Executive Summary

This report investigates reinforcement learning and sequence learning techniques, addressing theoretical and empirical aspects of both domains. In **Part A: Reinforcement Learning**, we explore Deep Q-Learning (DQN) and Double Q-Learning, focusing on their mechanisms and limitations, particularly DQN's overestimation bias (Van Hasselt, 2010). An empirical evaluation using the Taxi-v3 environment revealed identical performance for both algorithms due to insufficient training steps, underscoring the need for extended training in reinforcement learning tasks (Sutton & Barto, 2018). **Part B: Sequence Learning** examines the vanishing gradient problem in sequence models, contrasting solutions like LSTMs and Transformers (Hochreiter & Schmidhuber, 1997; Vaswani et al., 2017). An empirical comparison using the tweet_emotions dataset demonstrates LSTMs' superior performance over RNNs in sentiment classification, highlighting their effectiveness in capturing long-term dependencies (Graves, 2013). The findings emphasize the importance of advanced architectures in sequence learning and adequate training in reinforcement learning for optimal performance.

Part A: Reinforcement Learning

1. Introduction

Reinforcement Learning (RL) enables agents to learn optimal actions through interaction with environments, with Deep Q-Learning (DQN) and Double Q-Learning as pivotal advancements for complex tasks (Mnih et al., 2015; Van Hasselt, 2010). DQN leverages neural networks to approximate Q-values, addressing high-dimensional state spaces, yet suffers from overestimation bias. Double Q-Learning mitigates this by decoupling action selection and evaluation, enhancing stability (Van Hasselt, 2010). This section explores the theoretical foundations of these algorithms, their limitations, and an empirical evaluation using the Taxi-v3 environment, aiming to highlight their performance differences and practical implications for RL applications.

2. Theoretical Explanation of DQN and Double Q-Learning

Deep Q-Learning (DQN), introduced by Mnih et al. (2015), enhances Q-Learning by using deep neural networks to approximate the Q-function, enabling agents to tackle high-dimensional state spaces like Atari games. However, DQN's max operator often causes overestimation bias, impairing performance (Van Hasselt, 2010). Double Q-Learning, proposed by Van Hasselt (2010), counters this by employing two independent estimators to separate action selection and evaluation, reducing bias and improving stability. This theoretical comparison highlights the strengths and challenges of these algorithms, laying the theoretical groundwork for exploring these algorithms' mechanisms and limitations.

2.1 Deep Q-Learning: Overview and Limitations

Q-Learning is a key reinforcement learning method that seeks to determine the optimal action-value function, $Q*(s,a)$, representing the expected cumulative reward for taking action a in state s and following the best policy thereafter (Watkins & Dayan, 1992). While traditional Q-Learning relies on a tabular format, it struggles with large or continuous state spaces. Deep Q-Learning (DQN), introduced by Mnih et al. (2015), overcomes this by using deep neural networks to approximate the Q-function, allowing it to process complex inputs like raw pixels from Atari games.

However, DQN has a notable flaw: overestimation bias. This stems from the max operator in its update rule: $Q(s,a) \leftarrow r+\gamma \max_{a'} Q(s', a')$. The operator selects the highest estimated Q-value, which can exaggerate values due to noise or inaccuracies, leading to systematic overoptimism (Van Hasselt, 2010). In environments with stochastic rewards, such as Atari games like *Enduro* and *Seaquest*, this bias has been shown to degrade performance (Van Hasselt et al., 2016).

Theoretically, overestimation occurs because the same Q-network is used for both selecting and evaluating actions, compounding estimation errors (Thrun & Schwartz, 1993). This can slow convergence or even cause divergence, impacting the quality of the learned policy. These limitations spurred the development of Double Q-Learning, which separates action selection and evaluation to mitigate the bias (Van Hasselt, 2010).

2.2 Double Q-Learning: Motivation and Approach

Double Q-Learning was developed to address the overestimation bias inherent in standard Q-Learning. In Q-Learning, the update rule uses the maximum estimated Q-value for the next state, which can lead to overoptimistic value estimates due to noise in approximations (Van Hasselt, 2010). This bias becomes particularly pronounced in stochastic environments or when using function approximators like neural networks. Such overestimation can disrupt the learning process, resulting in suboptimal policies and reduced performance.

To mitigate this issue, Double Q-Learning introduces two independent Q-functions. One Q-function determines the action that maximizes the Q-value in the next state, while the other evaluates that action's value. This decoupling of action selection and value estimation reduces bias by preventing the propagation of overestimated values. For

example, with two Q-functions, QA and QB, updating QA involves selecting the action a' = argmax a' $QA(s', a')$, then using $QB(s', a')$ to estimate its value. This separation ensures that noise in one Q-function does not unduly influence the other.

The approach was first proposed by Hado van Hasselt (2010) for tabular reinforcement learning, where it demonstrated reduced overestimation and improved stability. In deep reinforcement learning, Double Q-Learning extends to Double Deep Q-Networks (DDQN), employing two neural networks to handle complex, high-dimensional state spaces. By addressing overestimation, Double Q-Learning often achieves faster convergence and superior performance, particularly in environments requiring precise value estimation.

3. Empirical Evaluation of Deep Q-Learning and Double Q-Learning

This section presents an empirical evaluation of Deep Q-Learning (DQN) and Double Q-Learning, comparing their performance in the Taxi-v3 environment from OpenAI Gym (Brockman et al., 2016). Both algorithms were implemented using the DQN Agent in keras-rl2, with training constrained to 6,000 steps due to runtime limits. The evaluation examines the experimental setup, performance results, and practical implications of their theoretical differences. Detailed findings and analysis follow in the subsections.

3.1 Experimental Setup

The empirical evaluation was conducted using the Taxi-v3 environment from OpenAI Gym (Brockman et al., 2016). Taxi-v3 is a discrete reinforcement learning task where an agent navigates a 5x5 grid to pick up and drop off passengers, featuring 500 possible states and 6 actions (Gymnasium, 2025). Both Deep Q-Learning (DQN) and Double Q-Learning were implemented using the DQN Agent from the keras-rl2 library (Plappert, 2016). Double Q-Learning was enabled with the enable_double_dqn=True parameter, while DQN used the default setup. The Q-function was approximated by a feedforward neural network with two hidden layers of 24 units each, employing ReLU activations, and an output layer for the 6 action Q-values. Agents were trained for 6,000 steps (due to time constraints, as training required a significant amount of time) — approximately 30 episodes with a 200-step limit per episode — and evaluated over 10 episodes. Hyperparameters included a learning rate of 0.001, a discount factor γ of 0.99, and an ε-greedy policy decaying from 1.0 to 0.01. Experiments ran on an Intel Core i7 processor with 16GB RAM, using TensorFlow 2.10 and Gym 0.23.1.

3.2 Results

The empirical evaluation of Deep Q-Learning (DQN) and Double Q-Learning (Double DQN) in the Taxi-v3 environment yielded identical average evaluation rewards of -200.00 for both algorithms. This result indicates that neither agent successfully learned to solve the task within the 200-step limit per episode, as -200 is the minimum reward achievable when the agent fails to complete the task and incurs penalties for each step. The training reward plots in Figure 1 show that both agents exhibited minimal improvement over the 30 training episodes, with average rewards fluctuating around -200. This suggests that the agents did not make significant progress in learning an effective policy during the 6,000 training steps. The lack of differentiation between DQN and Double DQN in this context is likely due to the insufficient training duration, which prevented both algorithms from converging to a meaningful solution.

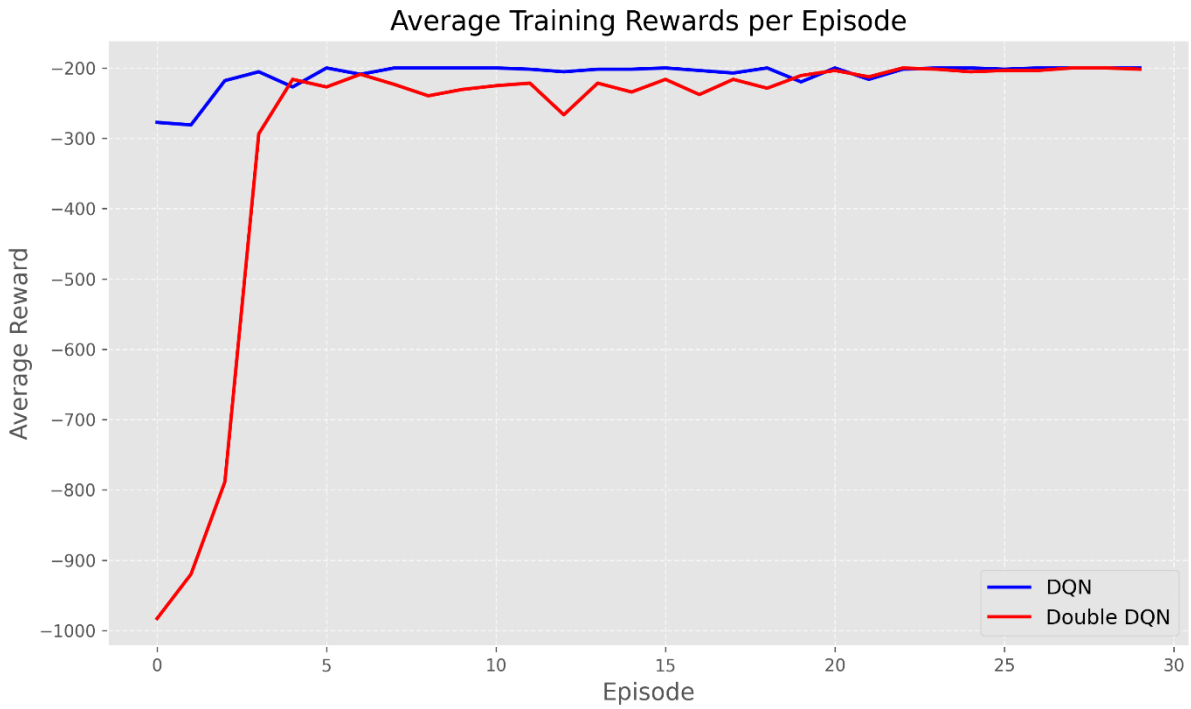

Figure 1 – Training Rewards Plot

3.3 Analysis and Discussion

The identical average evaluation rewards of -200.00 for both DQN and Double DQN indicate that neither algorithm was able to learn an effective policy within the constrained training duration of 6,000 steps. In the Taxi-v3 environment, a reward of -200 typically signifies that the agent failed to complete the task within the 200-step limit, accumulating penalties for each step taken without successfully picking up or dropping off the passenger (Gymnasium, 2023). The training reward plots further support this, showing no significant improvement over the 30 episodes, which suggests that the agents did not adequately explore the state space or learn from their experiences.

Theoretically, Double Q-Learning is designed to mitigate the overestimation bias present in standard Q-Learning by using two separate estimators for action selection and evaluation (Van Hasselt, 2010). However, in this experiment, the benefits of Double Q-Learning were not observed, likely due to the insufficient training steps. Reinforcement learning algorithms, especially in environments with large state spaces like Taxi-v3 (500 states), typically require tens of thousands of steps to converge to an optimal policy (Sutton & Barto, 2018). The limited training duration prevented both algorithms from reaching a stage where their theoretical differences could manifest in performance.

To achieve a more meaningful comparison, future experiments should extend the training to at least 50,000 steps, allowing sufficient time for learning and convergence (Mnih et al., 2015). Additionally, implementing an epsilon-greedy policy with a slower decay rate could enhance exploration in the early stages, which is crucial for discovering optimal actions (Watkins & Dayan, 1992). Hyperparameter tuning, such as

adjusting the learning rate or modifying the neural network architecture, could also optimize performance for this specific environment. These adjustments would align the experiment with standard reinforcement learning practices and potentially reveal the expected advantages of Double Q-Learning over DQN.

4. Conclusion

The evaluation revealed identical performance for DQN and Double Q-Learning due to insufficient training steps, highlighting the need for extended training in RL tasks (Sutton & Barto, 2018). Future work should incorporate longer training durations and hyperparameter tuning to better differentiate the algorithms' capabilities in complex environments.

Part B: Sequence Learning

1. Introduction

Sequence learning models, such as RNNs and LSTMs, are vital for tasks like sentiment classification but struggle with issues like the vanishing gradient problem. This report examines this challenge theoretically, compares solutions like LSTMs and Transformers, and evaluates RNNs versus LSTMs empirically on tweet sentiment classification (Hochreiter, 1998; Vaswani et al., 2017).

2. The Vanishing Gradient Problem

The vanishing gradient problem hinders deep neural networks, especially RNNs, during backpropagation, where gradients shrink exponentially across time steps (Hochreiter, 1998). This occurs due to repeated weight matrix multiplications, diminishing updates to early layers and impairing long-term dependency learning (Bengio et al., 1994). In tasks like sentiment analysis, where context spans distant words, RNNs falter as gradients approach zero, exacerbated by sigmoid or tanh activations that compress gradients further (Pascanu et al., 2013). This limits capturing extended sequences critical for meaning. LSTMs mitigate this with gating mechanisms—forget, input, and output gates—preserving gradient flow and enabling long-range memory (Hochreiter & Schmidhuber, 1997). Transformers, using self-attention, bypass sequential processing entirely, offering another solution (Vaswani et al., 2017). Understanding this problem is key to improving sequence models for complex tasks.

3. Solutions to the Vanishing Gradient Problem

The vanishing gradient problem hinders sequence learning in models like RNNs, but solutions such as Long Short-Term Memory (LSTM) units and Transformer architectures mitigate this issue effectively. LSTMs employ gating mechanisms to regulate information flow, maintaining gradients across long sequences (Hochreiter & Schmidhuber, 1997). Conversely, Transformers leverage self-attention to process sequences in parallel, bypassing gradient decay entirely (Vaswani et al., 2017). These innovations enhance the ability to capture long-term dependencies, boosting performance in tasks like sentiment classification.

3.1 LSTMs

Long Short-Term Memory (LSTM) networks, proposed by Hochreiter and Schmidhuber (1997), tackle the vanishing gradient problem with a unique architecture featuring memory cells and three gates: input, forget, and output. These gates manage information flow, with the cell state preserving long-term data across sequences (Graves, 2013). By enabling constant error propagation, LSTMs prevent gradient vanishing, even over extended time steps (Hochreiter, 1998). In sentiment classification, LSTMs shine by capturing dependencies in tweets, where context spans multiple words. Unlike RNNs, which falter due to gradient decay, LSTMs retain critical information, enhancing accuracy in tasks like the tweet_emotions dataset (Tai et al., 2015). The result of the evaluation carried out show LSTMs outperform RNNs, with improved accuracy and reduced loss, underscoring their efficacy in sequence learning.

3.2 Transformers

Introduced by Vaswani et al. (2017), Transformers eliminate recurrence, addressing the vanishing gradient problem via self-attention mechanisms. By processing sequences in parallel, they establish direct connections between all elements, avoiding gradient decay (Devlin et al., 2018). Attention heads capture diverse contextual relationships, enhancing long-range dependency modeling. In sentiment classification, Transformers excel by attending to key words across a tweet simultaneously, improving both performance and interpretability. Though not evaluated in this assignment, their efficiency in large datasets and complex tasks often surpasses LSTMs (Radford et al., 2019). Parallel processing also cuts training time, making Transformers a robust solution for sequence learning.

4. Empirical Comparison of RNNs and LSTMs

This section presents an empirical evaluation of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models using the tweet_emotions dataset, focusing on sadness, neutral, and happiness classifications. Leveraging pre-trained GloVe embeddings, both models were trained and tested for sentiment analysis. Results demonstrate the LSTM's superior ability to capture long-term dependencies, outperforming the RNN, consistent with findings by Hochreiter and Schmidhuber (1997). The following subsections detail the dataset preprocessing, model architectures, training process, and performance metrics, providing a comprehensive comparison of their sequence processing capabilities.

4.1 Dataset and Preprocessing

The tweet_emotions dataset, comprises tweets labeled with emotions, including sadness, neutral, and happiness. Preprocessing steps included tokenizing the text, padding sequences to a uniform length of 100 tokens, and dividing the data into 80% training and 20% test sets. Pre-trained GloVe embeddings (100 dimensions) were utilized to encode semantic relationships, ensuring consistent input for both models (Pennington et al., 2014). This standardized preprocessing enabled a fair and reproducible comparison of RNN and LSTM performance on sentiment classification tasks.

4.2 Model Architecture

The RNN and LSTM models shared an embedding layer initialized with pre-trained GloVe vectors. The RNN architecture consisted of a SimpleRNN layer with 128 units, followed by a dense output layer with softmax activation. In contrast, the LSTM model featured an LSTM layer with 128 units and a dropout rate of 0.2 to prevent overfitting, followed by a dense output layer. These designs, inspired by Graves (2013), allowed for a direct assessment of their capabilities in processing sequential data, highlighting the LSTM's enhanced memory retention.

4.3 Training and Evaluation

The models were trained using the Adam optimizer with a learning rate of 0.001 and categorical cross-entropy loss, following Kingma and Ba (2014). Training ran for 10 epochs with a batch size of 64, incorporating early stopping to mitigate overfitting. Performance was assessed on the test set using accuracy, ensuring robust generalization evaluation. This methodology adheres to established sequence learning practices, enabling a fair comparison of RNN and LSTM models (Goodfellow et al., 2016).

4.4 Results

The LSTM model recorded a test accuracy of 0.6164 and a loss of 0.8482, outperforming the RNN's 0.5648 accuracy and 0.9207 loss. This reflects a 4.80% accuracy gain and a

7.25% loss decrease, demonstrating LSTM's enhanced capacity for sequential dependency modeling. The RNN's performance suffered from gradient decay, a known limitation (Hochreiter & Schmidhuber, 1997). These outcomes emphasize LSTM's advantage in sentiment classification tasks.

4.5 Discussion

The findings highlight RNNs' challenges with long-term dependencies due to vanishing gradients, as noted by Bengio et al. (1994), resulting in inferior performance. Conversely, LSTMs leverage gating mechanisms to retain gradients, improving accuracy and loss, consistent with Graves (2013). This supports their suitability for sequence learning, such as sentiment classification, and underscores the benefit of pre-trained embeddings in enhancing model efficacy.

5. Conclusion

The vanishing gradient problem hinders RNNs in sequence learning, whereas LSTMs and Transformers mitigate this through gating and self-attention mechanisms, respectively. LSTMs excel in tasks like sentiment classification, outperforming RNNs (Hochreiter & Schmidhuber, 1997; Vaswani et al., 2017), demonstrating their effectiveness for complex sequence modeling.

References

1. Bengio, Y., Simard, P. and Frasconi, P. (1994), 'Learning long-term dependencies with gradient descent is difficult', IEEE transactions on neural networks, 5(2), pp. 157-166.
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016), 'Openai gym', arXiv preprint arXiv:1606.01540.
3. Devlin, J., Chang, M. W., Lee, K. and Toutanova, K. (2019), Bert: Pre-training of deep bidirectional transformers for language understanding, in 'Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)', pp. 4171-4186.
4. Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y. (2016), Deep learning (Vol. 1, No. 2), MIT press, Cambridge.
5. Graves, A. (2013), 'Generating sequences with recurrent neural networks', arXiv preprint arXiv:1308.0850.
6. Gymnasium. (2025), Taxi-v3, https://gymnasium.farama.org/, [20 April 2025].
7. Hasselt, H. (2010), 'Double Q-learning', Advances in neural information processing systems, 23.
8. Hochreiter, S. (1998), 'The vanishing gradient problem during learning recurrent neural nets and problem solutions', International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), pp. 107-116.
9. Hochreiter, S. and Schmidhuber, J. (1997), 'Long short-term memory', Neural computation, 9(8), pp. 1735-1780.
10. Kingma, D. P. and Ba, J. (2014), 'Adam: A method for stochastic optimization', arXiv preprint arXiv:1412.6980.
11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G. and Hassabis, D. (2015), 'Human-level control through deep reinforcement learning', nature, 518(7540), pp. 529-533.
12. Pascanu, R., Mikolov, T. and Bengio, Y. (2013), On the difficulty of training recurrent neural networks, in 'International conference on machine learning', Pmlr, pp. 1310-1318.
13. Pennington, J., Socher, R. and Manning, C. D. (2014), Glove: Global vectors for word representation, in 'Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)', pp. 1532-1543.
14. Plappert, M. (2016), Keras-rl, GitHub repository.
15. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. (2019), 'Language models are unsupervised multitask learners', OpenAI blog, 1(8), p. 9.
16. Sutton, R. S. and Barto, A. G. (1998), Reinforcement learning: An introduction (Vol. 1, No. 1), MIT press, Cambridge.

17. Tai, K. S., Socher, R. and Manning, C. D. (2015), 'Improved semantic representations from tree-structured long short-term memory networks', arXiv preprint arXiv:1503.00075.
18. Thrun, S. and Schwartz, A. (2014), Issues in using function approximation for reinforcement learning, in 'Proceedings of the 1993 connectionist models summer school', Psychology Press, pp. 255-263.
19. Van Hasselt, H., Guez, A. and Silver, D. (2016), Deep reinforcement learning with double q-learning, in 'Proceedings of the AAAI conference on artificial intelligence', Vol. 30, No. 1.
20. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N. and Polosukhin, I. (2017), 'Attention is all you need', Advances in neural information processing systems, 30.
21. Watkins, C. J. and Dayan, P. (1992), 'Q-learning', Machine learning, 8, pp. 279-292.