Stabilizing Transformer-Based Action Sequence Generation For Q-Learning

Gideon Stein [1],
Andrey Filchenkov [2], **Arip Asadulaev** [3]
ITMO University
St. Petersburg, 198215
[1]gideon@steinml.de, [2]afilchenkov@itmo.ru, [3]aripasadulaev@itmo.ru

Abstract

Since the publication of the original Transformer architecture (Vaswani et al. 2017), Transformers revolutionized the field of Natural Language Processing. This, mainly due to their ability to understand timely dependencies better than competing RNN-based architectures. Surprisingly, this architecture change does not affect the field of Reinforcement Learning (RL), even though RNNs are quite popular in RL, and time dependencies are very common in RL. Recently, Parisotto et al. (2019) conducted the first promising research of Transformers in RL. To support the findings of this work, this paper seeks to provide an additional example of a Transformer-based RL method. Specifically, the goal is a simple Transformer-based Deep Q-Learning method that is stable over several environments. Due to the unstable nature of Transformers and RL, an extensive method search was conducted to arrive at a final method that leverages developments around Transformers as well as Q-learning. The proposed method can match the performance of classic Q-learning on control environments while showing some limited potential on selected Atari benchmarks. Furthermore, the method was critically evaluated to give additional insights into the relation between Transformers and RL.

Introduction

Transformer architectures revolutionized the field of Natural Language Processing (NLP). The classic Transformer (Vaswani et al. 2017) and its successors such as Devlin et al. (2018) or Radford et al. (2019) outperform traditionally used RNN-based architectures on the majority of tasks in NLP. Their superior performance can be mostly attributed to their ability to understand timely dependencies and notably long-term dependencies better than RNN-based methods. Timely dependencies are not only interesting for NLP tasks. In reinforcement learning, such an ability is useful to perform in environments that are only partially observable. RNN-based methods are traditionally deployed to such environments. Furthermore, there are multiple successful examples of applications of the Attention mechanism (the core functionality of the Transformer) to RL (Iqbal and Sha 2019), (Oh et al. 2016) and (Manchin, Abbasnejad, and van den Hengel 2019). Due to these facts, the deployment of Transformer-based architectures to RL is a promising research direction. While there were several studies on Transformer-based methods in RL, many of them such as Upadhyay et al. (2019) or Mishra et al. (2017) reported random performance for their Transformer-based approaches. Even on simple MDP or Multi-armed Bandit problems. Contrary to that, the first major success with Transformer-based RL methods was recently reported by Parisotto et al. (2019). So while it is possible to use Transformer-based architectures in RL, it seems to be a nontrivial task. In RL, the information signal is affected by past decisions. This creates dependencies and makes optimization harder than training on a fixed dataset. Additionally, Transformer-architectures are quite hard to optimize which was already stated in (Vaswani et al. 2017). As an example, they are strongly dependent on a specific learning rate schedule to be optimized. Together, these two conditions make the optimization of Transformer-based architectures in RL challenging.

To support the results of Parisotto et al. (2019) and to further evaluate the possibility of Transformer-based models in RL, this paper seeks to create a new Transformer-based RL method that, contrary to Parisotto et al. (2019), is based on the original Transformer from (Vaswani et al. 2017) instead of the Transformer-XL (Dai et al. 2019). Furthermore, this work is alternatively using the Transformer-based model as a value function for a Deep Q-learning agent. While the approach of Parisotto et al. (2019) showed strong performance, its optimization method, as well as their Transformer model, are quite complex. On contrary, this paper analyzes the behavior of a simple transformer-based model in the context of a well-known optimization method (Q-learning) to add to a clear understanding of the interrelations between RL and transformer-based models. Finally, it is hoped that results that are reported in this paper encourage additional research, which is needed to fully understand the relation between Transformers and RL.

Background

Deep Q-learning

The goal of Q-learning is to find a function that correctly maps state (s) action (a) pairs to their corresponding value for an agent that interacts with an environment. The simplest form of Q-learning is defined as updating a Q-function in the following manner:

$$\begin{aligned} td =& R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \\ Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha * td \end{aligned} \tag{1}$$

where R is a reward, γ is a discount factor and Q represents the value of any pair (s,a). By updating Q-values after rewards, the greedy policy as-well as the value function change frequently. Under the condition that all states are explored sufficiently, these updates are guaranteed to converge to a Q-function that correctly represents the environment. Based on this Q-function, a policy will be formed by greedily sampling the action with the highest Q-value at every step. Traditionally, the Q-function was implemented as a table. However, it is possible to approximate it with a neural network. This method is known as Deep Q-learning. When using a function approximator for the Q-function, the definition of an update changes since it is only possible to update weights of the network and not specific Q-values directly. An update for a Deep Q-network (DQN) is therefore defined as:

$$\begin{aligned} &target = R(s, a, s') + \gamma \max_{a'} q_k(s', a') \\ &td = \left[(q_\theta(s, a) - target)^2\right] \\ &\theta_{k+1} \leftarrow -\alpha * \nabla_\theta \mathbf{E}_{s'\ P(s'|s,a)} td \Big|_{\theta=\theta_k} \end{aligned} \tag{2}$$

where θ represents the network weights and s' and a' are the state and action in the timestep t+1. When a network is updated according to Formula 2, an issue arises. Data that is acquired by an agent interacting with an environment is quite different from a fixed dataset that is normally used to train neural networks. Today, Replay Buffers, a method to save experience and reuse it during model training, and target networks, a method to make the target of the update more stable, are used to counter these issues. By applying these two methods to Deep Q-learning, Mnih et al. (2013) opened the field of Deep Reinforcement Learning. Their method will be the baseline RL method for the course of this work.

The Transformer Architecture

The Transformer model is a sequence to sequence architecture (seq2seq) that was initially developed to perform translation tasks in NLP, and which relies heavily on the Attention mechanism. A seq2seq structure is defined as a model that takes in a sequence of signals and returns a sequence of outputs. Also, the Transformer is an Encoder-Decoder structure that splits into two distinct submodels. An Encoder that transforms an input sequence into an encoded representation and a Decoder that generates, based on the encoded representation, a new sequence as an output. Since the proposed architecture is based on the Encoder of the classic Transformer (Vaswani et al. 2017), its structure will be discussed further. The Transformer Encoder takes in some word tokens and transforms them into the same number of encoded representations. To do this efficiently, the Transformer stacks several identical blocks on top of each other. These blocks are called Encoder layers. Additionally, the Encoder features an

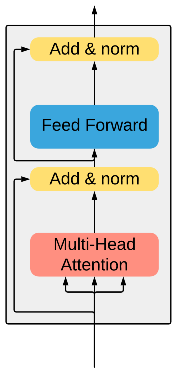


Figure 1: The standard Transformer Encoder layer

Embedding layer and a positional encoding of the input sequences which both are added before the first layer. A single Encoder layer is constructed out of two main components. An Attention block and a feed-forward network. Additionally, residual connections and normalization are added. Fig. 1 shows the structure of the Encoder layer. Note that the input and the output of the Encoder have the same dimension. This makes layer stacking possible. The computation that takes place in a single Encoder layer is defined as:

$$\begin{aligned} out_1 &= Norm(Attention(X) + X) \\ out_2 &= Norm(FF(out_1) + out_1) \end{aligned} \tag{3}$$

where X represents an input tensor with the shape (batch size, input sequence length, model dimension). Typically, Dropout is deployed after the Attention block and after the feed-forward block.

The Attention mechanism

Attention is a mechanism that understands the importance of specific inputs for other inputs and combines these into a new vector that includes this information. This mechanism does not rely on a hidden representation that includes all past information but attends directly to the full inputs. This helps Attention to perform better than RNN-based approaches in many cases, especially when long-term dependencies are present and relevant. Based on Attention, Multi-Head Attention is performed by multiple Attention operations in parallel on sub-parts of the inputs. This allows attending multiple sub-areas of inputs at once. The Transformer features the use of Scaled Dot Product Attention as well as Multi-Head Attention to understand dependencies. Scaled dot product Attention is defined as the operation on three inputs. Keys (K), Queries (Q), and Values (V):

$$\begin{aligned} out_1 &= QK^T \\ out_2 &= out_1 / \sqrt{dim_{key}} \\ out_3 &= softmax(out_2)V \end{aligned} \tag{4}$$

Where Q, K, V are input matrices, and dim_{key} is the last dimension of K. Intuitively, this can be understood as a way to scale and add the content of V by a factor that is a combination of Q and V. Through this channel, V attends to the information that is included in Q and K and is altered accordingly. To perform Multi-Head Attention, the initial input vector is simply split. When performing Multi-Head Attention in the Encoder, the embedded input sequence represents K, Q, and V. This specific form of Attention is called Self-Attention, since the input sequence attends to itself. It is a key component that allows the Encoder to encode the input sequence efficiently.

Transformers for Q-learning

Transformer-based Q-Networks

This paper proposes to use an altered version of the Transformer Encoder as a Q-network for a Q-learning agent. However, the original structure has to be altered slightly to be usable. To map to Q-values at the end of the model, the output of the Encoder has to be mapped to the Q-value dimension which is achieved by adding a fully connected layer after the last Encoder layer. Also, the embedding layer of the classic Transformer has to be replaced by a fully connected layer that maps from the state dimension to the model dimension. After these two steps, a Transformer-based Q-network (TBQN) that can map from states to Q-values is obtained. It can be examined in Figure 2.

The literature (Parisotto et al. 2019), (Upadhyay et al. 2019), (Mishra et al. 2017) suggests, that a Q-learning agent using the proposed TBQN would be very hard to optimize and most likely unstable. To preemptively counter this, a method variation search space was constructed which includes three categories. Firstly, changes to the model structure itself. Secondly, the application of additional methods for DQNs and Transformers. Both of these categories represent small model or method variations that are proposed in the literature and might be able to improve the performance of TBQNs. Thirdly, a selection of possible impactful Hyperparameters is included. This search space was then filtered to find a method variation that is easier to optimize and more stable than a base Q-learning agent featuring the base TBQN.

Transformer layer variations

Since the original publication of the Transformer (Vaswani et al. 2017), many Transformer layer variations were introduced in the literature. These structural changes are exclusively made to make the Transformer more stable during training. From this literature, several Transformer layer variations were selected to be tested as the core layer for TBQNs.

Dropout free models (layer type 2) Since Transformers were initially developed for NLP, they feature the usage of Dropout layers. Typically implemented to counter overfitting, the usage of Dropout in RL is not popular. Due to this, a layer without Dropout was tested. Additionally, all layer variations are tested with and without Dropout after the final layer. This layer variation is displayed in Figure 3a.

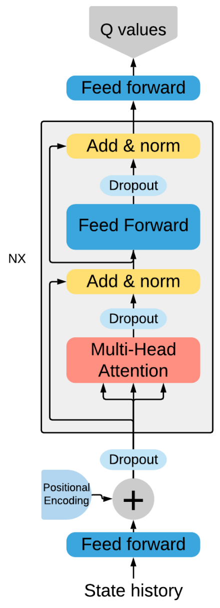


Figure 2: The proposed Transformer-based Q-network

Identity Map Reordering (IMR) (layer type 3) A layer variation that was described by Parisotto et al. (2019). It features the positional change of the normalization layer to the start of each sub-layer. Furthermore, an additional ReLU activation after every sub-layer was added to prevent two linear layers in a row. Its implementation can be observed in Figure 3b.

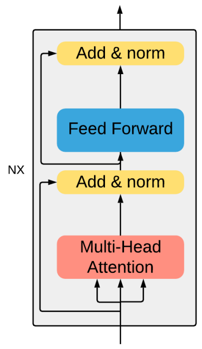
Pre layer Normalization (layer type 4) Very similar to IMR, this variation described by Xiong et al. (2020) changes the position of the layer normalization to the beginning of each sub-layer. While this is identical to IMR, this variation does not feature an additional ReLU activation. Its implementation can be observed in Figure 3c.

Output gate connections (layer type 5) Also described by Parisotto et al. (2019) this variation based on IMR additionally replaces the residual connection with a gated layer. While residual connections were initially implemented to improve the training of deep neural networks, they seem to make training Transformers more unstable. They are replaced with the following gate formulation, where W and b are trainable parameters:

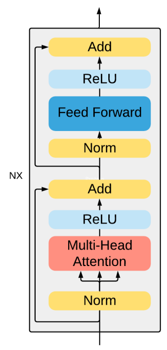
$$g_l(x, y) = x + \sigma(W_l^g x - b_l^g) \odot y \tag{5}$$

This variation was also already tested for Transformer-based methods in RL and it will be used as it was proposed in (Parisotto et al. 2019).

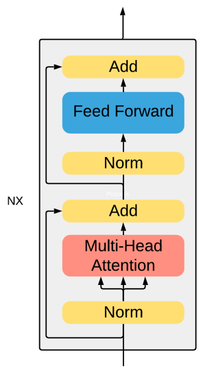
GRU gate connections (layer type 6) Finally, another variation will be tested which features the usage of a different gating mechanism based on a GRU unit. Again, this variation was introduced by Parisotto et al. (2019) and is based on IMR. Noteworthy is that this model variation combined with Maximum a Posteriori Policy Optimization (Song et al.



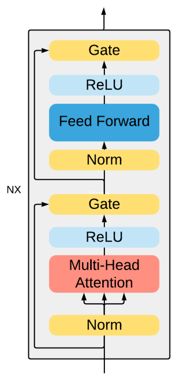
(a) Layer type 2 (no dropout)



(b) Layer type 3 (IMR)



(c) Layer type 4 (norm first)



(d) Layer type 5/6 (gated)

Figure 3: The Transformer Encoder layer variations

2019) achieved SOTA results for DMLab-30. It remains to be seen if this is also the case for Q-learning. The mechanism is defined by Formula (6). W and U are trainable parameters.

$$
\begin{aligned}
H &= \tanh(W_l^H y + U_l^H (R \odot x)) \\
Z &= \sigma(W_l^Z y + U_l^Z x - b_l^g) \\
R &= \sigma(W_l^R y + U_l^R x) \\
g_l(x, y) &= (1 - Z) \odot + Z \odot H
\end{aligned}
\tag{6}
$$

Additional methods and Hyperparameters

Additionally to these layer variations, the following methods and Hyperparameters were included categorically in the search space to test their effect on the performance of a TBQN:

- Double Q-learning
- Target update period
- Target update (τ) (Lillicrap et al. 2015)
- Gradient Clipping
- Learning rate schedules
- Depth-Scaled Initialization (Zhang, Titov, and Sennrich 2019)
- Depth-Scaled Initialization of the last Layer (Zhang, Titov, and Sennrich 2019)
- Number of Attention Heads
- Initial collection steps
- Loss function
- Environment normalization
- Epsilon Greedy
- Replay Buffer size
- Future reward discount (γ)
- Batch size
- Learning rate
- Encoder type (whether or not dropout is used outside of the Encoder layers)

Experiments

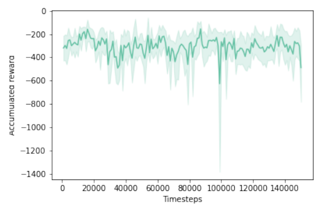
Baseline performance

To motivate the method variation search and to set a base performance of TBQNs, a Q-learning agent with the proposed base TBQN and with no special additions (except a Replay Buffer and a Target Network) was evaluated. The agent was trained on four environments (MountainCar-v0, Acrobot-v1, CartPole-v1, and LunarLander-v2) for 150k steps. All these environments are implemented by OpenAI GYM (Brockman et al. 2016). The accumulated rewards of episodes during training can be examined in Figure 4 . 10 training runs per environment were executed. The agent was not able to solve any environment sufficiently, had a high fluctuation, and even diverged on some occasions (LunarLader-v2 reward drop). For these experiments, the following Hyperparameters were used. Initial collect steps: 1000, mean squared loss, 4 Attention Heads, epsilon greedy: 0.1, Replay Buffer length: 100000, batch size: 32, learning rate: 1e-4. The rest of the parameters were not used. It shows clearly, that TBQNs need additional help to perform.

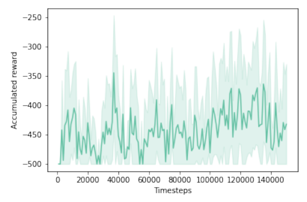
Selecting the optimal method variation

While it would be ideal to test every possible method variation, this is unfeasible due to computational complexity. Due to that, a two-step method based on two distinct studies was constructed to find a well-performing method variation.

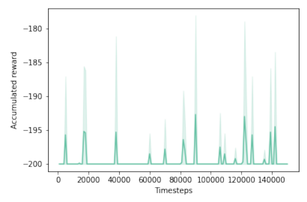
Study one - Parameter importance The first study focused on narrowing down the method search space significantly. This was achieved by estimating the Mean Decrease Impurity Importance Score for all parameters in the method search space. Based on these scores, parameters with low importance were excluded entirely. Furthermore, parameters with high importance were further evaluated to select the best performing values and exclude the rest from the search space. To estimate these scores, the method search space had to be sampled and evaluated. Since grid search was infeasible, a Tree-Structured Parzen Estimator, which was firstly described by Bergstra et al. (2011), was used to sample from the search space. Every method search space sample was trained for 15k steps. As a final performance score, the



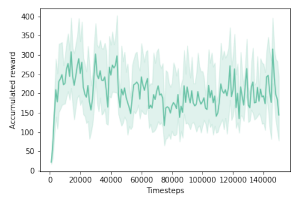
(a) LunarLander-v2



(b) Acrobot-v1



(c) MountainCar-v0



(d) CartPole-v1

Figure 4: Accumulated rewards of episodes during training of a base Q-learning agent with the proposed TBQN base in four different environments.

average return of the last 10 episodes was used. To guarantee generality, the study was conducted independently in three different environments (CartPole-v1, Acrobot-v1, and LunarLander-v2) and the final importance score for every parameter was averaged between these environments. All studies were performed on a single GPU (Nvidia1080Ti). Further information can be found in Appendix B.

Study two - Final selection After having narrowed down the method search space significantly, the remaining search space samples were evaluated further to find the model variation with the best performance. Two methods were used to determine the effect of certain parameter values on the performance of TBQNs and to select a final method variation: On one hand, the mean reward of the last ten episodes between all samples where a certain parameter value was present was calculated. On the other hand, the search space samples with the highest rewards for every environment were extracted. This was done to determine whether combinations of specific parameter values performed especially well. Again, the study was conducted in three different environments (CartPole-v1, Acrobot-v1, and LunarLander-v2) to guarantee generality. All search space samples were trained for 75K steps in the environments CartPole-v1 and AcroBot-v1 and for 150k steps in the environment LunarLander-v2. All experiments were conducted on a single Nvidia GPU(1080ti). Further information can be found in Appendix B.

Results

Based on the two studies, the method variation represented by Table 1 was selected as it performed better than the baseline model in all environments and proved to be stable during training. The parameters initial collect steps, Environment normalization, Replay Buffer size, τ, double Q-learning, and the Encoder type had low importance for control environments and are not specified. The following comments should be made to accompany this selection:

- The optimal values for several parameters are environment-dependent (denoted by Semi-fixed). This means the performance of a Q-learning agent using a TBQN relies strongly on the right value selection. The optimal values however change from environment to environment.

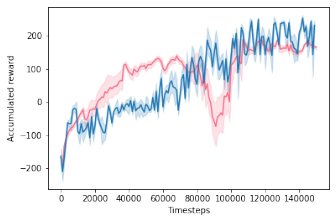
Parameter	Value	Category
Gradient Clipping	True	Fixed
Batch size	32	Fixed
Learning rate	1e-4	Fixed
Layer type	3	Fixed
Custom lr schedule	”No”	Fixed
Depth-Scaled Initialization	1	Fixed
Num Heads	4/2	Fixed
Target upate period	10+	Semi-fixed
Epsilon Greedy	(0. - 1.)	Semi-fixed
Depth-Scaled Initialization (last layer)	(T/F)	Semi-fixed
Loss function	(Huber, Squared)	Semi-fixed
γ	(.99, .95)	Semi-fixed

Table 1: Final method variation

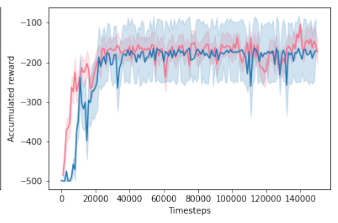
- Surprisingly, IMR layers (layer type 3) perform the best while GRU-gated layers (layer type 6) were excluded early due to frequent divergence.
- While being very important for NLP, learning rate schedules are not required for TBQNs. It is estimated that TBQNs with layer variations do not require learning rate schedules which makes them obsolete.
- Depth-Scaled Initialization (Zhang, Titov, and Sennrich 2019) is beneficial. Models that were initialized with it tended to diverge less and achieved higher average rewards at the end of training.
- Gradient Clipping is very important for TBQNs. Since the Transformer has problems with divergence in the RL setting, Gradient Clipping helps to mitigate destructive updates.
- Several parameters are not important for model performance (Assuming no abstruse values). During the parameter search, they showed no significant impact on the performance of TBQNs.

Performance in control environments

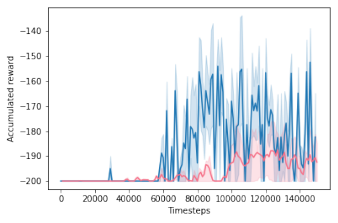
To evaluate the performance of the method variation specified in table 1, it was compared to an optimized classic Q-



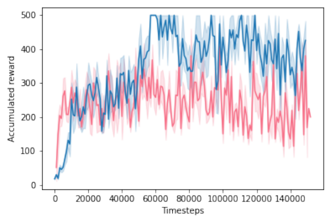
(a) LunarLander-v2



(b) Acrobot-v1



(c) MountainCar-v0

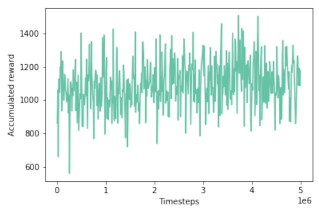


(d) CartPole-v1

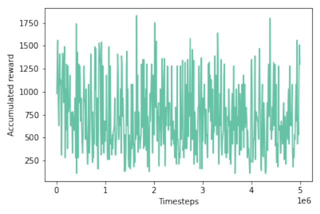
Figure 5: Accumulated rewards of episodes during training of the proposed method (orange) and of an optimized classic Q-learning agent (blue) in four control environments

learning agent in four Environments (CartPole-v1, Acrobot-v1, MountainCar-v0, and LunarLander-v2). The method was extracted from Rl-zoo baselines (Raffin 2018), a collection of Hyperparameter optimized methods. By examining Figure 5, it is visible that the performance of the proposed method increased significantly compared to the baseline method. Furthermore, while it can not match the classic Q-learning agent on CartPole-v1 and MountainCar-v0, it performs very similar on LunarLander-v2 and Acrobot-v1. Finally, the proposed method shows clear and consistent learning on all environments except on CartPole-v1 which is a important feature for further evaluation. Especially, since the baseline is does not show much learning at all.

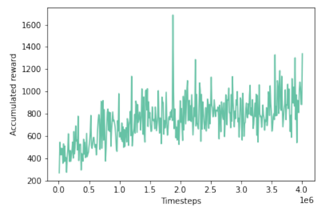
Performance in ATARI environments



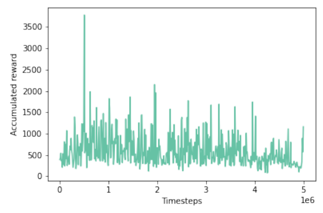
(a) ”Asteroids-ram-v0” (run 1)



(b) ”Asteroids-ram-v0” (run 2)



(c) ”MsPacman-ram-v0” (run1)



(d) ”MsPacman-ram-v0” (run2)

Figure 6: Accumulated rewards of episodes during training on ATARI

Additionally to the control environments, the proposed method variation was trained in two environments (”MsPacman-ram-v0” and ”Asteroids-ram-v0”) of the popular ATARI benchmark which is implemented by OpenAI GYM (Brockman et al. 2016). Two parameters that are not included in 1 were scaled up from their initial values to match the complexity of the new environment and to keep them in reasonable ranges. The initial collect steps were increased from 1000 to 5000. Additionally, the Replay Buffer size was increased from 100k to 200k. For both environments, the RAM state which consists out of 128 pixels was used as the state vector for training. Unfortunately, the performance of the proposed method on ATARI is inconclusive. Fig. 6 shows four selected training runs and the accumulated rewards of episodes during training. Firstly, a strong difference between training runs can be reported. This difference is especially visible between the two ”MsPacman-ram-v0” runs. While the first run shows a stable improvement during the whole period of training, the second run shows no learning. Interestingly, when saving the best performing method state during training, the proposed method can be temporary compared to the final performance after training of different DQN methods (Table 2). While surprising, such a comparison can of course not be used to form big conclusions about TBQNs. While it is believed, based on these results, that TBQN based methods can perform in ATARI environments, it is still a challenging task. More experiments must be conducted to form a final conclusion. It is also suspected that conducting the parameter search in control environments, might have harmed the performance of TBQNs in ATARI environments. We are positive, that this challenge can however be overcome by committing more computational resources in the future.

Methods	Asteroids	MsPacman
DQN^2	1629 +/- 542	2311 +/- 525
DQN^1	1070+/-345	2363 +/-735
RQN^1	1020 +/-312	2048+/-653
TBQN	1813+/- 396	1555+/-696

Table 2: Reported average returns of different methods on ATARI. 1 = (Hausknecht and Stone 2015), 2 = (Mnih et al. 2015)

Conclusion

During this work, the interaction of Transformer architectures and Deep Q-learning was evaluated. The goal of this work was to craft a new RL method based on the combination of Deep Q-learning and Transformer-based models which was partly successful. Through an extensive method variation search, a Transformer-based Deep Q-Learning method was constructed which leverages developments around Transformers as well as Q-learning. The proposed method can compete with the performance of an optimized classic Q-learning agent on multiple control environments while showing some limited potential on selected Atari environments. However, this research project is still a work in progress. The testing of the proposed final method variation on more environments and especially environments that require a deep understanding of past states is still essential to form a final conclusion. The results of this work are complementary to Parisotto et al. (2019) and another step to a better understanding of Transformer architectures in RL. This work defies past results that neglect Transformer architectures in RL and shows that they can be used when being handled carefully. While the proposed method is connected to the one that was used by Parisotto et al. (2019), it represents a different version of a Transformer-based RL method that can be deployed, tuned, and tested more easily. To further encourage this, the code base of this research can be accessed under (Stein 2020). It is hoped that this work can help to support new studies on the topic of Transformers in RL and leverage them to RL mainstream.

Appendix

A. Model specifications

Throughout this work, the TBQN dimensions specified in Table 3 were used. All experiments and studies were conducted on a single GPU (Nvidia1080Ti). Specific parameters that are not explicitly defined can be observed in the experiment scripts available at (Stein 2020)

Specification	Control	ATARI
History horizon	5 steps	3 steps
Encoding Dimension	64	64
Number of Layers	3	2
Dff Dimension	256	256

Table 3: TBQN dimensions throughout this work

B. Study specifications

Table 4 and Table 5 hold additional information concerning the studies that were conducted to arrive at a final method variation.

Specification	Value
Number of evaluated search space samples	30
Number of environments	3
Runs per sample	2
Training steps	15k

Table 4: Additional information for study 1

Specification	Value
Remaining search space samples	24
Number of environments	3
Runs per sample	2
Training steps	150k / 75k

Table 5: Additional information for study 2

References

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, 2546–2554.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym.

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q. V.; and Salakhutdinov, R. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* .

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Hausknecht, M.; and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

Iqbal, S.; and Sha, F. 2019. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, 2961–2970. PMLR.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* .

Manchin, A.; Abbasnejad, E.; and van den Hengel, A. 2019. Reinforcement learning with attention that works: A self-supervised approach. In *International Conference on Neural Information Processing*, 223–230. Springer.

Mishra, N.; Rohaninejad, M.; Chen, X.; and Abbeel, P. 2017. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141* .

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533.

Oh, J.; Chockalingam, V.; Singh, S.; and Lee, H. 2016. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128* .

Parisotto, E.; Song, H. F.; Rae, J. W.; Pascanu, R.; Gulcehre, C.; Jayakumar, S. M.; Jaderberg, M.; Kaufman, R. L.; Clark, A.; Noury, S.; et al. 2019. Stabilizing Transformers for Reinforcement Learning. *arXiv preprint arXiv:1910.06764* .

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1(8): 9.

Raffin, A. 2018. RL Baselines Zoo. URL https://github.com/araffin/rl-baselines-zoo.

Song, H. F.; Abdolmaleki, A.; Springenberg, J. T.; Clark, A.; Soyer, H.; Rae, J. W.; Noury, S.; Ahuja, A.; Liu, S.; Tirumala, D.; et al. 2019. V-MPO: On-Policy Maximum a Posteriori Policy Optimization for Discrete and Continuous Control. *arXiv preprint arXiv:1909.12238* .

Stein, G. 2020. Gideon-Stein/TBQN. URL https://github.com/Gideon-Stein/TBQN.

Upadhyay, U.; Shah, N.; Ravikanti, S.; and Medhe, M. 2019. Transformer Based Reinforcement Learning For Games. *arXiv preprint arXiv:1912.03918* .

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Xiong, R.; Yang, Y.; He, D.; Zheng, K.; Zheng, S.; Xing, C.; Zhang, H.; Lan, Y.; Wang, L.; and Liu, T.-Y. 2020. On layer normalization in the transformer architecture. *arXiv preprint arXiv:2002.04745* .

Zhang, B.; Titov, I.; and Sennrich, R. 2019. Improving deep transformer with depth-scaled initialization and merged attention. *arXiv preprint arXiv:1908.11365* .