Regression Tsetlin Machine for Reinforcement Learning

Eivind Hagtvedt[1,2*]

[1*] Department of Information and communication technology , University Of Agder, Jon Lilletuns vei 9, Grimstad, 4879, Agder, Norway.

Corresponding author(s). E-mail(s): eiviha16@uia.no;

Abstract

The Tsetlin Machine is a recent Machine Learning algorithm that uses propositional logic for inference. Its explainability and potential for fast computation whilst maintaining competitive in performance makes it a desirable alternative to artificial neural networks. In this paper the application for Tsetlin Machines is used to solve cartpole by replacing the deep neural network commonly used in Deep Q-Network algorithms with a tsetlin Machine for each action of the agent. It explores the algorithm using common Temporal Difference to calculate the target Q-values as well as multi-step Temporal difference, and experiments where the feedback and update is more controlled. All the experiments solves the environment, but only multi-step temporal difference achieves a perfect score of 500.0. While they all solve cartpole, their performance is highly volatile between training sessions. Eventually their performance collapses and does not recover with the exception of the Multi-step Temporal Difference variant which have consistent peaks throughout the entire training of the algorithm.

Keywords: Reinforcement Learning, Deep Q-Network, Tsetlin Machine

Code available at: Github

1 Introduction

In this paper we are exploring the potential for Tsetlin Machine (TM) in reinforcement learning. Research with the integrating TMs with existing reinforcement learning algorithms have been done with moderate success for cartpole. TMs provide an interpretable algorithm where the behavior of the agent can be explained. Artificial Intelligence (AI) models where the underlying reason behind the inference can be understood may be useful for legal analysis, improving algorithms or making informed decisions. [1]

Among the challenges of machine learning is to balance high performance with energy efficiency, memory footprint and computation speed. This can enable embedded devices to run inference with machine algorithms themselves rather than using cloud solutions. TMs are already showing promise with being able to efficiently trade performance with hardware requirements, and is outperforming alternative solutions such as binary neural networks. [2]

2 Background

Reinforcement learning is a class of machine learning algorithms where an agent learns a policy for an environment to maximize its cumulative reward. During training, the agent will have to balance **exploitation** and **exploration**. The agent can either exploit its current policy by choosing actions that give the highest expected reward, or it can choose actions that may not result in high immediate reward, but overtime result in the agent learning a better policy. Normally, the agent will be set to explore more in the beginning and progress towards exploitation over time. A common type of reinforcement learning methods are **model-free** algorithms where an agent learns by interacting directly with the environment. [3]

Q-learning is an off-policy reinforcement algorithm that uses Q-values, which estimates the cumulative reward of a state action pair [4]. The algorithm chooses the action with the highest Q-value given the state, but with a probability of choosing a random action during the learning process, which is called the ϵ-greedy policy [5]. The randomness is introduced in the algorithm to encourage the agent to explore the environment. With Q-learning the agent alternates between acting in the environment to generate samples, and training on these examples during experience replay [6].

Deep Q-Network (DQN) are similar to Q-learning except that they use a deep neural network to predict Q-values for the actions of the agent given a state [4]. Since DQNs use deep neural networks to calculate the Q-values, it can be used for problems with a continuous state space. While regular Q-learning can be used for continuous state spaces, the discretizing of the states can result in a very large q-table making the algorithm computationally inefficient [4].

Temporal Difference (TD) is a concept used in DQN's, although instead of calculating the TD, the Bellmann equation 1 is used to calculate the target Q-value [7]. The discount factor γ is used to determine whether the algorithm should favor immediate rewards versus future rewards. To update the DQN the mean squared error between the target Q-value and the predicted Q-value of the network is commonly used. [6]

$$Q^* \leftarrow MSE(Q(s_t, a_t), R_t + \gamma * max(Q(s_{t+1}, a_{t+1}))) \tag{1}$$

A variation of TD is the multi-step TD algorithm where the target Q-value is calculated using n transitions into the future. This has been used to produce more accurate predictions. The equation for the target value can be seen in 2. [6]

$$Q^* \leftarrow MSE(Q(s_t, a_t), \sum_{i=0}^{n}(\gamma^i R_{t+i}) + \gamma^n max(Q(s_{t+n}, a_{t+n})) \tag{2}$$

Tsetlin Machines are a recent approach to machine learning problems, and uses propositional logic to form its predictions [8]. The TM is created by using a series of Tsetlin Automata (TA) each of which receives a binary input and then produces a conjunctive clause of literals of positive and negative polarity for each feature [9]. Each TA represents one literal in a clause . For a clause to be true, all the literals in the clause have to be true. The output in the common TM is decided by clause voting, where all the clauses that are true are summed together representing one class and the clauses that are false representing the other class. Then the majority class is chosen as the final output [9]. With the Regression Tsetlin Machine (RTM) it is slightly different. Instead, the clauses are summed together and used to calculate an output within a predefined range [10]. Equation 3 shows how the prediction is calculated. y is the summed clause output by the prediction and T is the threshold.

$$Y = 1.0 * y * \frac{y_{max} - y_{min}}{y_{min}} \tag{3}$$

The output from the RTM is not continuous, and depends on the number of clauses to define the resolution of the output. This can become computationally heavy in circumstances where the resolution have to be high. To reduce the computational cost Morten Goodwin et.al. [11] introduces the Integer weighted RTM, which turns multiple clauses that captures the same sub-pattern into one.

The TM is updated with Type I and Type II feedback. Type I feedback is used to reinforce true positive patterns, whilst Type II feedback reinforces the TM to discriminate, reducing false positives. The clauses are updated depending on the value of T, which enables the control of the size of the updates to the TM. For the RTM the type of update is decided by whether the predicted value is lower than the target value, resulting in Type I feedback, or whether it is higher, resulting in Type II feedback. This is seen in equation 4 Additionally, the number of clauses updated is also proportional to the size of the error, where larger errors results in more clauses being updated. [8]

$$Feedback = \begin{cases} \text{Type I, if } y > \hat{y} \\ \text{Type II, if } y < \hat{y} \end{cases} \tag{4}$$

To make the TM more robust to noisy data, the clauses are updated with varying levels of intensity depending on how close the summation of the clauses are to a threshold T. For Type I feedback when the summation of the clauses are closer to threshold the probability of update is reduced. Similarly, for Type II when the summation of the clauses are closer to 0 the probability of update is reduced. Adjusting the threshold can be used to control the TM's ability to express sub-patterns in the data. [12]

To control how strongly the include action is favored for a TA the specificity s, is used. A higher specificity means literals are more likely to be included and less likely to be excluded. Increasing specificity can boost the learning of infrequent sub-patterns. [12]

3 Related works

Off-policy and on-policy reinforcement learning with the Tsetlin Machine is a paper by Ole-Christoffer Granmo and Saeed Rahimi Gorji that explores RTM for both off-policy and on-policy reinforcement learning [9]. In this study the RTM is integrated with the off-policy value iteration algorithm as well as with the on-policy SARSA algorithm to solve the gridworld problem. They find that using multi-step TD in combination with high frequency propositional logic patterns makes their on-policy algorithm learn significantly faster. Both off-policy and on-policy produced comparable results to their two layer neural network model.

Interpretable Reinforcement Learning with the Regression Tsetlin Machine by Varun Ravi Varma, replaces the neural network in the Deep Q-Network with RTM to solve cartpole and mountain car environments from the gymnasium library [13]. The experiments were largely performed with changes to the RTMs parameters and with Integer Weighted RTM. The results showed that the RTMs performance was unstable, but that it showed potential to learn a policy for the cartpole environment. However, it did not solve the environment.

Exploring the Potential of Model-Free Reinforcement Learning using Tsetlin Machines by Didrik K. Drøsdal et. al. [14] is a paper that explored the TM's potential to solve the cartpole environment and pong. Their paper is comprised of two studies. The first was to explore the TM's ability to learn a policy by having it train on the generated dataset of a pretrained Proximal Policy Optimization model and on the dataset of an optimal mathematical solution for cart pole. The second study involves model-free learning by with three approaches. The first is having a Multiclass TM learn to output the correct action by training it on a modified version of cartpole that gives reward based on the poles deviation from the upright position. The second was double TM-Learning, which reuses the concepts from double Q-Learning. This was done to reduce the instability of the TM agent's performance. The final strategy was to use sequential batching, where the TM was trained on data that prioritizes successive actions that lead to good outcomes. The results of the first study in the paper shows that the TM can learn the policies of simple environments such as cartpole and in the second study it discovered that using sequential batching showed the most promise, although it did not perform as well for pong.

4 Method

This section details the approach to the solution and the experiments performed.

4.1 Combining the Regression Tsetlin Machine with Q-Learning

To integrate the TM with the DQN the TM substitutes the Deep Neural Network (DNN) in the DQN algorithm. Instead of a single DNN, there are two Integer Weighted RTMs, one for each action of the cartpole agent. Each RTM is only updated on its own action. This means that the calculated target Q-value is only used to update the RTM that was chosen as the action during the interaction with the environment. The range of values for each RTM is set to 100.

The input values are binarized using observation data generated from the cartpole environment, and is set to have 25 bits per feature, and the RTMs are bootstrapped by using observation examples along with randomized output values. The gamma value for the TD is 0.8 and 0.85 for multi-step TD. The $\epsilon - greedy$ algorithm is set with exploration probability defined in 5 where α refers to the decay constant which is 0.001.

$$\epsilon_{n+1} = \epsilon_n * e^{-\alpha} \tag{5}$$

The models will be tested after every fifth training session, where it is tested on a validation set of 100 episodes, the best performing models are then saved. The final test is on another 100 episodes with the mean and standard deviation being reported.

4.2 Experiment 1

The first experiment is done with the Integer Weighted RTMs integrated with a classical Q-learning algorithm to verify that the RTM Q-learning (TMQN) implementation can learn the environment. The tests will be done on a range of parameter values to find a decent fit. The primary parameters that will be explored is the number of clauses, threshold, specificity and the number of bits in each automaton.

4.3 Experiment 2

There is no control on what type of feedback the RTMs get, and so the algorithm may end up with an excess of one particular type of feedback, resulting in the RTMs not discriminating well enough or discriminating too much. We will therefore perform an experiment where the RTMs receives a balanced portion of training samples where the agent fails and where the agent succeeds. When the agent fails the target value is low, whilst when the agent succeeds the target value is high. This means that the RTMs is more likely to receive Type I feedback when the agents action is successful, and more likely to receive Type II feedback when the action results in terminating the episode. To ensure that there are samples available of both instances for the agent to train on, the replay buffer is divided in two, one for successful actions and one for actions that results in failure. When sampling from the replay buffer it will sample with a set probability for each. The tests will be for a ratio of 0.25, 0.50 and 0.75 of successful actions. Additionally, a dynamic version will be tested where the sampling is based on the inverse ratio of the feedback from the previous sample it was trained on.

This is counter to having a strict enforcement of Type I and Type II feedback. The problem with this approach is that if the algorithm receives no feedback of a particular type, it might just stop receiving any feedback altogether, or very little. One could go further with forcing the algorithm to use as many samples as is necessary until a balance of feedback was achieved for a training session, but this could make the training highly inefficient. Ultimately it's the balance between data the TM should discriminate against and data that it should reinforce that is important.

4.4 Experiment 3

There is also no control on whether both RTMs are engaged in the decision making process. It could occur that one of the RTMs has a greater range of Q-value predictions, and therefore is always the decision maker which may reduce performance. We therefore want to ensure that both RTMs are always training during a training a session. This is done by separating the replay buffer into two, one for each of the RTMs. The tests will be done on various ratios of sampling to study the effect that this has on the results.

4.5 Experiment 4

We will also employ the multi-step TD algorithm. When the agent makes predictions based purely on the current and next reward, the agents decisions are highly biased towards immediate rewards, which may not benefit the algorithm in the long run. Additionally, in the same sense that experiment 2 uses the fail cases to boost Type II feedback, having the algorithm consider longer successive actions may balance the feedback since it results in a higher ratio of samples with instances where the agent falls over.

5 Experimental results

This section showcases the results from each of the experiments. The mean score on the validation set over training cycles of the best performing models from each experiment is seen in graph 1a, with the benchmarks being placed separately in 1b for clarity. The feedback from these models is shown in 4a and their mean Q-values are seen in 2. The best performing models of each experiment can be seen in table 5.

5.1 Experiment 1

The results for experiment 1 in table 1 shows that the TMQN performs the best with 1000 clauses, a threshold of 750 and a specificity of 10, but a specificity of 5 wasn't much worse. A specificity of 15 on the other hand resulted in poor performance, as well as with a specificity of 1. The bit memory for a TA does not seem to affect results too strongly other than when the memory is very high.

The performance of the one with the best parameters seen in graph 1a first begins climbing, but eventually collapses resulting in very low performance and the Q-values show a similar collapse where they make predictions are quite low until the end where it begins to show signs of recovery, but this is not reflected in the score. The feedback is predominantly Type I feedback, with occasional spikes of Type II feedback.

Table 1: Results for experiment I

Regression Tsetlin Machine Q-Network					
clauses	T	s	bits	mean	std
500	250	5	6	205.77	98.97
1000	750	5	6	458.82	96.36
1500	1000	5	6	101.56	71.87
500	250	10	8	171.61	59.69
1000	750	10	8	**493.12**	20.37
1500	1000	10	8	47.03	30.53
500	250	15	8	122.73	109.49
1000	750	15	8	278.9	198.93
500	250	15	6	32.83	21.15
1000	750	15	6	293.16	198.72
1500	1000	15	6	51.39	34.14
1000	750	10	6	487.67	30.41
1000	750	1	8	142.69	113.81
1000	50	10	8	209.71	83.45
1000	250	10	8	423.5	126.26
1000	500	10	8	450.76	75.21
1000	750	10	4	480.72	66.76
1000	750	10	20	335.08	99.66

Figure 3b shows that the Q-value predictions of each RTM in the model are inversely related. If one of the RTMs have a higher than the mean Q-value prediction, the other one will have a lower predicted value.

5.2 Experiment 2

Table 2 shows that the change in ratio of feedback did not have a significant impact on performance, with all of them succeeding quite well, with the exception of the run where the probability ratio was based on the inverse ratio of the previous training session. This solution performed poorly compared to the rest of them. In figure 4b the feedback ratio for each trial, shows that the majority of them mostly received Type I feedback, with spikes of type II feedback, except for the dynamic one which had mostly type II feedback in the first half of the training. The one with with a balanced sampling ratio of 0.5 remained close to a 0.5 ratio of Type I and Type II feedback.

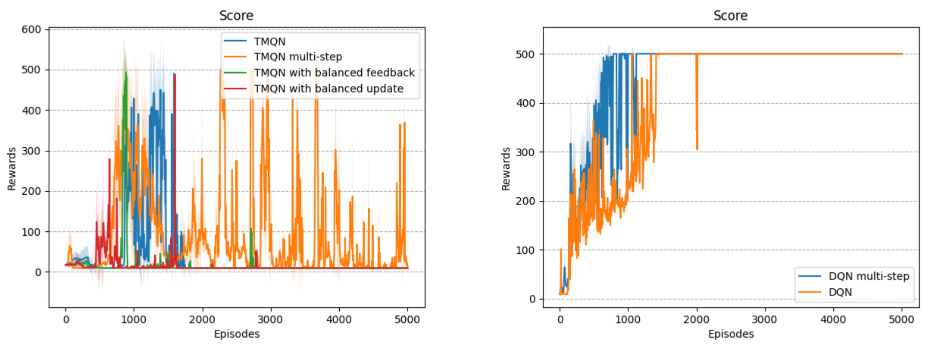
The model with a sampling ratio of 0.25, shows similar behavior to experiment 1 where the performance is improving at first and then collapses. This is also reflected in the Q-value predictions where the Q-values falls sharply, but without any recovery. The Q-values also have occasional spikes where it has made very high Q-value predictions, which may suggest that the algorithms have received too much Type I feedback for all of its clauses, or that the integer weighing is assigning very high values to some clauses.

5.3 Experiment 3

In table 3 the results for experiment three shows that the agent performs the best when the ratio of update between the RTMs is equal, although the first TM being updated with a ratio of 0.25 produces significantly better results than when it was being updated with a ratio of 0.75.

Table 2: Results for experiment II

Regression Tsetlin Machine Q-Network with feedback balance						
clauses	T	s	bits	ratio	mean	std
1000	750	10	8	0.25	**498.35**	8.29
1000	750	10	8	0.50	463.95	75.41
1000	750	10	8	0.75	383.41	156.81
1000	750	10	8	dynamic	254.21	205.91



(a) Tsetlin Machine Q-Networks (b) Deep Q-Networks

Fig. 1: The performance of the selected algorithms over 5000 episodes of training. The benchmarks algorithms are in their own graph for clarity.

Table 3: Results for experiment III

Regression Tsetlin Machine Q-Network with update balance						
clauses	T	s	bits	ratio	mean	std
1000	750	10	8	0.25	465.37	100.6
1000	750	10	8	0.5	**491.63**	36.02
1000	750	10	8	0.75	206.06	186.78

Similarly to the previous experiments the performance of the best one peaks early, and then collapses without recovery, with the same being seen in the Q-values. It also has mean Q-value predictions higher than the DQN and multi-step DQN in the first half and then falls to 0 rather quickly, which matches that of experiment 2, but without the enormous spikes.

5.4 Experiment 4

The results for experiment four with the n-step temporal difference reward function in table 4 shows that using 5 or 10 steps produced the best results with a perfect score. 2 steps and 20 steps produced more variable results.

The multi-step TMQN has highly unstable performance, varying wildly between a high score and a very low one. While the previous experiments eventually started

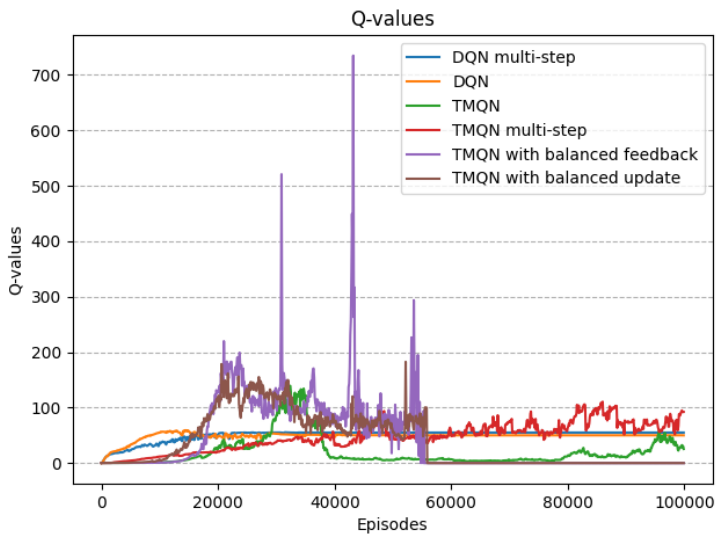
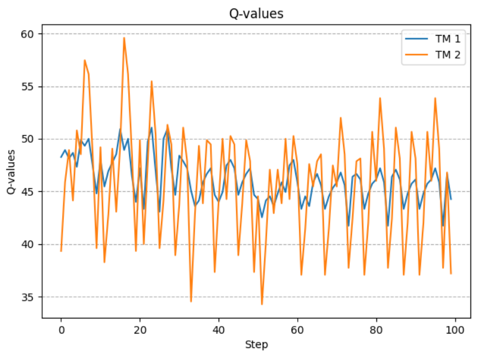


Fig. 2: The mean Q-value prediction of the best performing algorithm from each experiment and the benchmark algorithms over the course of the training.

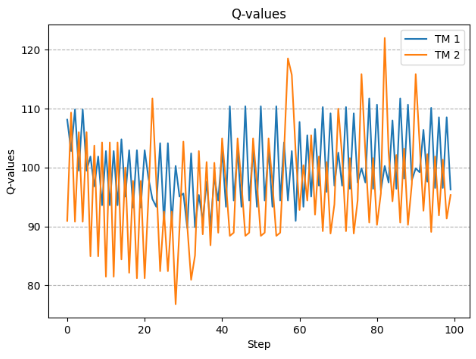
Table 4: Results for experiment IV

Regression Tsetlin Machine Q-Network with Multi-step TD						
clauses	T	s	bits	steps	mean	std
1000	750	15	6	2	434.98	68.34
1000	750	15	6	5	**500.0**	0.0
1000	750	15	6	10	**500.0**	0.0
1000	750	15	6	20	496.72	30.69
1000	750	10	8	2	435.95	91.95
1000	750	10	8	5	433.58	94.61
1000	750	10	8	10	482.1	67.95
1000	750	10	8	20	305.49	109.91

performing poorly without recovery, the multi-step TMQN recovers after collapsing. The mean Q-value predictions matches the predictions made by the benchmark algorithms. In graph 3a the subset of Q-values from the best model shows that the second RTM have a much greater range of Q-value predictions than the other one. They also do not share an inverse relationship in their predictions, where if one predicts a higher Q-value the other is likely to do the same as it is with the regular TMQN. This is likely due to the increased variance introduced by using the multi-step TD, where the optimal action regards future steps as well.

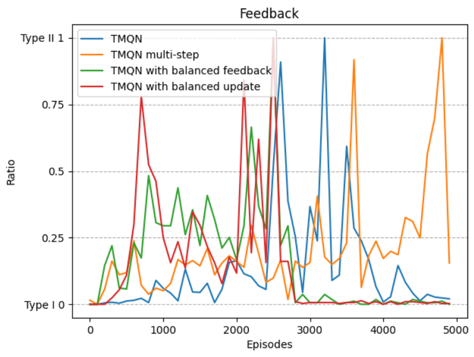


(a) Q-values multi-step TMQN

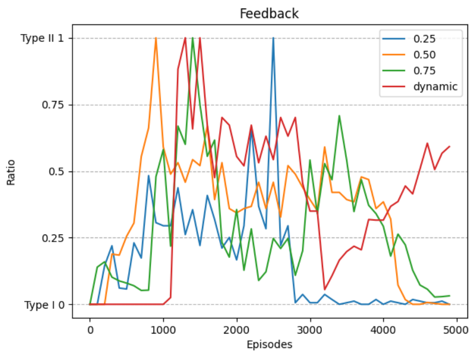


(b) Q-values TMQN

Fig. 3: A subset of the Q-values for the best performing TMQN and multi-step TMQN.



(a) Feedback for the best of each experiment.



(b) Feedback for experiment 2.Feedback

Fig. 4: The feedback ratio with regards to Type II feedback between Type I and Type II feedback. A value closer to 0 means a higher ratio of Type I feedback.

Table 5: Comparing the results of the best performing model from each experiment and the benchmark algorithms.

Performance for Cartpole		
Algorithm	mean	std
Benchmarks		
DQN	500.0	0.0
multi-step DQN	500.0	0.0
Best performance from each of the experiments		
TMQN	493.12	20.37
TMQN with feedback balance	498.35	8.29
TMQN with update balance	491.63	36.02
multi-step TMQN	**500.0**	0.0

6 Discussion

All the best models from each experiment performed quite well and achieved a score above 475 which shows that using RTM in combination with Q-learning can solve cartpole, and suggests that it may also be used to solve other reinforcement learning environments as well. Using multi-step TMQN seems to produce better results more consistently, but given the volatility of the results, there could be a factor of luck involved. Overall using TMQN with multi-step TD seems to be the most promising.

Having more clauses did not result in better performance, with 1000 being ideal. Considering the simplicity of the cartpole environment it is possible that the number of clauses required to solve more complex environments such as bipedal walker will be very high. This can adversely affect the efficiency and explainability of using Tsetlin Machines for reinforcement learning.

The Q-values for the multi-step TMQN not diverging too far from the Q-values for the DQN means it makes fairly good Q-value predictions and that it may suggest that it will not collapse with more training. The Q-value predictions for the regular TMQN are very high compared to what is produced by the DQN benchmark, suggesting that the predicted Q-value is not very accurate. This may suggest that the regular TMQN is not as viable as the multi-step TMQN.

The model performed better with a specificity of 15 for the multi-step TMQN whilst it performed quite poorly for the regular variant. This could be due to a high specificity for the regular TMQN results in overfitting whilst it does not result in overfitting when using multi-step TD. This could mean that when using multi-step TD it requires higher specificity to pick up on useful sub-patterns. Having a relatively high threshold of $\frac{3}{4}$ of the number of clauses seems to be ideal, this indicates that the clauses updating with high intensity to represent sub-patterns in the data without risk of over-fitting.

Among the TMQN with balanced feedback, the one with a ratio that is intended to favor Type I feedback had the best results, but it did not have significantly more type I feedback than the other experiments. Whilst it is difficult to to effectively control the feedback ratio, there seems to be some degree of influencing it. Having a ratio of 0.75 Type II feedback may result in too many clauses being 0, so that the feedback after this becomes consistently Type I. Nevertheless, controlling the feedback did not stabilize the models.

6.1 Instability

All of the RTM solutions are highly unstable, with greatly varying performance between each test. This could be due to the literals in the RTMs flipping too easily, making it vulnerable to noise. Whilst it results in a significant drop in performance, it is clear that it can relearn optimal patterns again for a while as the algorithms recovered quickly. Although, eventually some of them did not recover.

In the third experiment, having one RTM with a ratio of training being 0.25, should not produce a greatly different result than if the the ratio was the inverse. This reflects the overall unreliability of the results, where the performance of the algorithms can vary greatly.

6.2 Unlearning

For some of the experiments the models go through a process of unlearning where the agent receives minimal reward and does not recover. The predicted q-values for both of the RTM falls to 0. Logically this would happen due to too much Type II feedback resulting in all the clauses being false. When both of the Q-values are calculated to be the same it always chooses the first RTM over the other. It is unclear why this happens, as the subsequent dominance of Type I feedback should result in clauses switching back to being true again.

7 Conclusion & future work

In this paper the RTM algorithm was used in combination with Q-network to solve cartpole, a common reinforcement learning environment. The paper shows that the environment can be solved whilst using the RTM, and can achieve a high score for both a general RTM Q-Network and other variations. It is however highly unstable in that its performance will change greatly after a few cycles of training. Additionally, the algorithms will eventually start unlearning and collapse in performance. Using multi-step TD can mitigate this collapse, but will experience similar levels of instability as the other variations.

The challenges with using TMs for reinforcement learning, such as instability may be addressed by increasing the memory of the TMs as it may allow patterns to strengthen such that they are harder to unlearn. This would make it less susceptible to occurrences where it receives feedback that unlearns favorable patterns.

Additionally, using a double DQN type solution for TMQN may improve the stability of the agent. The way to mimic the soft update of the evaluation network of the Double DQN for a Double TMQN solution, would be to change a percentage of the clauses at each update.

References

[1] Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., Zhu, J.: Explainable ai: A brief survey on history, research areas, approaches and challenges. In: Tang, J., Kan, M.-Y., Zhao, D., Li, S., Zan, H. (eds.) Natural Language Processing and Chinese Computing, pp. 563–574. Springer, Cham (2019)

[2] Maheshwari, S., Rahman, T., Shafik, R., Yakovlev, A., Rafiev, A., Jiao, L., Granmo, O.C.: REDRESS: Generating Compressed Models for Edge Inference Using Tsetlin Machines. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(9), 11152–11168 (2023) https://doi.org/10.1109/TPAMI.2023.3268415

[3] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. Institute of Electrical and Electronics Engineers Inc. (2017). https://doi.org/10.1109/MSP.2017.2743240

[4] Mishra, S., Arora, A.: A Huber reward function-driven deep reinforcement learning solution for cart-pole balancing problem. Neural Computing and Applications (2022) https://doi.org/10.1007/s00521-022-07606-6

[5] Van Hasselt, H.: Double Q-learning. Technical report

[6] Hernandez-Garcia, J.F., Sutton, R.S.: Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target (2019)

[7] Yang, L., Shi, M., Zheng, Q., Meng, W., Pan, G.: A Unified Approach for Multi-step Temporal-Difference Learning with Eligibility Traces in Reinforcement Learning (2018)

[8] Darshana Abeyrathna, K., Granmo, O.C., Zhang, X., Jiao, L., Goodwin, M.: The regression Tsetlin machine: A novel approach to interpretable nonlinear regression. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **378**(2164) (2020) https://doi.org/10.1098/rsta.2019.0165

[9] Rahimi Gorji, S., Granmo, O.C.: Off-policy and on-policy reinforcement learning with the Tsetlin machine. Applied Intelligence **53**(8), 8596–8613 (2023) https://doi.org/10.1007/s10489-022-04297-3

[10] Abeyrathna, K.D., Granmo, O., Jiao, L., Goodwin, M.: The regression tsetlin machine: A tsetlin machine for continuous output problems. CoRR **abs/1905.04206** (2019) 1905.04206

[11] Abeyrathna, K.D., Granmo, O.-C., Goodwin, M.: A Regression Tsetlin Machine with Integer Weighted Clauses for Compact Pattern Representation (2020)

[12] Granmo, O.-C.: The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic (2018)

[13] Ravi Varma, V.: Interpretable Reinforcement Learning with the Regression Tsetlin Machine. Technical report (2022)

[14] Drøsdal, D.K., Grimsmo, A., Andersen, A., Granmo, O.-C., Goodwin, M.: Exploring the Potential of Model-Free Reinforcement Learning using Tsetlin Machines. Technical report