evaluated by how much reward the agent gets by following it, the total expected reward from the game is set as

the objective that needs to be maximized in order to achieve a better policy. Small gradient updates are made on the policy function to push it towards the optimal value. These gradients are obtained by differentiating the objective with respect to the unknown function parameters. For a policy π_θ, the policy gradient theorem (Sutton et al., 1999) shows that the gradient of the expected average return for a certain start state s_0 is $\sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\partial \pi_\theta}{\partial \theta} Q_{\pi_\theta}(s,a)$ where μ_{π_θ} is the weighted average of each state visited during the trajectory from s_0. In practice, sampling is used to generate estimates for the gradient updates. Since the updates do not involve any state transition probabilities, it leads to a model-free learning method where we do not need to know the dynamics of the model beforehand.

2.3 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) decomposes a learning problem into a hierarchy of sub-tasks where a high-level action can, in turn, execute several low-level actions. Such low-level actions are often called primitive actions as they are the simplest and can be stacked on top to build a hierarchy. For example, normal day-to-day human actions involve a lot of complex hierarchies that we are not aware of. Like when asked to open a door we only focus on grabbing the handle and pulling it away, while internally our brain stimulates an array of signals that redirect the muscle fibres which in turn forces them to contract accordingly and so on. Here the sub-tasks are the transfer of brain signals, contraction of muscle fibres and twisting of finger joints etc. all of which together accomplish the primary objective. Thus for complex problems, HRL can help to solve them with a divide-and-conquer strategy where the smaller sub-tasks are solved and put back together for a more cost-effective high-level solution (Hengst, 2010). Barto and Mahadevan (2002) compares such abstraction to the subroutines of a processor which can call other subroutines or execute primitive commands.

The modelling of the environment as MDPs is extended such that each action from the set of possible actions A can call another action and the overall policy is defined over the hierarchy of such actions. These low-level actions can run on their own for extended periods. As such all actions can now be temporally extended over a time τ and the process becomes a Semi Markov Decision Process (SMDP). Fundamentally, while in MDPs all actions were executed and terminated in a single step, in SMDPs actions are spread over time. Following the action-value function of MDPs, the same for SMDPs can be defined as $Q(s,a) = r(s,a) + \sum_{s',\tau} \gamma^\tau \mathrm{P(s',\tau|s,a)} Q(s',a')$, where $P(s',\tau|s,a)$ is the joint probability of transitioning from state s to state s' when action a is taken and τ time is spent in state s. Here τ is defined as a random variable whose distribution depends on the policies and termination conditions of all other activities under action a (Barto and Mahadevan, 2002; Forestier and Varaiya, 1978).

2.3.1 Options Framework

S. Sutton (1998) introduced the term options to generalise the one-step actions in MDPs so that it can also include actions extended over time durations. The simplest option includes three components: a policy $\pi : S \times A \rightarrow [0, 1]$, a termination condition $\beta : S \rightarrow [0, 1]$, and an initiation set $I \subseteq S$. The option $<\pi,\beta,I>$ is available in a state s if and only if $s \in I$. When an option is invoked in such a state s, the agent follows the policy afterwards till its termination condition β is invoked. On termination of the option, the agent can select another option and the process continues. Such options are called Markov options since in state s, the action taken by the option policy only depends on the previous state and not on the history

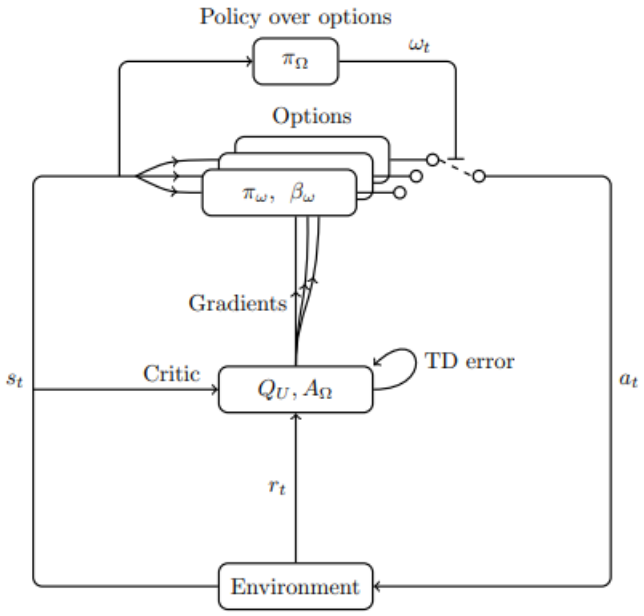


Figure 2.2: Option-crtic architecture. The option execution model is depicted by a switch over the contacts (Bacon, Harb and Precup, 2016)

of the agent (S. Sutton, 1998; Hengst, 2010). The termination condition and initiation set restrict the available scope for each option and can help in decomposing complex problems into sub-tasks. The policy that selects options in each state is called the policy over options and the policy that directs actions taken by each option is called the intra-option policy.

For an MDP with a certain set of options, the policy that chooses over those options and executes each of them till termination becomes an SMDP (S. Sutton, 1998). Based on the model of SMDP described earlier, we can then describe the set of temporally extended actions to options and the rewards generated will be the rewards accumulated during each option from its initiation till termination. As a result, the policy of each option is Markov in nature while the overall policy over options is Semi-Markov and the distribution of state transitions, rewards and time spent in each state depends only on the option and its history since starting in a state s.

2.3.2 Option-Critic architecture

Learning algorithms based on options require well-defined sub-goals or states that will have a high probability of termination. However, identifying such temporal abstractions or constructing sub-goals autonomously has proved to be difficult (Konidaris and Barto, 2009; Silver and Ciosek, 2012). Bacon, Harb and Precup (2016) introduced a new framework called Option-Critic which performs option discovery and option learning simultaneously using the policy gradient theorem.

Within this architecture, the intra-option policy and termination function is defined as π_θ and β_ν where variables θ and ν are differentiable parameters. For the purpose of simplicity, it is assumed that all options are available everywhere. The unknown parameter variables are updated using methods such as stochastic gradient descent. This reduces the problem of discovering options and termination policies to an optimization problem which can be solved

empirically by maximizing some quantity. The quality of each option is evaluated after every timestep by estimating the expected reward from the current state if the current option is followed. The simplest representation of these estimates can be the action value functions only now the actions have been replaced by options. These value functions are called the critic since they evaluate the effectiveness of each option. The termination function decides whether it would be useful to continue with the same option in the next state or to start with a new option. This enables the agent to switch options autonomously without external guidance. At the same time, each option and termination function gets to improve by updating their parameters to maximize the total expected reward. These updates are obtained by differentiating the reward objective with respect to θ and ν. The gradient updates are given by the intra-option policy gradient theorem and termination gradient theorem (Bacon, Harb and Precup, 2016). This enables an end-to-end approach to learning options without introducing any a priori knowledge integrating both option discovery and improvement.

Chapter 3

Methodology

3.1 Implementation of Tetris

As discussed in section 1.3, the traditional 20 X 10 size board takes a long time to complete episodes of play and there is usually a lot of variance in the scores obtained. These make developing and experimenting with algorithms difficult and subject to very long durations of waiting before getting results. Hence for the majority of the scope of this project, experiments have been carried out on a mini version of the bigger grid. Two versions, 5 X 4 and 10 X 10 grids have been considered here.

For the 5 X 4 grid, only two pieces of the tetrominoes namely the I and L piece is available and all other pieces have been discarded. Furthermore, since the 5 X 4 grid only has 5 rows, each piece is composed of three cells instead of the more traditional four-cell pieces. Additionally, on the small grid making all the pieces available would result in them being stacked upon one another quickly and the game would end in two or three timesteps. Hence due to this limitation of the game dynamics, it will be impossible to learn any policies that would clear lines. These two considerations help to preserve the complexity of the environment without compromising on its nature. At the beginning of every timestep of the game, the board evaluates all the possible positions and configurations on the grid where a tetromino can be stacked on top of the existing pieces on the board. This generates the legal moves which are performed by rotating and translating the tetromino along the rows. The game ends when the agent is no longer able to fit a tetromino in the 5 X 4 grid. The reward is defined such that maximizing the expected sum of the reward would maximize the score (number of lines cleared) from the game. The code for the implementation of mini Tetris in this project has been adapted from nuno faria (2021) with modifications made that would suit the purpose of this project. It uses a convex function of the number of lines cleared as the reward function: (1 + lines-cleared^2 X width of the board). To summarise, the mini Tetris can be formulated as an MDP where the state is defined by the current board configuration plus the falling piece, the action set is the set of all possible legal action for the current piece, an initial state of an empty board at the beginning with an I or L as the falling tetromino, state transition probability of moving from the current board configuration to the next configuration once the falling piece has been stacked and a reward of (1 + lines-cleared^2 X width) from the board after each action.

The board is represented with an array of length 20 where each element can be a 0 or 1. Empty cells of the grid are 0 and filled cells are 1. The pieces are indexed from 0 to n - 1,

where n is the number of pieces. Since we are considering only the I and L tetromino, I is 0 and L is 1. Thus state of the board is an array of length 21 where the first 20 elements are from the grid and the last element is the tetromino index. Discrete actions are represented as tuples (x position of translation, rotation angle). Considering the centre of the board as the origin, a tetromino can move left (-) or right(+) by the number of cells present. For example, +2 indicates moving the current piece two cells to the right from its position, -1 indicates moving it 1 cell to the left and 0 indicates keeping it in place. There are four possible rotations 0°, 90°, 180°, 270°.

For experiments done on the bigger 10 X 10 board, the length of the pieces is restored back to 4 cells and all 7 tetrominoes are considered. The state vector size is increased to 101, where the first 100 elements are from the Tetris grid.

3.2 Tabular Q Learning

The first experiment conducted is to learn a tabular Q-agent on the 5 X 4 board that will act as a benchmark from which further agents can be compared and contrasted. The agent maintains a look-up table indexed by the state action pair and stores the expected total reward from that state as the q-value. All values are initialized to zero. The agent visits a state s_t, takes action a_t, observes the reward r_{t+1} and the next state s_{t+1} and updates the look-up table by its TD-error (Sutton, 1988) defined as $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$, where α and γ are the learning rate and discount factor. The values of α and γ has been set to 0.2 and 0.9 respectively. The agent follows an epsilon-greedy policy where it takes a random action with a probability of 0.15 and rest of the time chooses the action that has the maximum value for $Q(s_t, a_t)$ in that state. Since this is performed on a smaller board of Tetris with only two pieces, it is expected to learn quickly to clear lines and should do considerably well compared to a random agent. It is hoped that this method will establish a good baseline score against which the performance of more complex agents can be evaluated. The algorithm runs for 3000 episodes with each episode terminating when the grid is filled and there is no space for the next piece.

3.3 Deep Q Learning

The second set of experiments is run by replacing the tabular look-up table with deep neural networks. The setup is split into 3 different experiments, where the neural networks have been constructed differently. For the first experiment, the input layer is a vector containing the board state, current piece, action tuple x and rotation degree. Since there are four rotations they are represented numerically as 0, 0.33, 0.66 and 1. Followed by a dense layer of 64 neurons with relu activation and another dense layer with 1 neuron and linear activation. During a single episode, at each timestep, if there are n legal actions then n individual vectors of length 23 are processed in a batch with the output of size n X 1. Each value in the output represents a score for that state-action pair and the action with the maximum value is executed as the desired action. The model is trained following the architecture discussed in Mnih et al. (2013) with experience replay and a target network for stability. Experience replay is where the dqn agent stores in memory past episodes of the game that have already been played and after certain timesteps, a batch of these experiences sampled randomly are used to train the agent(Mnih et al., 2013). The target network is a second copy of the same dqn agent but

its parameters are updated periodically to improve learning stability. Here it is updated after every 500 timesteps where its weights are replaced by the current network and a memory with 100000 past episodes is used for experience replay with a batch size of 32. The agent follows an epsilon-greedy policy with a constant epsilon value of 0.15 as before and a discount factor of 0.99. The model is trained for 2000 episodes and optimized with Adam (lr - 0.001) and huber-loss.

For the second set of experiments, the input layer is modified such that it only gets the current state of the board and the next piece and outputs n different values where n is the number of possible actions. This representation is more similar to a normal Q-agent and might produce better results since it generates scores for all actions instead of evaluating each of them individually. For a 5 X 4 board, with the centre (second cell) as the origin, it can only move a position to the left or two positions to the right and there are 4 possible rotations hence n is 16. Since not all moves are legal moves under different circumstances, the agent filters out the legal actions and executes the one with the maximum value. The input layer is fed to two successive dense layers of 64 neurons each and relu activation and a final linear layer with 16 neurons. The target network is updated every 100 timesteps and the experience replay uses a memory of size 100000 with batch size 32. The agent follows an epsilon-greedy policy where the value of epsilon is decayed exponentially by a factor of 0.998 after every timestep starting from 1 and the minimum value is 0.01. The discount factor is kept the same as before. The model is also trained for 2000 episodes and optimized with Adam (lr - 0.001) and huber-loss.

The third experiment is run on a larger grid of size 10 X 10 with the same network architecture as the last one. The input layer has 101 neurons consisting of the board state and the current piece. This layer is fed into three successive dense layers with neurons 256, 128 and 64 respectively and all with relu activation. The final layer has 40 neurons with linear activation indicating the number of possible actions for any piece. The loss function and optimizer are the same as before. The coding implementation of Tetris used for this experiment is from Bonnet et al. (2023). The network was trained for 2,500 episodes with an epsilon-greedy policy and a decay factor of 0.9998. The target network is updated after every 200 timesteps and the experience replay has a memory size of 500000 and batch size 64.

It is hoped that for the 5 X 4 version of the board, DQN should perform as well as the tabular Q-agent if not better. Q-learning is expected to have more stable results considering the small state space and should converge faster than DQN. Deep RL methods have proved to perform very well on an existing array of video games like Atari. However, a survey of previous literature suggests that this methods have hardly been able to show any good performance on Tetris. Furthermore, the presence of a lot of hyperparameters makes it difficult to effectively optimize the learning for the best results. There is a long delay between actions and a line being cleared, and during this time a lot of random actions can be fed into the memory of the experience replay which makes it difficult to track back which actions led to the reward. Hence learning algorithms that can plan a strategy to improve such sparse rewards should perform better. It is expected that these experiments would establish the idea that the complexity of Tetris cannot be resolved by only using better function approximators but it requires the agent to have temporal knowledge planning towards different sub-goals that will increase the final score. This will lead way to moving on from agents with only primitive actions to agents planning with a hierarchy of action sets.

Algorithm 1: Option-critic with tabular intra-option Q-learning

$s \leftarrow s_0$
Choose ω according to an ϵ-soft policy over options $\pi_\Omega(s)$
repeat
  Choose a according to $\pi_{\omega,\theta}(a \mid s)$
  Take action a in s, observe s', r

  1. Options evaluation:
  $\delta \leftarrow r - Q_U(s, \omega, a)$
  if s' *is non-terminal* **then**
    $\delta \leftarrow \delta + \gamma(1 - \beta_{\omega,\vartheta}(s'))Q_\Omega(s', \omega) + \gamma\beta_{\omega,\vartheta}(s') \max_{\bar{\omega}} Q_\Omega(s', \bar{\omega})$
  end
  $Q_U(s, \omega, a) \leftarrow Q_U(s, \omega, a) + \alpha\delta$

  2. Options improvement:
  $\theta \leftarrow \theta + \alpha_\theta \frac{\partial \log \pi_{\omega,\theta}(a \mid s)}{\partial\theta} Q_U(s, \omega, a)$
  $\vartheta \leftarrow \vartheta - \alpha_\vartheta \frac{\partial \beta_{\omega,\vartheta}(s')}{\partial\vartheta} (Q_\Omega(s', \omega) - V_\Omega(s'))$

  if $\beta_{\omega,\vartheta}$ *terminates in* s' **then**
  choose new ω according to ϵ-soft$(\pi_\Omega(s'))$
  $s \leftarrow s'$
until s' *is terminal*

Figure 3.1: Option-Critic Algorithm (Bacon, Harb and Precup, 2016)

3.4 Option-Critic method

The Option-Critic system has two components, namely the actor part of the system which is the policy over options, intra-option policy and termination functions and the critic part which contains the value functions. The intra-option policy and termination functions are parameterized by unknown variables and updated through gradient descent. The option over policies parameter and the critic value functions are updated using intra-option Q learning rules (Sutton, Precup and Singh, 1999). These rules allow to learn a model of the option from experience and extend traditional one-step Q learning updates to actions that are temporally extended. It is represented as $Q(s, o) \leftarrow (1 - \beta)Q(s, o) + \beta(r_{t+1} + \gamma \max_{o'} Q(s, o))$. Here $Q(s, o)$ is the option value function, β is the termination probability of the current option o and γ is the discount factor. The agent starts from a state s_0 with an option sampled from the policy over options, takes action according to the intra-option policy, observes the reward and moves to the next state. All the parameters are updated after each timestep. The hyperparameters are the learning rates for the value functions and the number of options.

The experimental setup is split into a few different configurations, based on the kind of distributions on which the policies are parameterized and the number of options specified as hyperparameters. All experiments are carried out with a tabular setting for value functions and parameters. For the first experiment, the policy over options is parameterized as epsilon-greedy, the intra-option policy as softmax and the termination functions with sigmoid function. The value of epsilon is decayed exponentially by a factor of 0.99998 after every timestep to a minimum of 0.01 from 1 for 250000 episodes. All learning rates are initialized to 0.01 and a discount factor of 0.99. For the 5 x 4 board with two pieces, three different temporally

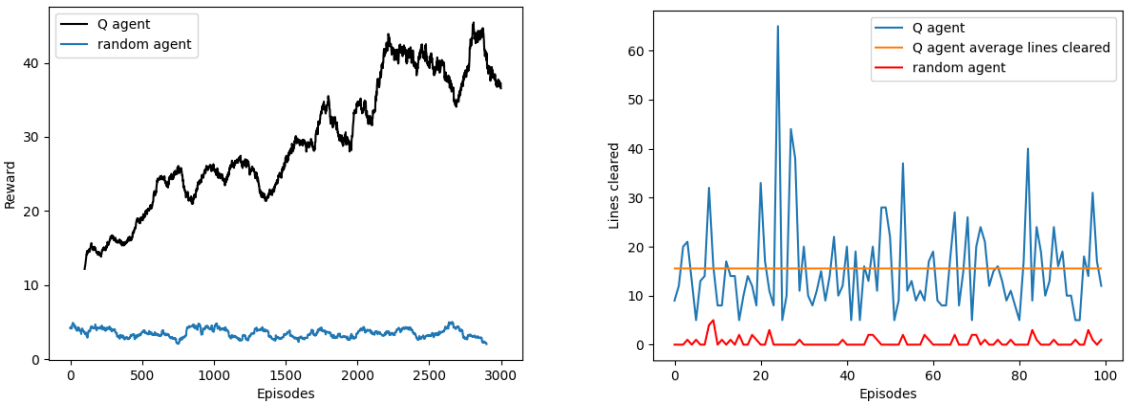
extended settings have been experimented with 2,3 and 5 options. For the second experiment, the policy over options is replaced with a softmax policy keeping all other parameters the same as before. The softmax distribution experiments are run only on the 5 x 4 board with two tetrominoes and temporal settings of 2 and 3 options.

The main objective of the experiments here is to identify a certain hierarchy of actions that we assume is present in Tetris and try to interpret them. If our assumption is correct, an agent with the ability to represent temporally extended knowledge should learn an effective strategy to obtain high rewards. It is hoped that all of these agents with hierarchical actions will outperform the tabular Q-agent and the DQN. However, another challenge is to find a way to interpret the learned options that can match our intuitions. For experiments with two pieces, one intuition can be if an agent has 2 options then each option might specialize to each tetromino. Another possibility is that different options work towards achieving different sub-goals that effectively increase the overall reward. Furthermore, the option over policies and termination functions should learn to initiate and end which option when based on dropping which piece would clear more lines. The frequency of primitive actions taken by each option can also be analysed and compared to other options to verify if their actions set up some sub-goal that can clear lines under certain special circumstances. Increasing the number of options increases the complexity of the SMDP but should help in abstracting more temporally extended knowledge and better planning of strategies. The policies of agents can be analysed extensively by looking at different episodes of gameplay to see if any intuitive skills are being observed. The agents can be forced to play full episodes of the game with only one option available. This will help to judge the effectiveness of each individual option and assess how the policy changes when the agent is allowed to plan with them all. However, if at all hierarchies are set up they might not always be interpretable from a human perspective. Additionally, human players in Tetris are capable of making fast decisions without exhaustively computing all possible actions and outcomes. This proposes that decision-making in nature is more sample-efficient and has some underlying hidden component to it. The structure of this latent knowledge might or might not be hierarchical in nature, however, it might be interesting to analyse if the learning agents here discover any such sample-efficient planning. It is also important to point out that in our settings, the agent has to learn all these skills and at the same time apply them effectively to solve the game. This might make it difficult to produce good results in an efficient number of trials.

Chapter 4

Results and Discussion

4.1 Baseline from tabular methods



(a) Learning curve.

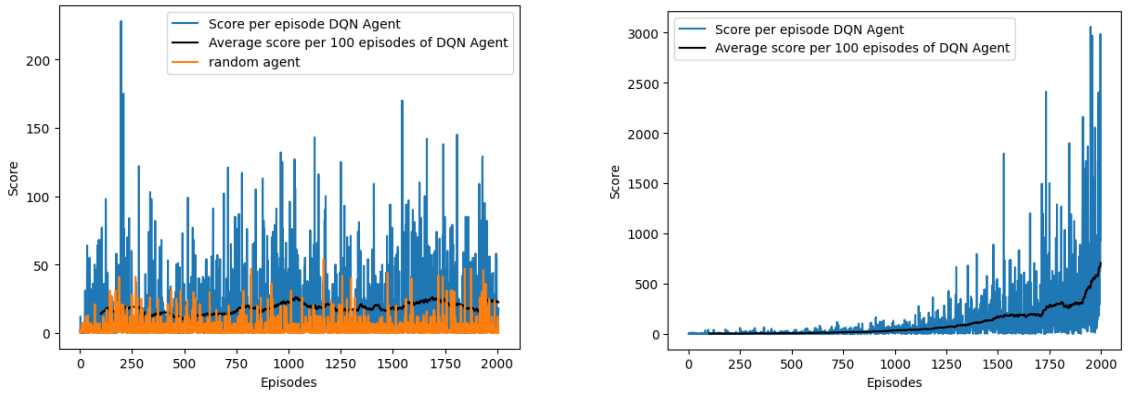
(b) Number of lines cleared.

Figure 4.1: Results from tabular Q-agent. For (a) the agents ran for 3 independent trials from the same initial conditions and their performance was averaged. The line shown is the rolling average of that mean over 100 episodes.

The learning curve for a tabular Q-agent on the 5 x 4 board is shown in Fig 4.1a. The reward function here is (1 + lines-cleared^2 x width of the board). The total number of states explored for epsilon of 0.15 was around 6,000. After about 2,000 episodes, the agent appeared to converge to an optimal policy. The training was stopped following another few hundred episodes since no improvement in the policy was observed. In order to make the result more interpretable, it has been compared to a random agent. The epsilon-greedy policy clears around 15 lines on average when epsilon is set to zero whereas an agent with a random policy only clears 1 or 2 lines at best. As hypothesised before in Section 3.2, tabular methods on a small board can quickly learn an optimal policy that is just better than a random player. The project establishes this performance as the baseline and further agents are expected to achieve higher scores.

4.2 Function Approximation results

The results in Fig 4.2 show the learning curves for two different DQN agents on the 5 x 4 board. The plot on the left is from the agent whose input had the board configuration, current piece, x position and rotation value. The plot on the right had only the board configuration and current piece as input and 16 nodes as output each assigning a value to a possible action. The former agent does slightly better than a random agent while the latter one shows considerable learning after around 1000 episodes. Changing the input state architecture makes a dramatic



(a) DQN with input as board configuration, current piece, x position, rotation degree

(b) DQN with input as board configuration and current piece

Figure 4.2: Learning Curves for DQN on the 5 x 4 board

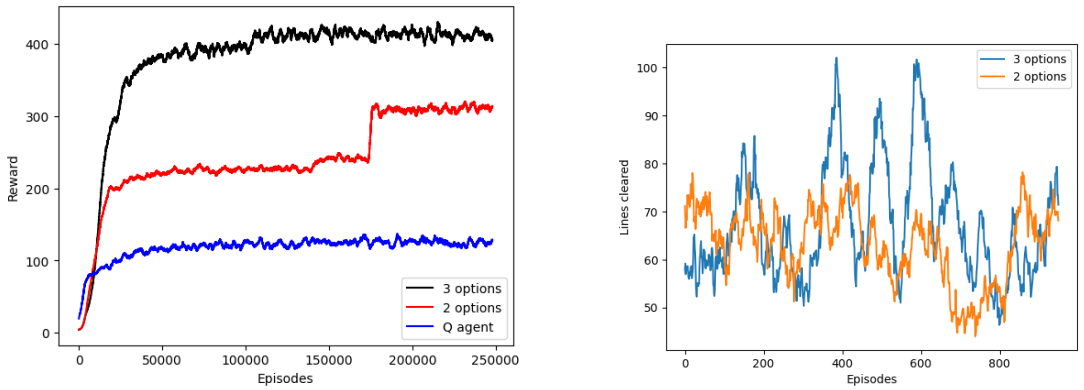
improvement in the performance by almost a magnitude of order 10. This improvement in performance might be attributed to the fact that the second agent only has to learn a good approximation of states to produce which actions are reasonable instead of independently scoring all state-action pairs. However, since the project focuses mostly on hierarchical agents and these experiments were done to have a general idea of deep RL agent performances in the game, no further analysis of this behaviour was done. Additionally, the performance on the 10 x 10 board has not been shown here since it failed to show any learning after 2000 episodes. This is not surprising given that a bigger board is much more difficult to solve and would require training for several more episodes to do even marginally better than a random agent.

4.3 Options

4.3.1 Policy over options parameterized as softmax distribution

Fig 4.3a shows the learning curve for an agent trained with 2 options and an agent trained with 3 options. These results are compared to a tabular Q-agent. The intra-options policy and policy over options here are set as softmax and the termination functions are parameterized with sigmoid function. For both experiments, we can see that the average reward per step is much higher than the Q-agent with primitive actions from section 4.1. It can be hypothesized that initially, the 3 options model explores more trying to improve the policy whereas for 2 options it favours exploiting and following the current model's optimal behaviour. One piece of evidence in support of this assumption is that the total states explored by the 3 options model at the end were reported to be around 13,000 while for the 2 options model, it was around 10,000. This results in a better overall policy for the former. As a result, the 3-options

model achieves a higher score by clearing more lines during gameplay than the 2-options model as seen in Fig 4.3b. From the plot Fig 4.3a, we can see the 2-options model stops exploring after around 10,000 episodes and follows the optimal actions directed by the existing model. However, the 3-options model chose to keep exploring till almost 50,000 episodes before converging. Furthermore, I presume the presence of the 3rd option adds a more stochastic nature to the overall planning and encourages the development of better temporal abstraction from the game dynamics, however, with only 2 options available it slowly degenerates(Kamat and Precup, 2020) into a single option that dominates the other one. Further discussions on this have been provided while interpreting the options below.



(a) Learning curve.

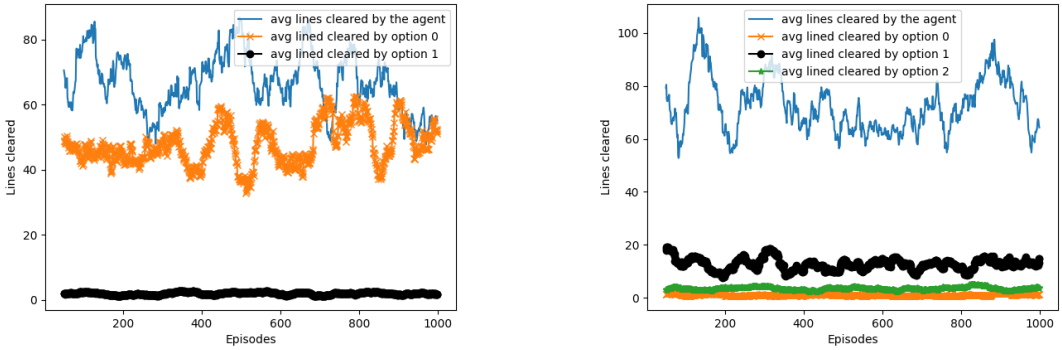
(b) Number of lines cleared during gameplay.

Figure 4.3: Options model with softmax policy. For (a) all the agents ran for 3 independent trials from the same initial conditions and their performance was averaged. The line shown is the rolling average of that mean over 2000 episodes.

Interpretting the action hierarchies with 2 options. The dynamics of the Tetris environment is a difficult field to analyse and explain the level of hierarchies that can be present. The models can learn arbitrary hierarchical abstractions from the game that result in well-planned policies. However, they might not match with our intuitions and can be difficult to identify. Since the small 5 x 4 board presents a more controlled environment, there are a few ways we can try to look inside such temporal abstractions. In order to do so the agents were forced to play the game with only one option available to them and the number of lines cleared in each situation has been plotted in Fig 4.4.

In the 2-options model, as observed in Fig 4.4 option 0 has a performance comparable to the overall policy of the agent whereas the policy represented by option 1 is almost equivalent to that of a random agent. My hypothesis is that the hierarchy set up here is such that the agent learnt to play the game with option 0 but it hands over the control to option 1 in certain special cases where there are chances of setting up a high reward. As a result, the option 0 intra-option policy is a near-optimal policy to that of the overall agent however it fails to gather those extra rewards. The intra-option policy for Option 1 has very little representation of the state dynamics and hence behaves as a random agent when forced to play alone. In contrast to this, if we look at the plot on the right of Fig 4.4, each of the individual options for the 3 options model has performances that of a random agent or slightly better. However, all those options together generate higher scores during gameplay than the 2-options model. This further supports the idea discussed earlier that during training for the 2-options model, most of the time the hierarchy was degenerated to option 0 only which dominated option 1.

The 3-options model does not suffer from this issue and hence learns to plan better with its knowledge of temporal abstraction. This also explains why option 0 for the former model lasts several timesteps during gameplay while option 1 lasts for only 1 or 2 timesteps.



(a) Model with 2 options

(b) Model with 3 options

Figure 4.4: The agents were forced to play the game with single options only

Further evidence for this might be observed by peeking into the gameplay episodes. For this purpose, several episodes were analysed and some of the results have been presented. Fig 4.5, shows the desired placement for the L and I pieces on the board at the time of start. The starting option is always 0 for both cases. In Fig 4.6a, we look at the same gameplay after some time. The agent has followed option 0 all this time till the 4th timestep in the figure,

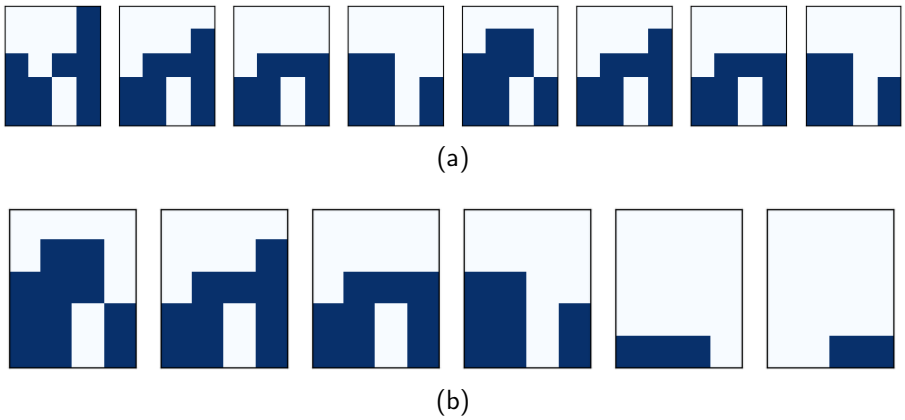
(a) (b)

Figure 4.5: The images shown here are taken from different gameplay trajectories. Each single grid depicts a timestep in the game and the successive grids show the next timesteps. The left figure shows the start of a new game with the first piece being L and the right one being I. Irrespective of the first piece, the agent always starts the game with option 0 and continues to follow the same option for the next timesteps. Fig (a) and (b) show the desired placement for those pieces.

where it suddenly switches to option 1. This is probably motivated by the fact that the board configuration here gives a choice to gain extra reward by clearing 2 lines if the next piece is an I. We can see that happen in this instance. This can often be observed when option 0 tries to set up a high-reward situation and gives control over option 1. The presence of such switching behaviour for the overall policy indicates that the agent has been able to identify certain useful skills that can generate high rewards. As a result, it has planned its policy to optimise the gameplay to lead to these states whenever possible. It is worthwhile to mention that the agents only have information about the current piece and all these actions to set up certain skills are taken based on that. Hence the successful completion of that move also depends

Figure 4.6: Each grid is a single timestep followed by the next from a random trajectory of the game. The trajectory here shows an agent that was following option 0 for the past timesteps switches to option 1 for the 4th grid since there is an opportunity to clear two lines by an I piece. After that, it returns back to option 0 for the next actions.

on whether the next piece is the same as predicted by the option. A few more examples are discussed further to demonstrate this.



(a)

(b)

Figure 4.7: The figure shows a sample trajectory of the game that starts with the first grid in Fig (a) and continues till the last grid of Fig (b).The agent switches to option 1 in the 4th, 8th and 12th grid since there is a chance to secure high rewards by clearing two lines. The move is successful only if the next piece is I. Option 0 is active elsewhere.

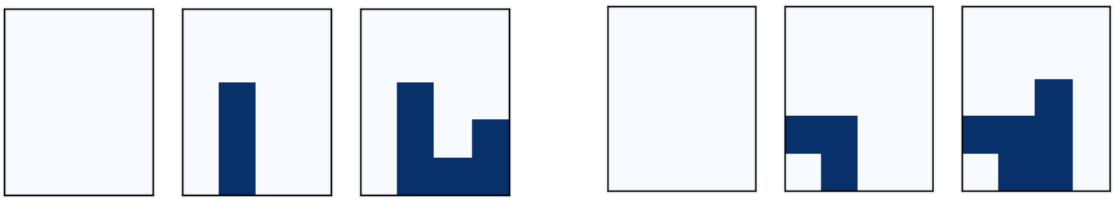
Fig 4.7 shows that the agent tries to set up the board from a different initial condition and at the 4th timestep it again switches to 1 from 0. However, the next tetromino is the L piece and hence it terminates option 1 and switches over to option 0 again. The process is repeated two more times, and in the third instance, it clears two lines when the next piece is an I. This also helps to free up more space on the board for new pieces.

One more example is shown in Fig 4.8, where the agent switches to option 1 from option 0 in the first two timesteps and switches over to option 0 after that. The agent is able to identify that under option 1, a board configuration like this can be useful in extracting high rewards with certain pieces. In this case, the I piece drops first followed by the L and the setup is successful in clearing two lines. From these results, it can be suggested that a two-level hierarchy has been set up here. At the top level, the policy over actions is concerned with playing the game to clear lines and keeps shuffling between options skilfully to achieve the maximum reward. At the bottom level, option 1 has primitive actions that correspond to that skill which achieves high rewards only whereas option 0 corresponds to actions that can keep the game going most of the time without it terminating. For the overall policy, a potential sub-goal is the set up of blocks to have high rewards and it employs option 1 whenever there is an opportunity for that. It is worthwhile to point out that the similarity of option 0 to the

Figure 4.8: Each grid is a successive timestep from a random gameplay. Actions are directed by option 1 in the 1st and 2nd grid here.

overall policy exists because of the degeneration problem discussed earlier, and prevents the agent from constructing a more refined structure to the hierarchy.

Interpretting action hierarchy with 3 options. With 3 options available, the model planning for the start state is different from the one shown in Fig 4.5 for 2 options. For the I piece, it has a very high probability of starting with option 2 for the first timestep and handing over the control to option 1 for the successive states. When the first piece is the L piece, it follows the reverse policy. This is shown in Fig 4.9. Furthermore, as earlier discussion established the policy over options learnt here is much more stable such that each option does not score well when forced to play alone as seen but is well integrated into the overall hierarchy. This resulted in improved temporal abstraction of knowledge to set up the hierarchy. Hence the average performance of the agent is better.



(a) I tetromino (b) L tetromino

Figure 4.9: The starting states for the 3 options model. For a game with the first piece as I, the agent chooses option 2 first (the first grid here on the left) and reverts back to option 1 for the following timesteps. If the first piece is L, it chooses option 1 (the first grid on the right) and reverts back to option 2.

However, this makes interpreting each individual option difficult from a human intuition perspective. Analysing the game trajectories as before, one interesting observation was that while most of the game was played either with option 1 or 2, at times the agent switched to option 0 when there was an opportunity to clear holes stuck beneath the current stack. As shown in Fig 4.10, the agent switches to option 0 to clear the blocked holes at the bottom of the grid and once the space is freed the control turns back to option 1. It might be possible that option 0 has learnt useful skills that can free up such blocked spaces on the grid otherwise unreachable. This was observed in a few cases but due to the lack of further evidence, it cannot be confirmed whether any such hierarchy exists for the 3 options.

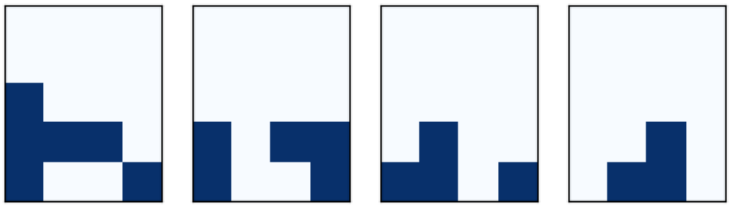
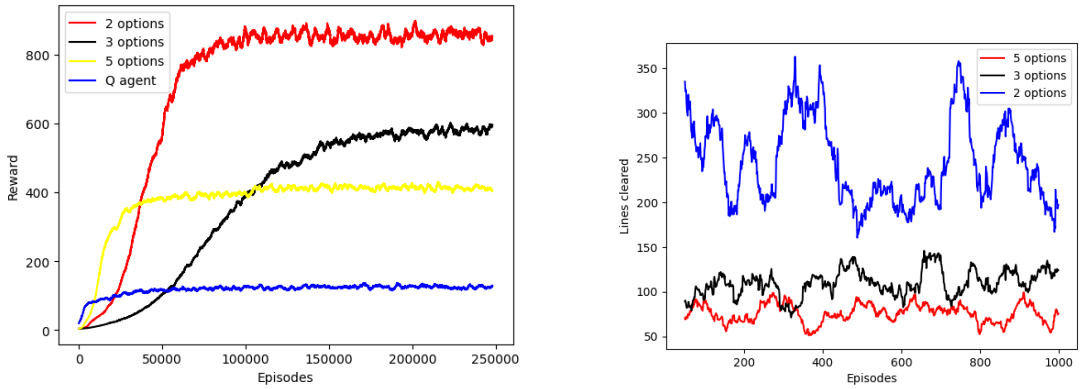


Figure 4.10: Game trajectory with each frame representing a timestep. Option 0 is active for the first two time steps and option 1 elsewhere.



(a) Learning curve.

(b) Number of lines cleared during gameplay.

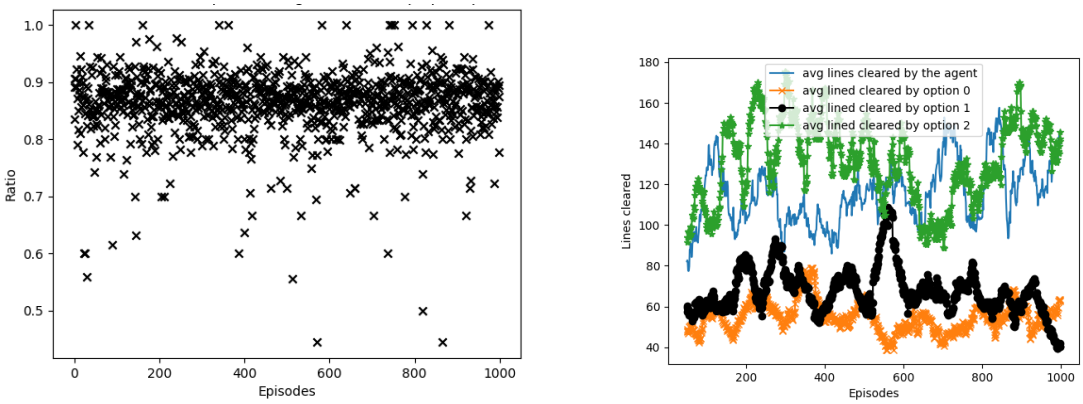
Figure 4.11: Options model with epsilon-greedy policy as policy over options. For (a) all the agents ran for 3 independent trials from the same initial conditions and their performance was averaged. The line shown is the rolling average of that mean over 2000 episodes

4.3.2 Policy over options parameterized as epsilon-greedy

For the following experiments, the policy over options has been replaced by epsilon-greedy from softmax distribution before. The intra-option policy is softmax and the termination functions are sigmoid functions, the same as in earlier experiments. Fig 4.11a shows the learning curve for agents trained with 2,3 and 5 options separately. All of them have higher average rewards than a tabular agent. As observed the agent with 2 options significantly outperforms all other variations. The quality of the optimal policy declines as the number of options is increased. This is in contrast to what we observed before for agents with softmax as policy over options. The optimal policy there improved by increasing the number of options as this prevented the policy from degenerating. The best model is thus the agent with epsilon-greedy as policy over options with 2 options. It also achieves higher scores than the previous best agent which had softmax as the policy with 3 options. The average lines cleared by the former is around 250 while the latter only clears around 100 lines.

The poor performance of the agents with 3 and 5 options can be reasoned as either that the policy over options deteriorates over time such that it learns multiple options that have very similar behaviour or the individual options themselves have gained irrelevant skills for Tetris. A survey of existing literature also shows that challenges like this have been observed in end-to-end learning methods that involve hierarchy (Kamat and Precup, 2020; Klissarov and Precup, 2021).

As the number of options is increased, each option terminates after few timesteps only and



(a) Ratio of number of time the agent switches options in an episode to total timesteps in the episode

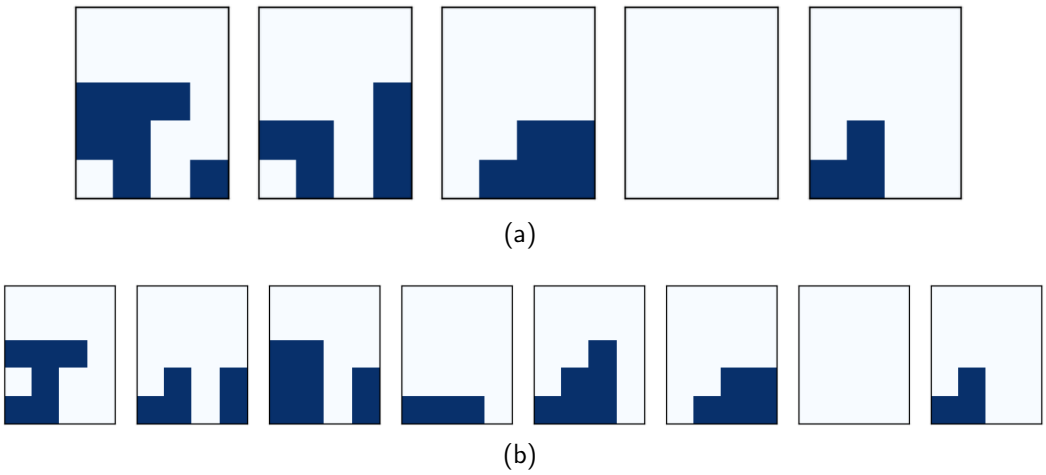
(b) Model with 3 options

Figure 4.12: Option degeneration

shrinks too much over time. As a result, the policy over options decides to choose only one particular option that has generated high rewards from previous experience. The option set collapses to one option learning to solve the entire problem and dominate others. These kinds of degeneracies negatively impact the overall policy as seen in Fig 4.11 and hurt the performance of the agent (Kamat and Precup, 2020). The evidence for this can be observed if the agent with 3 options is forced to play the entire game with individual options. As seen in Fig 4.12a, it presents the ratio of the number of times an option was changed in an episode to the total timesteps. The ratio lies between 0.8 to 0.9 which signifies that the policy terminates options almost every one or two timesteps. Thus options are being collapsed almost into single-step primitive actions. From the plot on the right 4.12b, we can see option 2 when let to play alone scored higher than the overall agent which can be evidence for the fact that during learning the option set degenerated to option 2 most of the time. Hence option 2 developed an optimal policy to solve the game on its own and dominate others. As a result, the overall policy never learns any useful temporal abstraction and hurts the performance of the agent.

It is also worthwhile to point out that the tendency for options to degenerate for softmax agents decreased with increasing the number of options, while the opposite trend has been observed in epsilon-greedy agents. The simplest explanation for this can be that as the number of options increases for the greedy models, the agents start taking certain options epsilon times frequently which leads them to redundant states that might not improve the overall existing policy. This also hinders the exploration of the rest of the states. Since epsilon is gradually decreased exponentially starting with a very high value, it can be said that this happens quite often in the beginning. Hence the 3 options and 5 options model with greedy policy degenerates much faster.

Interpretting action hierarchy with 2 options. With 2 options available, the nature of the hierarchy is similar as observed for the softmax agent. Based on observations, it can be said that option 1 tries to set a two-staired board configuration as seen in Fig 4.13. This provides an opportunity to clear two lines as well as empty the existing board by dropping an inverted L. This kind of situation can be observed frequently during gameplay trajectories. For example, Fig 4.13a and b are different trajectories where the agent starts with two different board configurations but leads to a setup where it can perform this skill. For the figure on the



(a)

(b)

Figure 4.13: Game trajectories. Fig a and b are two different trajectories. Each grid shows a single timestep from a random game followed by successive timesteps.

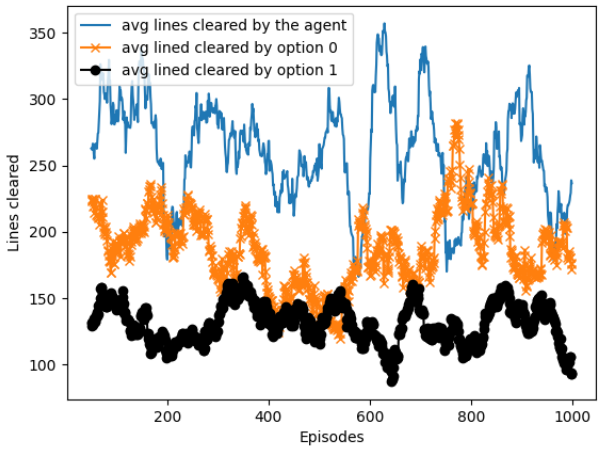


Figure 4.14: Model with 2 options. The number of lines cleared when forced to play with each option only.

top, we can see the agent has been following option 0 till the 2nd grid where it switches to option 1. This helps set up the stair-shaped block on the right corner. At the next timestep, option 1 places an inverted L to clear the lines. After that, the agent switches back to option 0. Similarly, the bottom figure shows a separate trajectory where option 1 is active during the 4th, 5th and 6th grid states. Here the nature of the skill setup and the planning at a temporal level suggests that the overall policy has learned to plan in a hierarchical nature that is comparable to the policy of the softmax agent with 2 options. The agent plays the game with option 0 most of the time but turns to option 1 in certain special cases where there are chances of setting up a high reward by performing some skill.

Also, we can see from Fig 4.14 that when forced to play with single options, each option has its own intra-options policy but none of them is comparable to the overall policy. This suggests that they do not suffer from the same degeneracy issues that have been discussed earlier. As a result, the overall policy is optimal while options on their own are sub-optimal policies.

Chapter 5

Conclusion and Future Work

This project set out to investigate the results of hierarchical reinforcement learning through option-critic methods on Tetris and identify temporally extended actions if observable. The results have empirically shown that such hierarchies are present and useful strategies can be planned around them to extract high rewards. Agents exploiting the knowledge of such temporal abstraction scored better than agents planning with only primitive actions. The average performance of a tabular Q-agent was around 15 lines cleared. Function approximation agents were able to clear at best 80 lines. All the agents that were trained to exploit temporal knowledge of the game dynamics had scores ranging between 100 to 350 lines depending on different experimental variations. The best hierarchical agent was trained with epsilon greedy as the policy over options with 2 options. Some of the options were interpreted intuitively to discover that they often learn specific skills that can guarantee high rewards. For example, one of the options had learnt to set up a stair-shaped block on the right corner which can be cleared by dropping an inverted L piece from the left to clear two lines and get an empty board. The agents are capable of planning policies that will incorporate these skills.

The immediate direction for future work would be to apply hierarchical methods on a larger grid of Tetris. The small environment here allowed us to discover interpretable action hierarchies that show promise of being applicable to a scaled-up board like a 10 x 10 grid. Function approximations such as deep neural nets can be used to replace the tabular settings since the state space would be large enough and not representable in tabular form. Skills learnt here should be applicable to a larger board where each option can learn to focus on a different objective and that would help to address the sparse nature of the overall reward in this game. Furthermore, as it turns out training with options hierarchy is sample inefficient compared to algorithms with just primitive actions. Humans can learn to have moderate performance on Tetris without playing several episodes. This is because, in the option-critic setup, an agent has to simultaneously discover essential skills, evaluate them and apply them to solve the overall objective. All these aspects might contradict each other initially during the learning process. For example, options might lead to setting up blocks on the board that do not generate any immediate result initially but at some later timesteps produce a high reward by performing some skill. An important avenue for future work can be to understand this gap in performance and develop methods that can learn to balance between both these aspects of learning.

Lastly, as seen previously for models with 3 and 5 options, the policy quickly degenerates to singular options which hurts the performance of the overall agent. This can be improved by encouraging exploration and constructing a diverse set of options. Future work can also

look into improving the termination conditions so that the options do not shrink too much over time. One method suggested by Kamat and Precup (2020), mentions adding an extra reward to the overall objective of the game that would encourage options to have more varied behaviour. The pseudo-reward is designed with the objectivity of maximising the divergence between action distributions of each option. They also tweak the termination functions in order to encourage these diverse options.

The dissertation presented here contains around 9529 words.

Bibliography

Algorta, S. and Şimşek Özgür, 2019. The game of tetris in machine learning. 1905.01652.

Bacon, P.L., Harb, J. and Precup, D., 2016. The option-critic architecture. 1609.05140.

Barto, A. and Mahadevan, S., 2002. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems: Theory and applications* [Online], 13. Available from: https://doi.org/10.1023/A:1025696116075.

Bertsekas, D.P. and Tsitsiklis, J.N., 1996. *Neuro-dynamic programming*. 1st ed. Athena Scientific.

Bonnet, C., Luo, D., Byrne, D., Surana, S., Coyette, V., Duckworth, P., Midgley, L.I., Kalloniatis, T., Abramowitz, S., Waters, C.N., Smit, A.P., Grinsztajn, N., Sob, U.A.M., Mahjoub, O., Tegegn, E., Mimouni, M.A., Boige, R., Kock, R. de, Furelos-Blanco, D., Le, V., Pretorius, A. and Laterre, A., 2023. Jumanji: a diverse suite of scalable reinforcement learning environments in jax [Online]. 2306.09884, Available from: https://arxiv.org/abs/2306.09884.

Burgiel, H., 1997. How to lose at tetris. *The mathematical gazette* [Online], 81(491), p.194–200. Available from: https://doi.org/10.2307/3619195.

Demaine, E.D., Hohenberger, S. and Liben-Nowell, D., 2002. Tetris is hard, even to approximate. cs/0210020.

Fahey, C., 2003. Tetris ai [Online]. Available from: http://www.colinfahey.com/tetris/tetris.html.

faria nuno, 2021. tetris-ai. https://github.com/nuno-faria/tetris-ai.

Forestier, J.P. and Varaiya, P., 1978. Multilayer control of large markov chains. *Ieee transactions on automatic control* [Online], 23(2), pp.298–305. Available from: https://doi.org/10.1109/TAC.1978.1101707.

Gabillon, V., Ghavamzadeh, M. and Scherrer, B., 2013. Approximate dynamic programming finally performs well in the game of tetris [Online]. In: C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Weinberger, eds. *Advances in neural information processing systems*. Curran Associates, Inc., vol. 26. Available from: https://proceedings.neurips.cc/paper_files/paper/2013/file/7504adad8bb96320eb3afdd4df6e1f60-Paper.pdf.

Hengst, B., 2010. *Hierarchical reinforcement learning* [Online], Boston, MA: Springer US, pp.495–502. Available from: https://doi.org/10.1007/978-0-387-30164-8_363.

Kakade, S.M., 2001. A natural policy gradient [Online]. In: T. Dietterich, S. Becker and Z. Ghahramani, eds. *Advances in neural information processing systems*. MIT Press, vol. 14.

Available from: https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

Kamat, A. and Precup, D., 2020. Diversity-enriched option-critic. 2011.02565.

Klissarov, M. and Precup, D., 2021. Flexible option learning. 2112.03097.

Konidaris, G. and Barto, A., 2009. Skill discovery in continuous reinforcement learning domains using skill chaining [Online]. In: Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams and A. Culotta, eds. *Advances in neural information processing systems*. Curran Associates, Inc., vol. 22. Available from: https://proceedings.neurips.cc/paper_files/paper/2009/file/e0cf1f47118daebc5b16269099ad7347-Paper.pdf.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. 1312.5602.

S. Sutton, R., 1998. Between mdps and semi-mdps:learning, planning, and representing knowledge at multiple temporal scales [Online]. Available from: https://scholarworks.umass.edu/cgi/viewcontent.cgi?article=1212&context=cs_faculty_pubs.

Silver, D. and Ciosek, K., 2012. Compositional planning using optimal option models. 1206.6473.

Sutton, R., 1988. Learning to predict by the method of temporal differences. *Machine learning* [Online], 3, pp.9–44. Available from: https://doi.org/10.1007/BF00115009.

Sutton, R.S., McAllester, D., Singh, S. and Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation [Online]. In: S. Solla, T. Leen and K. Müller, eds. *Advances in neural information processing systems*. MIT Press, vol. 12. Available from: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Sutton, R.S., Precup, D. and Singh, S., 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* [Online], 112(1), pp.181–211. Available from: https://doi.org/https://doi.org/10.1016/S0004-3702(99)00052-1.

Szita, I. and Lörincz, A., 2006. Learning tetris using the noisy cross-entropy method. *Neural computation* [Online], 18(12), pp.2936–2941. Available from: https://doi.org/10.1162/neco.2006.18.12.2936.

Thiery, C. and Scherrer, B., 2009. Building controllers for tetris. *J. int. comput. games assoc.* [Online], 32(1), pp.3–11. Available from: http://dblp.uni-trier.de/db/journals/icga/icga32.html#ThieryS09.

Tsitsiklis, J. and Van Roy, B., 1995. Feature-based methods for large scale dynamic programming [Online]. *Proceedings of 1995 34th ieee conference on decision and control*. vol. 1, pp.565–567 vol.1. Available from: https://doi.org/10.1109/CDC.1995.478954.