Transfer Learning for Adaptive Navigation

Darcy Corson

1 Introduction

This project employs transfer learning to train a Deep Q-Network (DQN) agent to navigate through and avoid obstacles within novel grid-based environments. Initially, training focuses on simple scenarios to establish a foundational understanding of navigation and obstacle avoidance. As the agent gains proficiency, it is introduced to more complex environments. This progression tests the agent's ability to recall previous training as well as its capacity for real-time adaptation and strategic decision-making. The agent's real challenge begins when it encounters environments with a higher density of obstacles than it saw during training, significantly increasing the navigation task's complexity. Here, the agent must leverage its past learning to successfully navigate through these more complicated settings. This step is important as it mirrors real-world scenarios where adaptability and quick thinking are essential, particularly for autonomous systems facing unpredictable conditions.

The project culminates in an evaluation of the performance of the trained agent in these most complex environments. Also, a comparison of the trained agent's performance to that of an untrained agent is performed, thereby illustrating the effectiveness of the training and adaptation strategies. This not only underscores the potential of transfer learning in enhancing autonomous navigation, but also contributes to our understanding of spatial awareness and strategic problem-solving in AI.

2 Background

2.1 Transfer Learning

In the realm of reinforcement learning, transfer learning offers significant advantages in enhancing learning efficiency and performance. By leveraging knowledge acquired from one task, transfer learning enables an agent to apply learned expertise to a different, yet related, task, thereby reducing the time and resources required for learning from scratch. This approach is particularly beneficial in complex environments where the cost of exploration and learning can be prohibitively high.

The principal advantage of transfer learning lies in its ability to leverage pre-existing knowledge. This capability is most important in environments that

are characterized by vast state spaces or intricate dynamics, where the cost and time associated with exploring and learning each new scenario independently could be prohibitive. By employing transfer learning, an agent can bypass a substantial portion of this exploratory phase, drawing upon previously acquired insights and strategies. This not only speeds up the learning process but also significantly reduces computational and temporal resources.

Further, transfer learning aids in the development of more sophisticated, versatile agents. Agents that are trained with this approach are equipped to handle a broader range of tasks and environments. They exhibit an enhanced ability to quickly adapt and generalize their knowledge to new situations. For example, an agent trained in a static obstacle environment can apply its learned navigation strategies to a dynamic obstacle environment, adjusting and refining its tactics in real-time to accommodate evolving challenges.

Transfer learning in RL significantly enhances an agent's capability to develop strategies that extend beyond mere reactive responses. Instead, it equips them with predictive skills. This is largely due to the agent's exposure to a variety of environments during training. Diverse experiences enables the agent to recognize patterns and extrapolate them to unfamiliar situations, thus anticipating future states and adapting their strategies. The process of transfer learning accelerates the agent's learning curve, allowing it to quickly grasp the underlying dynamics of new environments. This rapid acquisition of knowledge is essential in scenarios where quick, informed decision-making is most important. Through this training method, agents become adept at not just responding to immediate stimuli but also strategically planning for future outcomes, a critical skill in dynamic environments.

2.2 Neural Networks

In RL, neural networks are computational models inspired by the human brain's structure and functionality. They consist of interconnected units called neurons or nodes, organized into layers: the input layer, which receives the initial data; hidden layers, where the data undergoes complex transformations; and the output layer, delivering the final processed output. Each neuron in these layers processes the incoming data, applying weights and biases, which are parameters that are adjusted during the training phase to optimize the network's performance. Activation functions within these neurons (e.g. ReLU, sigmoid) introduce non-linearity, enabling the network to capture and model intricate and non-linear relationships in the data. The learning process of neural networks involves iteratively adjusting these weights and biases to minimize the difference between the network's predictions and the actual outcomes, a process typically achieved through backpropagation and gradient descent algorithms.

2.3 Q-Learning

In RL, Q-learning is an RL algorithm used to enable agents to determine the most beneficial actions to take in various states without relying on a prede-

fined model of the environment. The crux of Q-learning is encapsulated in the Q-value or action-value function, represented as Q(s, a). This function is designed to estimate the cumulative reward an agent can expect by starting from a particular state (s) and taking a specific action (a). The Q-learning process begins with the initialization of these Q-values, typically set to zero for all state-action pairs. As the agent interacts with the environment (by taking actions, receiving rewards, and transitioning into new states) these experiences form the foundation for updating the Q-values. The core of Q-learning lies in its Q-value update rule, which adjusts the Q-values based on the immediate reward received and the estimated future rewards, governed by parameters such as the learning rate (α) and the discount factor (γ). The discount factor is particularly important as it balances the importance of immediate versus future rewards. A significant challenge in Q-learning is the exploration-exploitation dilemma, where the algorithm must find a balance between trying new actions (exploration) and sticking to known rewarding actions (exploitation). This is often managed using strategies like the epsilon-greedy approach, where the agent randomly explores with a certain probability (ϵ) and exploits the best-known action with the complementary probability (1-ϵ).

At its core, Q-learning is grounded in the Bellman Equation, a recursive relationship that defines the optimal policy. For each state-action pair (s, a), the Bellman Equation updates the Q-value $Q(s, a)$ based on the immediate reward plus the discounted maximum expected future rewards. The Bellman Equation for Q-learning can be expressed as:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') \right)$$

This process iteratively converges towards the optimal policy that maximizes the cumulative reward. In practice, this involves updating the Q-values using a learning rate α to blend the new and old values, guiding the agent to better decision-making over time. The term γ represents the discount factor, which determines the importance of future rewards.

2.4 Deep Q-Network Learning

Deep Q-Network (DQN) combines the principles of Q-learning, a traditional reinforcement learning algorithm, with the advanced capabilities of deep neural networks. In traditional Q-learning, the algorithm maintains a Q-table to store and update the value of each state-action pair. However, this becomes infeasible in environments with a vast number of possible states and actions. DQN addresses this challenge by using deep neural networks to approximate the Q-value function, thereby predicting the potential rewards for each action given a specific state. This approach eliminates the need for a Q-table, enabling scalability and efficiency.

An important feature of DQN is the incorporation of the experience replay mechanism. The algorithm stores the agent's experiences in a replay buffer and randomly samples mini-batches from this buffer for training the network. This

strategy significantly enhances learning stability and prevents the network from overfitting to recent experiences. Another important aspect of DQN is the use of a target network, which is a duplicate of the primary network that is updated less frequently. This setup provides stable target values for the Q-function, further stabilizing the training process.

DQN's integration of neural networks allows it to efficiently handle high-dimensional input spaces, making it ideal for complex environments. The generalization capabilities of neural networks within DQN enable the agent to perform effectively in various scenarios, including those not encountered during training.

2.5 Replay Buffers in Deep-Q Networks

In DQNs, the use of a replay buffer, also known as experience replay, can improve the learning process's stability and efficiency. This type of buffer acts as a dynamic memory bank, storing a finite history of agent experiences, typically represented as tuples $(s_t, a_t, r_{t+1}, s_{t+1}, \text{done})$, where s_t is the state at time t, a_t is the action taken at time t, r_{t+1} is the reward received after taking action a_t, s_{t+1} is the next state the agent transitions to, and 'done' indicates whether or not the episode has concluded.

The incorporation of a replay buffer in DQN aligns with and exploits the off-policy nature of Q-learning, allowing the agent to learn more effectively from a broad range of interactions with the environment. This is true for several reasons. First, because sequential learning is often plagued by highly correlated experiences, which lead to inefficiencies and a tendency toward overfitting, agents could repetitively execute a limited array of actions without sufficient exploration. The replay buffer addresses this issue by storing a wide variety of past experiences and enabling the agent to learn from a randomized selection of them. This approach effectively reduces the temporal correlation among sequential experiences, thereby diversifying the data distribution from which the agent learns. Second, the capacity of the replay buffer to retain experiences allows for the reuse of each interaction. This magnifies the value derived from each environmental encounter. The update formula for the Q-value in experience replay, using a sampled mini-batch of experiences B from the buffer, is expressed as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \forall (s,a,r,s') \in B$$

Here, α represents the learning rate, and γ is the discount factor. Finally, by sampling from a varied array of experiences, the replay buffer ensures that learning isn't overly focused on recent trends as this could be misleading or non-representative.

Both the size and management of the buffer, as well as the sampling strategy, are important things to consider when implementing a replay buffer. The capacity of the buffer significantly impacts the quantity of experiences that can be stored. That is, a larger buffer is able to hold more data, but also requires

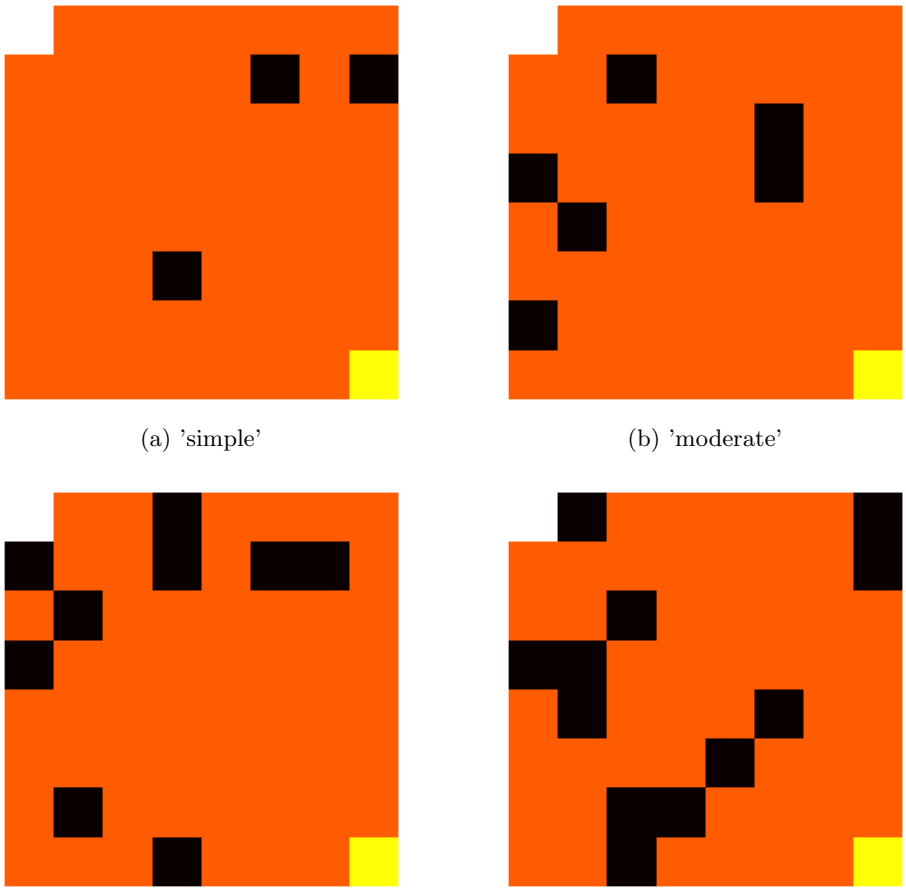
more memory and computational resources. Typically, older experiences are removed on a first-in-first-out basis as the buffer reaches capacity. While uniform random sampling is common, more sophisticated methods like prioritized experience replay can refine learning by focusing on more rare or unexpected experiences.

3 Experiment

3.1 Overview

In this experiment, an agent is trained to navigate an 8x8 grid with increasing obstacle density using a Deep Q-Network (DQN) enhanced with a replay buffer. The replay buffer's role is important, as it provides a diverse range of experiences from past actions and outcomes, significantly improving learning stability and breaking correlations in sequential data. The agent's performance is developed across different levels of environment complexity, guided by a detailed state representation that includes its position, the goal's location, and immediate surroundings. As the agent progresses, it receives rewards for reaching the goal and penalties for collisions with obstacles or going out of bounds, incentivizing efficient navigation strategies with no collisions. After training is complete, a testing phase begins where the performance of the trained agent is evaluated in an environment with a higher density of obstacles than seen during training. This environment is designed to test the limits of the agent's adaptability and learning. The trained agent's performance is then compared to that of an untrained agent in the same obstacle-dense environment, illustrating the effectiveness of the DQN with replay buffer training. This comparison is aimed at showcasing the trained agent's enhanced navigational capabilities and learning efficiency, underscoring the DQN's ability to equip agents with the skills necessary to tackle complex and unfamiliar spatial challenges.

3.2 Grid Environment



(a) 'simple' (b) 'moderate'

(c) 'complex' (d) 'very_complex'

Figure 1: The image presents four grid environments of varying levels of complexity. For each, the white square (0,0) represents the agent's starting location, the yellow square (7,7) represents where the agent's is being trained to navigate to, and the black squares represent obstacles.

The agent is trained and/or tested in a custom 8x8 grid-based environment, known as 'CustomGridEnvironment'. This environment presents a square field of 64 cells and introduces four levels of complexity ('simple', 'moderate', 'complex', 'very_complex'), dictating the density of obstacles in that field. Specifically, the 'simple' environment is constructed with 5% of the grid's area as obstacles, the 'moderate' environment with 10%, and the 'complex' environment with 15%. For testing, a 'very_complex' environment is constructed with 20% obstacles.

During training, each episode affords the agent a maximum number of 200 steps to complete its task, allowing for extensive exploration and learning from the environment before an episode is terminated. This higher step limit is crucial during training because it gives the agent enough opportunity to explore the state space thoroughly, make mistakes, and learn from them. This is essential for the agent to understand the consequences of its actions and to find and refine strategies for navigation. During testing, however, each episode affords the agent a maximum number of 32 steps to complete its task. This constraint allows for a more focused assessment of the agent's efficiency and decision-making skills. That is, by limiting the number of steps during testing, the agent is forced to use its policy to navigate to the goal in a shorter number of steps, demonstrating its ability to perform under tighter constraints and in a more efficient manner.

The agent begins each episode in the top-left corner (0,0), aiming to reach the bottom-right corner (7,7) of the environment, navigating around obstacles that are strategically placed based on the environment's complexity, as described above. A Breadth-First Search algorithm ensures at least one navigable path from the start to the goal for each environment presented to the agent. The agent's state is comprehensively represented as a 2D array marking free spaces and obstacles. A parallel discovered grid acts as the agent's memory, noting encountered obstacles. The agent's and goal's positions are encoded within the grid, providing a clear navigational aim. Surrounding cell information, the distance, and direction to the goal, along with a binary visibility indicator, provide comprehensive navigational data.

3.3 DQN Agent and Its Environmental Interaction

The agent in this experiment is initialized with a unique seed for the sake of reproducibility, and interacts with the environment via a DQN architecture. The DQN, instantiated in the 'QNetwork' class, consists of several layers with ReLU activation, ensuring non-linear function approximation. It takes the state of the environment as input and outputs a vector of action values.

As discussed above, the state includes a flat representation of the grid, the agent's current position, the goal's location, the distance and direction to the goal, and the immediate surroundings. This information allows the agent to make informed decisions about its movements. The 'get_flat_state()' method compiles this information into a comprehensive state vector that feeds into the DQN.

During training, the agent's interaction with the environment is facilitated by the 'step()' method. As the agent selects actions ('up', 'down', 'left', 'right'), it navigates through the grid environment, trying to reach the goal at (7,7). Each action that the agents takes results in feedback from the environment: the agent receives a new state reflecting the consequences of its action, and a reward signal that evaluates the action's effectiveness. This reward is designed to guide the agent towards efficient navigation. As such, positive rewards encourage the agent to move closer to the goal, while negative rewards discourage inefficient actions such as hitting obstacles or venturing out of bounds. Notably, step-

ping out of bounds not only incurs a penalty but also terminates the episode, reinforcing the negative consequences of such action.

Central to the agent's learning mechanism is the 'learn()' function. This function facilitates the Deep Q-Network (DQN) algorithm's update of the agent's knowledge, represented as Q-values. These Q-values are estimates of the expected rewards for taking certain actions in particular states. During the learning process, the agent uses experiences drawn randomly from the replay buffer to update these Q-values. This random sampling helps to break the correlations in the sequential observations the agent makes, providing a more robust and diverse set of experiences for the agent to learn from. Such diversity encourages the agent to develop a generalizable and resilient policy.

The 'ReplayBuffer' class supports this sophisticated learning mechanism. It functions as the agent's memory bank, storing a finite number of the most recent experiences, each comprised of the state, action, reward, next state, and a done flag indicating whether the episode has ended. By maintaining this buffer, and sampling from it randomly, the agent is able to recall – and learn from – past experiences. This method of learning mimics human learning, where reflection on past experiences can inform and improve future decisions. The replay buffer effectively allows the agent to learn from the past without being tethered to the immediate sequential flow of experiences, thus avoiding the pitfalls of short-term, often myopic, learning.

A target Q-network, part of the DQN architecture, helps to stabilize training by generating stable target values for the Temporal Difference (TD) error calculation. It operates as a secondary, more stable version of the agent's primary Q-network, and its weights are updated less frequently. This is done in order to prevent rapid oscillations and instability in the learning process. The 'soft_update()' method updates the target network's weights gradually by blending them with the local network's weights. A parameter, τ, dictates the degree to which the target network's weights should shift toward the local network's weights. A smaller τ value results in more gradual changes to the target network, providing a stable and more slowly adjusting reference for calculating the TD error and updating the local network's weights. This contributes to making the learning process more stable and effective. The update is mathematically represented as:

$$\theta_{\text{target}} = \tau \cdot \theta_{\text{local}} + (1 - \tau) \cdot \theta_{\text{target}},$$

where θ_{target} and θ_{local} are the weights of the target and local networks, respectively.

3.4 Training Phase

In the training phase of the experiment, the agent attempts to learn to navigate from the starting point (0,0) to the goal (7,7) while avoiding collisions with obstacles. As discussed above, the training environment's complexity level determines the number of obstacles present ('simple', 'moderate', or 'complex').

Two neural networks with the same architecture, the Q-Network and the target Q-Network, are initialized to predict Q-values for state-action pairs. A replay buffer is also established to store, and later recall, experiences.

The learning process unfolds over 100,000 training episodes, each allowing for a maximum of 200 steps, which provide the agent with ample opportunity to interact with and learn from each environment that it encounters. During each episode, the agent makes decisions based on an epsilon-greedy policy. This policy dictates that, at the beginning, the agent primarily explores the environment randomly to understand the consequences of various actions. However, as training progresses, epsilon decays from its initial value of 1.0 down to 0.01. Thus, the agent increasingly relies on the learned Q-values to make informed decisions, thereby shifting its approach from exploration to exploitation.

The environment responds to each action that the agent takes by updating the agent's position, calculating a reward based on the agent's proximity to the goal and interaction with obstacles, and determining if the episode has concluded. After an action, information about it and its consequences is stored in the replay buffer. Periodically, the agent samples from this buffer to learn from a diverse set of experiences. It uses these samples to update the local Q-Network's weights by minimizing the difference between predicted and target Q-values, the latter calculated using the more stable target Q-Network. To maintain stability in learning, the weights of the local Q-Network are softly copied to the target Q-Network at intervals, ensuring that the target values change gradually.

Another important aspect of the agent's training regime is the implementation of curriculum learning. That is, as the agent's performance in an environment of given complexity improves, the relative complexity of the environment escalates from 'simple' to 'moderate' to 'complex'. This gradual increase in difficulty ensures that the agent encounters and potentially learns to navigate increasingly challenging scenarios. The transition between complexities is based on the agent's performance, specifically the average score over the last 1,000 episodes. If the agent consistently achieves a predefined score threshold of 60, the environment's complexity level is upgraded. This approach is akin to providing a curriculum that adapts to the agent's learning pace. Initially, in more simple environments, the agent learns basic navigation and avoidance strategies. As the complexity increases, the agent refines these strategies and develops new ones to deal with the added difficulty. This progressive learning attempts to ensure that the agent isn't overwhelmed early on and can build on its knowledge incrementally.

It is important to note that, for the 100,000 training episodes, each episode presents the agent with a unique configuration of obstacles within the environment at whatever complexity level the agent is training at. Because, at a given complexity level, the number and placement of obstacles vary randomly from one episode to the next, even within the same complexity level, the agent encounters a new challenge in every episode. The rationale behind this approach to is multifaceted. First, by continuously changing the obstacle configuration, the agent is prevented from overfitting (memorizing) a specific layout. Overfitting

would result in an agent that performs exceptionally well on seen configurations but fails to generalize its strategy to new scenarios. The obstacle-varied layouts encourage the agent learns robust strategies that are effective across a wide range of situations. Further, each new configuration is an opportunity for the agent to learn something new or refine its existing strategy. This exposure to a wide variety of scenarios encourages the development of a more flexible and generalizable policy. In addition, the changing environment naturally encourages exploration. Since the agent cannot predict the next environment's layout, it must explore to understand how to navigate the new configuration successfully.

3.5 Testing Phase

During the experiment's testing phase, a total of 1000 trials are executed to evaluate the trained and untrained agents' ability to navigate a 'very_complex' grid environment. This environment is reinitialized for each trial, providing a unique and consistently challenging simulation space with a novel obstacle configuration. The trained agent, leveraging the learning accomplished during the training phase, navigates each of these 'very_complex' environments with a deterministic policy, taking no more than 32 steps per trial – a constraint that ensures each trial is bounded in time and complexity. This strategy showcases pure exploitation, as the agent relies entirely on its trained knowledge, without incorporating randomness into its decision-making process. In contrast, the untrained agent's approach is characterized by randomness, with its actions determined entirely by exploration (epsilon = 1.0), providing a baseline for performance comparison.

Each agent's interaction with the environment is recorded, capturing the total score achieved within the 32-step limit, the number of collisions encountered, and the specific path taken. Trained and untrained agents are tested on identical sets of 1000 environments.

Upon the completion of the 1000 trials, the average score and standard deviation for both agents across all iterations is calculated, offering a quantitative measure of their navigational proficiency and the consistency of their performance. This process of evaluation, enhanced by the aggregation of performance metrics and the visual depiction of navigational patterns, yields a nuanced assessment of the trained agent's ability to apply its learned behaviors effectively against the stochastic actions of the untrained agent.

3.6 Results

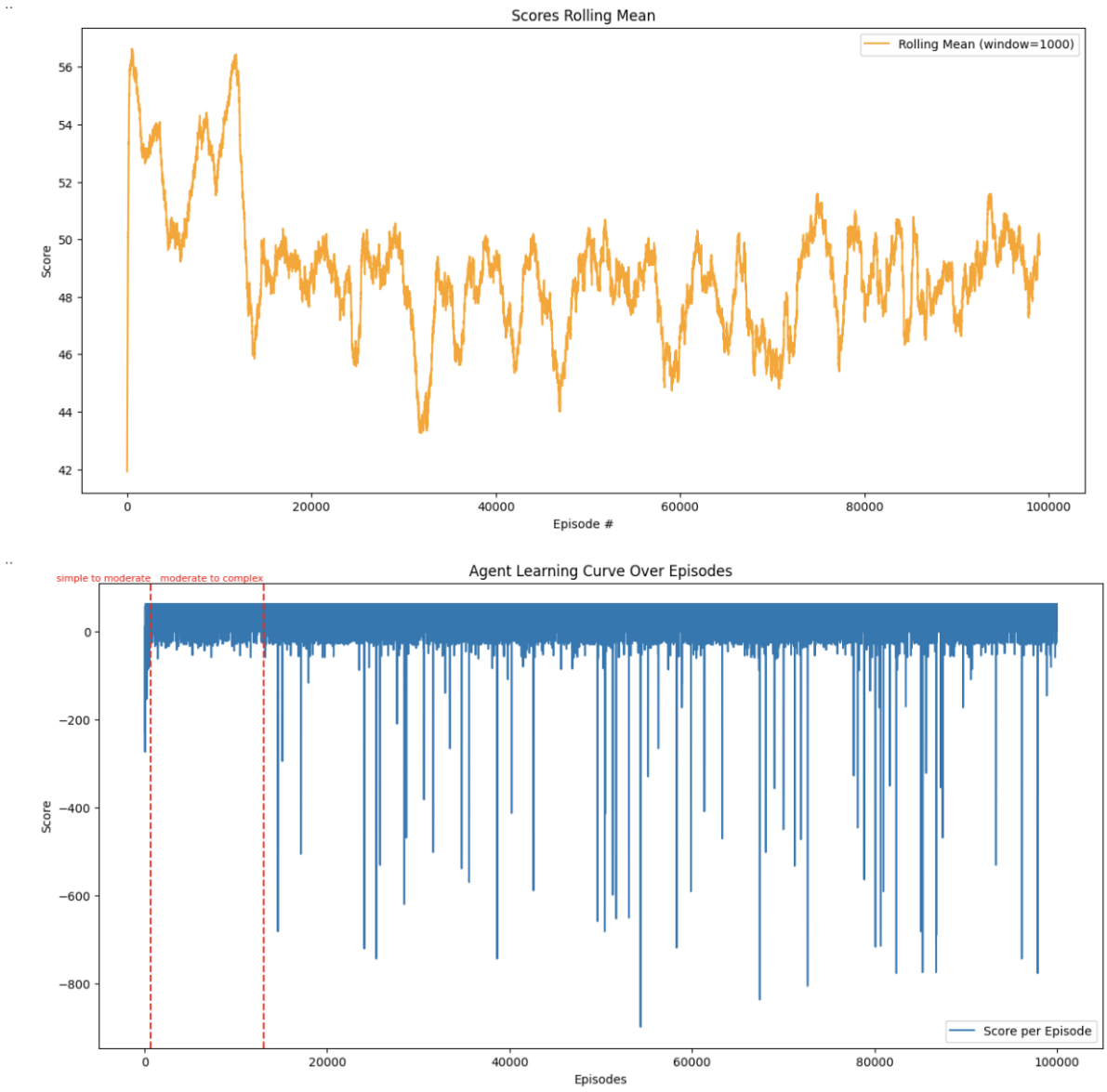


Figure 2: Agent performance over training episodes, highlighting the transitions between environment complexity levels. The points at which the complexity level increases are marked, showing the agent's adaptation to more challenging scenarios.

Figure 2 shows the DQN agent's score trends as it interacts with a progressively more complex environment during the training phase of the experiment. The top graph (Scores Rolling Mean) represents the rolling mean of scores achieved by the DQN agent over 100,000 training episodes, with the mean calculated over a window of 1,000 training episodes. The bottom graph shows the raw scores for the DQN agent over 100,000 training episodes. The vertical dashed lines represent the points during the training phase when the complexity of the environment being used for training increases.

Initially, the scores show a great degree of fluctuation, reflecting the agent's attempts to explore and learn from diverse scenarios within the 'simple' obstacle configuration environment. As the agent's experience accumulates, one might expect to see a trend where the average score gradually improves, signifying that the agent is starting to generalize and apply successful strategies to achieve its goals more consistently. However, this may not be the case for this experiment as the highest score achievable in the environments of different complexities may not be the same.

Upon transitioning to a more complex environment, scores may temporarily dip or exhibit increased volatility. This is a natural consequence of the agent encountering and learning to navigate new challenges that weren't present in the more simple environment. However, if the scores begin to rise again and maintain an upward trajectory, even amidst fluctuations, it indicates that the agent is adapting and learning from the complex environment.

The rolling mean serves as a smoother representation of this trend, dampening the impact of short-term fluctuations and highlighting the long-term progression. An increasing rolling mean, especially after the introduction of complexity, suggests that the agent is indeed improving over time. That is, it's learning not only to cope with the new challenges but also to apply its accumulated knowledge more effectively.

It's important to consider that while the overall trend might be upward, the path to improvement is rarely linear, especially in a learning context with many variables. It is expected that the agent's performance improvement will be punctuated by peaks and troughs as it encounters and overcomes new obstacles. An upward trend in the rolling mean, amidst these fluctuations, is an indication that the agent is on a positive learning curve.

Once the DQN agent begins training on the 'complex' environment, its scores seem to fluctuate within a certain range without a very clear increasing or decreasing trend over the long term, although it does look like they become slightly higher and less volatile as the number of training episodes completed approaches 100000. This could imply that the agent has reached a plateau in its learning or that the increasing complexity of the environment challenges the agent, preventing consistent improvement.

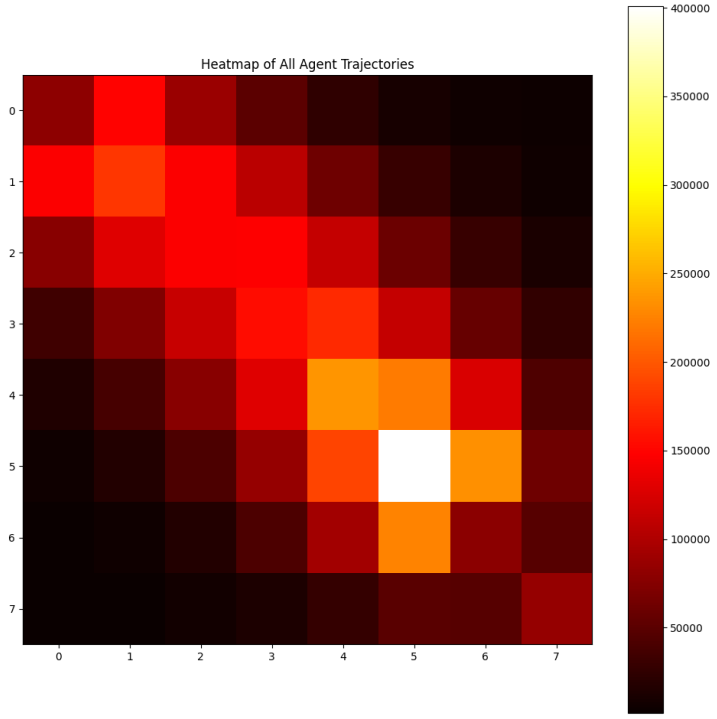
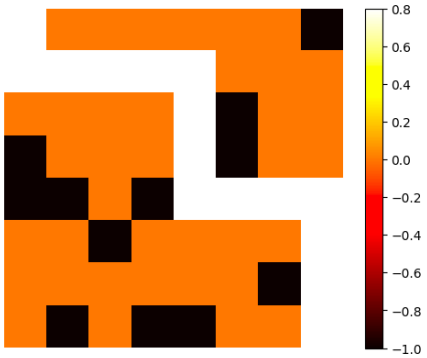
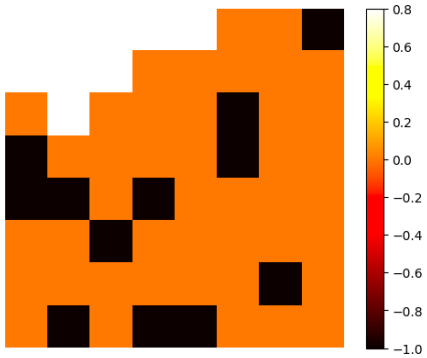


Figure 3: This heatmap depicts the frequency of the agent's positions throughout the training phase of the experiment. Brigther areas signify higher visitation frequencies, indicating the agent's learned preference for certain paths. The concentration of brighter colors along the diagonal trajectory from (0,0) to (7,7) suggests that the agent is learning to navigate efficiently towards the goal, reflecting the progressive optimization of its pathfinding strategy over time

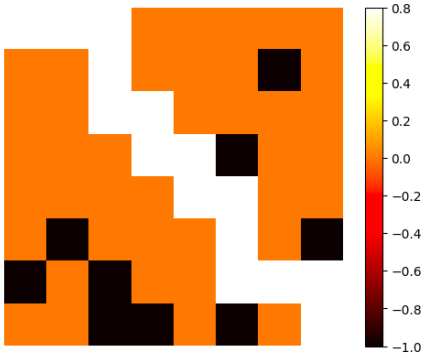
Figure 3, also produced during the training phase of the experiment, serves to help visualize the agent's learning progress. The brighter areas indicate paths that the agent frequented more often, suggesting these routes are likely to be the most efficient paths discovered by the agent to reach the goal. The presence of such patterns is indicative of successful learning, as it shows the agent is not just wandering randomly but is instead making consistent decisions that lead towards the goal.



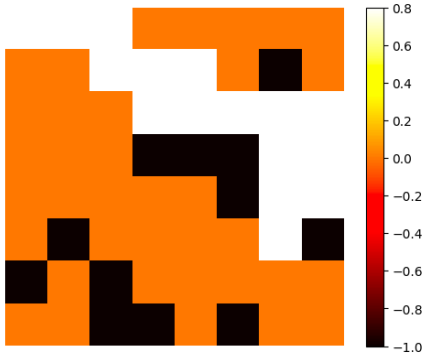
(a) Test Iteration 250 - Trained Agent



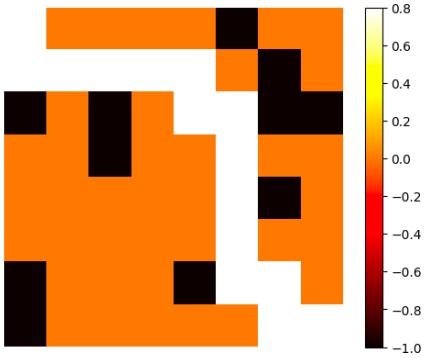
(b) Test Iteration 250 - Untrained Agent



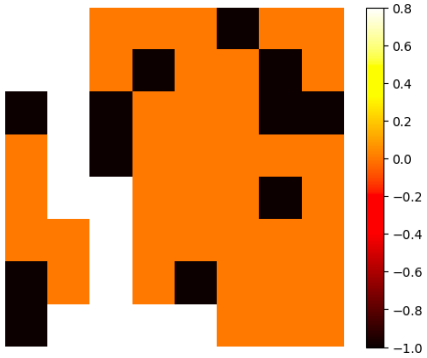
(c) Test Iteration 750 - Trained Agent



(d) Test Iteration 750 - Untrained Agent



(e) Test Iteration 1000 - Trained Agent



(f) Test Iteration 1000 - Untrained Agent

Figure 4: The figure presents a comparative visualization of the performance between the trained and untrained agents at various test iterations within the grid-based environment. Panels (a), (c), and (e) depict the progression of the trained agent at test iterations 250, 750, and 1000, respectively, showcasing its path-finding ability in navigating towards a goal while avoiding obstacles. Panels (b), (d), and (f), in contrast, illustrate the corresponding performance of an untrained agent at the same test iterations. For all, white represents the path traversed by the agent.

Figure 4 illustrates the behavioral contrast between the trained and untrained agent during the test phase of the experiment, where no additional learning occurs. The trained agent's paths demonstrate the effectiveness of its learned policy, developed during prior training episodes on less complex environments. That is, they reveal an ability to navigate towards the goal with consideration of the environment's structure and obstacles. Even without further learning, the trained agent exhibits strategic behavior, likely taking the most direct route to the goal and circumventing obstacles, reflecting the cumulative knowledge acquired during training.

Conversely, the untrained agent's paths, driven purely by exploration, lack purposeful navigation and display random movements. Since the untrained agent has not undergone learning, it does not benefit from prior experience and shows no improvement in its behavior across test iterations. Its movements are haphazard and indicate no understanding of the environment, goal location, or obstacle avoidance.

Comparative analysis of the trained and untrained agents' performance in the more complex test environment generates significant insights. The adeptness of the trained agent in navigating through the new, more challenging environments suggests that the agent has not only learned specific solutions to the training scenarios but has also developed a generalizable set of skills. This adaptability is indicative of transfer learning, where knowledge acquired during training is successfully applied to novel situations. The trained agent's ability to generalize and effectively use its learned policy to tackle increased complexity highlights the robustness of its decision-making processes. It shows that the agent has learned abstract representations or strategies that transcend the specifics of the training environment, allowing it to adapt and perform well even when faced with scenarios that it has not explicitly encountered during training.

On the other hand, the contrast presented by the untrained agent's behavior in the same environments, with its lack of systematic exploration or strategy, underscores the necessity and impact of the learning phase. The untrained agent's performance remains stagnant, with no clear progression or learning, which aligns with expectations as it operates without the benefit of accumulated experience.

Trained Agent Success Percentage: 97.70%
Untrained Agent Success Percentage: 1.10%
Trained Agent Collision Percentage: 0.00%
Untrained Agent Collision Percentage: 92.40%

The performance metrics above reveal a significant disparity between the performances of the trained and untrained agents during the testing phase, again highlighting the significant impact of training on agent behavior and efficacy. The trained agent boasts an impressive 97.70% success rate, indicating that is able to navigate from (0,0) to (7,7) through the novel, 'very_complex', obstacle-heavy test environments with near perfect reliability. This high success rate is a testament to the effectiveness of the training process, which has evidently

equipped the agent with robust skills and strategies necessary to navigate and perform in novel environments successfully. Further, the fact that the trained agent has a 0.00% collision rate underscores its precision and safety; it not only frequently achieves its goal but does so without colliding with obstacles.

In contrast, the untrained agent demonstrates a success rate of only 1.10%, suggesting that it is almost completely unable to navigate from (0,0) to (7,7) through the novel, 'very_complex', obstacle-heavy test environments. This lack of success points to a deficiency in necessary skills or decision-making capabilities that training might provide. Further, the untrained agent's high collision rate of 92.40% is indicative of its propensity for making significant errors.

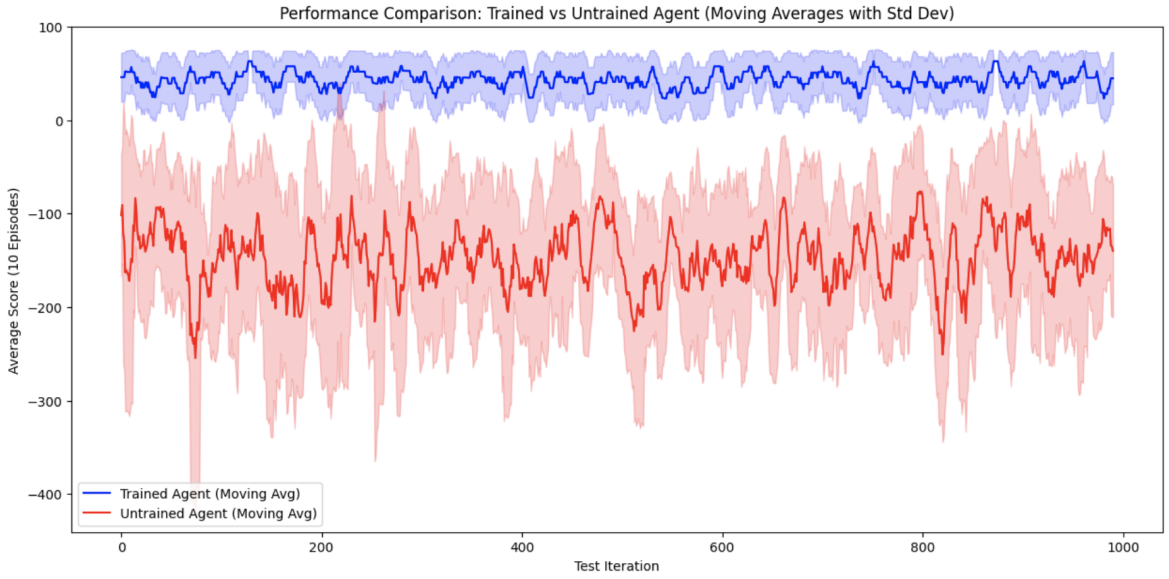


Figure 5: The graph presents a performance comparison of the trained agent against the untrained agent across the 1000 test iterations, showcasing the value of learning. The trained agent demonstrates a consistent, higher score with minimal standard deviation, indicating the reliability of its acquired knowledge. In contrast, the untrained agent displays highly variable performance with a significant standard deviation, underscoring the unpredictability and inefficacy of an exploration approach without prior learning.

The graph in Figure 5 presents a comparison of performance between the trained and the untrained agents over the series of 1000 test iterations. The horizontal axis denotes the test iterations, while the vertical axis represents the average score per 10 episodes, indicating the effectiveness of the agents' strategies in achieving their goal within the environment.

The blue line, representing the trained agent, maintains a relatively high and stable score with a smaller standard deviation shaded in lighter blue. This

stability suggests that the trained agent consistently applies a learned policy from its training phase, effectively navigating the environment and reaching its goal. The higher scores imply that the agent not only learned to perform well in the training environment but also successfully transferred this knowledge to the more complex test environment, a hallmark of successful transfer learning. The agent's ability to maintain performance when faced with increased complexity indicates that it has developed a robust strategy that generalizes beyond the conditions it was trained on.

In contrast, the red line depicts the untrained agent, which exhibits a lower average score with significant volatility, as evidenced by the broader red shading. The large standard deviation and the overall trend of the graph suggest that the untrained agent, which relies solely on exploration without prior learning, struggles to find successful strategies. The erratic nature of its performance indicates a lack of understanding and adaptation to the test environments' complexities.

The significance of this graph lies in its clear depiction of the benefits of training and the concept of transfer learning. The trained agent's superior performance in an unfamiliar test environment demonstrates that learning during the training phase enables the agent to adapt to new challenges effectively. It shows that the agent has not only memorized specific paths or strategies but has also developed an underlying understanding of how to approach goals, which it can apply to different contexts. Further, the graph validates the importance of the training phase, where the agent's experiences lead to the development of a nuanced policy that generalizes to new situations. This is especially pertinent in applications where agents must operate in dynamic environments, emphasizing the need for robust learning mechanisms that enable transfer learning. The contrast between the trained and untrained agents underscores the impact of learning on agent behavior.

CONFIDENCE INTERVALS FOR MEAN SCORES OF AGENTS NAVIGATING TEST ENVIRONMENTS

Average Score for Trained Agent: 43.07 ± 27.29
Average Score for Untrained Agent: -147.28 ± 95.58

The confidence intervals for the mean scores of trained and untrained agents on test environments provide additional insight into the relative performance and reliability of these agents. For the trained agent, the average score is 43.07 with a confidence interval of ±27.29. This means that we can be reasonably certain that the true average score of the trained agent lies somewhere between 15.78 and 70.36. The wide range of this interval reflects a significant variance in the scores, indicating that while the trained agent generally performs well, its performance can fluctuate depending on the test environment or other factors.

In contrast, the untrained agent has an average score of -147.28 with a much wider confidence interval of ±95.58. This suggests that the true average score is likely between -242.86 and -51.70. The negative scores indicate poor performance, and the wide interval signifies a high degree of uncertainty and inconsistency in its performance. Such a broad range also implies that the

untrained agent's behavior is highly unpredictable, with its performance varying dramatically across different test environments.

The significance of these confidence intervals lies in their ability to quantify the uncertainty and reliability of the agents' performance. For instance, the narrower confidence interval of the trained agent, despite its relative width, suggests a more consistent performance than the untrained agent. Thus, this result highlights the transfer learning on agents' ability to achieve more predictable and reliable outcomes.

The difference in performance between trained and untrained agents is statistically significant (p-value = 0.0000).

The statistical results indicating a significant difference in performance between trained and untrained agents also have important implications. The p-value, reported as 0.0000, is a measure of the probability that the observed difference in performance could have occurred by chance if there were actually no difference between the two groups. In this context, a p-value of 0.0000 is extremely low, indicating that the probability is less than 1 in 10,000. This is far below the commonly used significance threshold of 0.05, beyond which results are generally considered statistically significant. In practical terms, such a low p-value strongly suggests that the difference in performance between the trained and untrained agents is real and highly unlikely to be due to random chance.

t-statistic: 60.5280

The t-statistic of 60.5280 reinforces this conclusion. The t-statistic measures the size of the difference between groups relative to the variation in the data. A higher t-statistic indicates a larger difference between the groups. In this case, a t-statistic of 60.5280 is exceptionally high, suggesting a very large difference in performance between trained and untrained agents. A high t-statistic in combination with a low p-value confirms that the observed difference is not only statistically significant but also practically meaningful.

The significance of these results lies in their implications for understanding and improving agent performance. These statistically and practically significant differences underscore the effectiveness of training in enhancing the performance of an agent navigating an environment with obstacles. This conclusion can be drawn because the statistical evidence strongly indicates that the trained agent outperforms the untrained agent to a significant degree in this experiment.

4 Extensions

To broaden this experiment, one could explore numerous straightforward strategies to enhance both our understanding and the application of the findings presented here. This includes incorporating a focus on more effective curriculum learning schemas and investigating the relative rates of learning between trained

and untrained agents.

First, we could adjust the training approaches, such as the duration, intensity, and specific methods used, including various curriculum learning schemas. By experimenting with different curriculums, we may be able to identify a sequence and complexity of tasks that most effectively enhance the agents' learning. Further, it would also be interesting to diversify the range of test tasks, presented to the trained agent during the testing phase, in order to understand on what types of tasks the trained agent excels and/or still need improvement. It would be interesting to develop a sense of the reach and limits of the trained agent's training's helpfulness.

It would also be interesting to monitor and compare the learning rates of trained and untrained agents in the test environment. Specifically, I'd like to measure how quickly each type of agent improves and how long it takes for an initially untrained agent to catch up to the performance level of a trained one, if at all. This involves tracking their progress over time and identifying at what point, if ever, the untrained agents reach similar performance levels as their trained counterparts.

5 References

Chevalier-Boisvert, Maxime, et al. "Minigrid Miniworld: Modular Customizable Reinforcement Learning Environments for Goal-Oriented Tasks." arXiv, arXiv:2306.13831, 24 June 2023, arxiv.org/abs/2306.13831. Accessed November 2023.

Gros, Timo P. "Tracking the Race: Analyzing Racetrack Agents Trained with Imitation Learning and Deep Reinforcement Learning." Master's thesis, Universität des Saarlandes, 2021, https://mosi.uni-saarland.de/assets/theses/matimo.pdf.

He, George, et al. "Improving DQN Training Routines with Transfer Learning." George He's Research, 2021, https://georgehe.me/dqntransfer.pdf.

Lee, Min-Fan Ricky, and Sharfiden Hassen Yusuf. 2022. "Mobile Robot Navigation Using Deep Reinforcement Learning" Processes 10, no. 12: 2748. https://doi.org/10.3390/pr10122748

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. 2016. "Prioritized Experience Replay." arXiv preprint arXiv:1511.05952.

Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. The MIT Press, 2018.