TRANSFORMER BASED REINFORCEMENT LEARNING FOR GAMES

Uddeshya Upadhyay†, *Nikunj Shah** *Sucheta Ravikanti*$^{\alpha}$, *Mayanka Medhe*†

†Department of Computer Science and Engineering, Indian Institute of Technology-Bombay
*Department of Mechanical Engineering, Indian Institute of Technology-Bombay
$^{\alpha}$Department of Electrical Engineering, Indian Institute of Technology-Bombay

ABSTRACT

Recent times have witnessed sharp improvements in reinforcement learning tasks using deep reinforcement learning techniques like Deep Q Networks, Policy Gradients, Actor Critic methods which are based on deep learning based models and back-propagation of gradients to train such models. An active area of research in reinforcement learning is about training agents to play complex video games, which so far has been something accomplished only by human intelligence. Some state of the art performances in video game playing using deep reinforcement learning are obtained by processing the sequence of frames from video games, passing them through a convolutional network to obtain features and then using recurrent neural networks to figure out the action leading to optimal rewards. The recurrent neural network will learn to extract the meaningful signal out of the sequence of such features. In this work, we propose a method utilizing a transformer networks which have recently replaced RNNs in Natural Language Processing (NLP), and perform experiments to compare with existing methods.

***Index Terms*—** Deep Learning, Transformers, Q-Learning, Long Short Term Memory (LSTM), Natural Language Processing (NLP)

1. INTRODUCTION AND RELATED WORK

Recent advancements in reinforcement learning have witnessed the heavy use of Deep Neural Networks (DNN) to perform many of the reinforcement learning tasks such as prediction and control. These classes of approaches at the intersection of deep learning and reinforcement learning are known as Deep reinforcement learning (DRL). DRL uses deep learning and reinforcement learning principles to create efficient algorithms that can be applied on areas like robotics, video games, finance, and healthcare [11]. Implementing neural networks like deep convolutional networks with reinforcement learning algorithms such as Q-learning, Actor Critic or Policy Search results in a powerful model (DRL) that is capable to scale to previously unsolvable problems [12]. That is because DRL usually work with raw sensors or image signals as input (for instance in Deep Q Networks - DQN for ATARI games [13]) and can receive the benefit of end-to-end reinforcement learning as well as that of convolutional neural networks. In other words, neural networks act as function approximators, of action-value functions used in predicting the next best action, which is particularly useful in reinforcement learning when the state or action space is too large to be completely known or to be stored in memory.

Recently, DQN with recurrent layers also called as Deep Recurrent Q Networks (DRQN), have started showing promising results in reinforcement learning problem. They have the capability to outperform the DQN's outcomes and generate better trained agents in certain scenarios. The primary reason for this being, DQN has limited or no amount of distant history. In practice, DQNs are often trained with just a single state representation corresponding to current time-steps (i.e often times they neglect the temporal aspects of the input), even when fed with the sequence of state representation, standard DQN architecture without RNNs are unable to capture and extract from the temporal aspect of the sequences. Thus DQN will be unable to master games that require the player to remember events more distant in the past. Put differently, any game that requires a memory of past states in the trajectory will appear non-Markovian because the future game states (and rewards) depend on more than just DQN's current input. Instead of a Markov Decision Process (MDP), the game becomes a Partially-Observable Markov Decision Process (POMDP) [14]. Hence, LSTM along with DQN is used in such cases.

Since LSTM or recurrent neural networks, in general, have had a strong presence in the Natural Language Processing (NLP), we tried to implement a technique inspired from NLP called Transformer to perform the reinforcement learning tasks. Transformer was first introduced in [15] to perform sequence-to-sequence translation, however, it has been adapted successfully in various applications spanning language, speech, etc.

NLP uses a sequence-to-sequence architecture at the core in various tasks. In NLP we generally need to analyze a sequence of words and generate another sequence of words based on them, therefore language translation task is one such example where sequence to sequence architecture is applicable. LSTM was a popular choice to be used as an encoder and decoder for the above-specified task and related architecture. LSTM based approach implicitly accounts for 'attention' which is a mechanism that looks at an input sequence and decides at each step which other parts of the sequence are important. The transformer was a novel technique introduced to replace this attention-mechanism performed by LSTM by more effective and explicit attention-mechanism. It is also an encoder-decoder model but differs from LSTM by avoiding any usage of recurrent neural networks which is common in LSTM, GRU, etc. This improves training time as well as accuracy in NLP tasks.

2. METHODS AND EXPERIMENTS

In the following we describe the techniques we used to extend the current framework to accommodate our transformer based proposal and training procedure, we also describe the various experiments performed to compare methods.

Natural Language Processing often deals with the problem of predicting one set of sequences from another set of sequences (for instance, a sequence of words from the sequence of acoustic features for automatic speech recognition tasks, or sequence of words in German from a sequence of words in English for language translation task, etc). As described in the above section, deep reinforcement learning also makes use of the sequence and we take inspiration from recent updates in NLP to propose a new method for DRL.

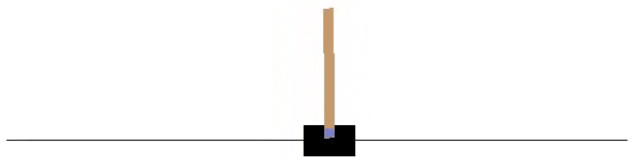


Fig. 1: Frame from the Cartpole environment of OpenAI Gym. The task is to balance the pole on the cart, by moving the cart left or right

2.1. Environment

In our experiments, we set up an environment for the "Cartpole" game (using OpenAI gym [16]) where the goal is to balance the pole on the cart (i.e., prevent the pole from falling) by moving the cart left or right. Figure 1 shows a frame from the game to visualize the environment. The set of actions A consists of $\{$*left*, *right*$\}$ and the environment provides a reward from the set of rewards R consisting of $\{+1, -1\}$. For ever time-step where the pole does not fall the environment provides a reward of $+1$ and when the pole falls (i.e., the angle it makes from the cart crosses a certain threshold) the episode is completed and the agent receives a reward of -1. The typical deep reinforcement learning pipeline passes the frames (or sequence of frames) through a deep convolutional neural network to extract the useful features, which are further processed to estimate the *value function* (V) or the *state-action value function* (Q). However, this results in model with relatively larger number of parameters, which also requires access to large GPUs to train the models using learning algorithms, not to mention that such algorithms take significantly longer to train. In order to overcome this problem we decided to use the RAM provided by the OpenAI environment describing the state of the game. In this case the state of the game can be described using the:

- position of the cart (on the X-axis, from -4.8 to 4.8)
- velocity of the cart (from $-\infty$ to ∞)
- angle the pole makes with the cart (from -24 to 24)
- pole velocity at tip (from $-\infty$ to ∞)

However, to make the problem more interesting and difficult we set up a partially observable Markov decision process (POMDP), where the system dynamics are known to follow an MDP but the agent can not directly observe the states. In our experiments, the agent only observes the partial state consisting position of the cart and the angle pole makes with the cart. Different algorithms evaluated in this paper takes in the sequence of partially observed game states (a window of previous states preceding the current state) and predicts the action to be taken at the current state. The intuition is that the sequence of partially observed states should be sufficient for the algorithms to learn about the missing state features and act optimally.

In this work, we evaluate three different classes of algorithms namely, 1) Deep Q-Networks (DQN), 2) Deep Recurrent Q-Networks (DRQN) and 3) Deep Transformer Q-Network (DTQN).

2.2. DQN, DRQN, and DTQN

Deep Q-Learning (DQN) uses a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. In a typical Deep Q learning setup, all the past experiences are first stored in the memory, the next action is then determined by using epsilon greedy policy with respect to current Q values and the final loss is computed using equation 1, where S_t, a_t, r_t is the state, action taken and reward received at time-step t and S_{t+1} is the state at the next time-step.

$$L = ||r_t + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, a_t)||_2^2 \qquad (1)$$

The term $r_t + \gamma \max_{a \in A} Q(S_{t+1}, a)$ in equation 1 is known as the *target* and will change erratically at every time-step as

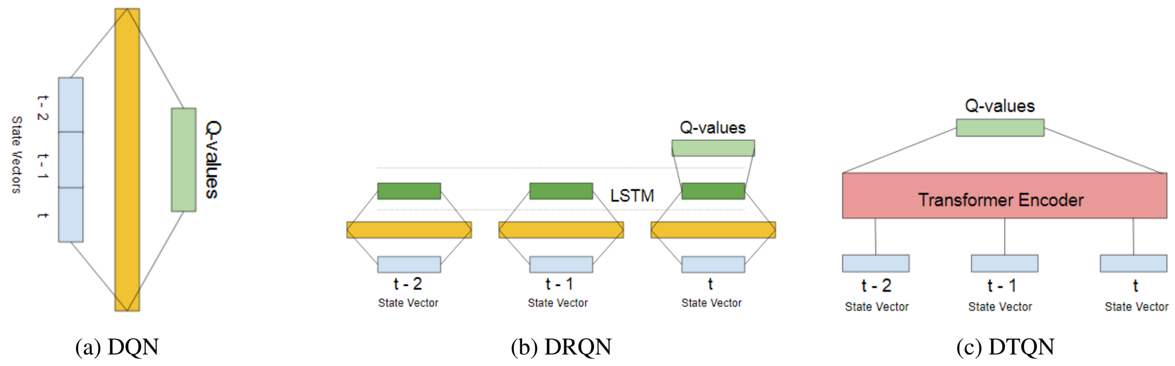


Fig. 2: Different representative architectures. (a) DQN, (b) DRQN, (c) DTQN.

the Q values will change erratically at every time-step, in order to make the learning more stable we use a second copy of the deep neural network called *target network*. The target network has the same architecture as the function approximator but with frozen parameters. For every C iterations (a hyperparameter), the parameters from the prediction network are copied to the target network. This leads to more stable training. Figure 2a shows a representative architecture for the DQN. In our experiments, the input to DQN is the feature vector produced by concatenating the partially observed states from current and time-steps with the partially observed states of previous time-steps (4 time-steps in total, including current time-step).

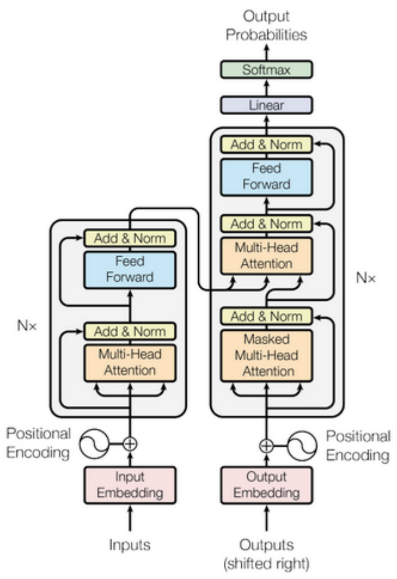


Fig. 3: Transformer: encoder taking input sequence and decoder taking output sequences

Deep Recurrent Q-Learning (DRQN) uses a recurrent neural network to approximate the Q-value function. Sequence of states is given as the input and the network consists of RNN unit (LSTM/GRU), the output of the RNN unit at the final time-step is used to predict the Q-value of all possible actions is generated. Such networks are also often trained using the loss function defined equation 1. Figure 2b shows a representative architecture for DRQN. In our experiments, the RNN unit (GRU) is fed the sequence of partially observed states, the output from the final time-step is used to predict the Q-value function.

The idea behind Deep Transformer Q-learning (DTQN) is to use a *transformer* instead of RNN to extract the meaningful feature out of the input sequence. However, transformers were designed to work for sequence to sequence (seq2seq) tasks such as automatic speech recognition or language translation. But in this work we do not have s sequence to sequence task, rather we need to predict the Q value function given the input sequence.

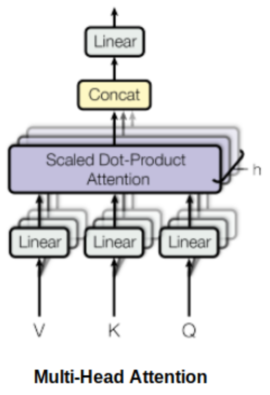
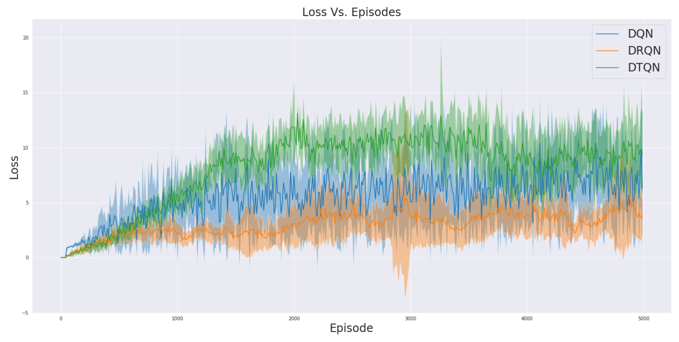
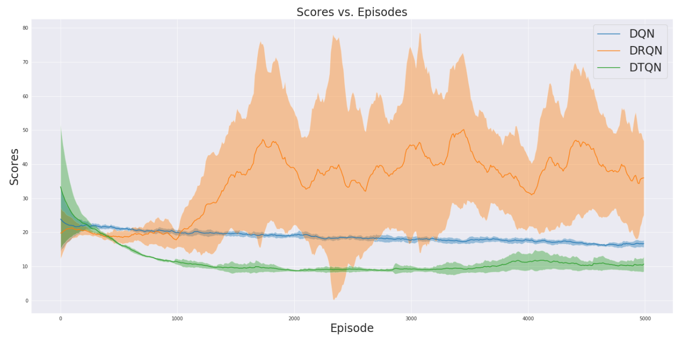
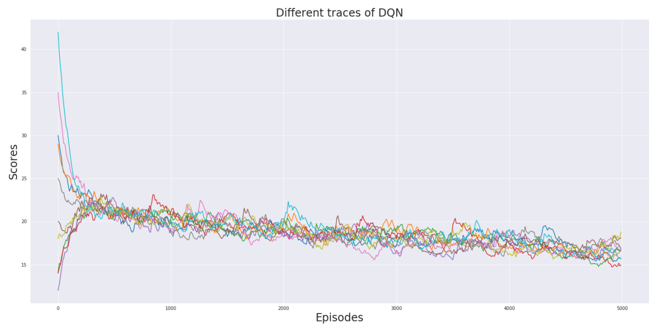


Fig. 4: Transformer: encoder taking input sequence and decoder taking output sequences

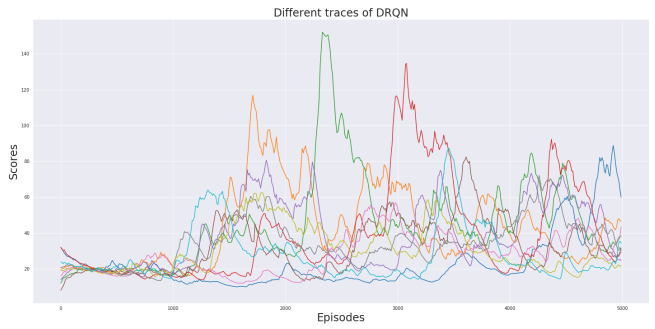
The transformer model consists of encoder and decoder modules, as shown in figure 3. The encoder module processes the input sequence of embeddings (tokens from input sequence are first embedded and then passed through the network) through a multi-head attention module followed by a feed-forward neural network. In our experiments we only use



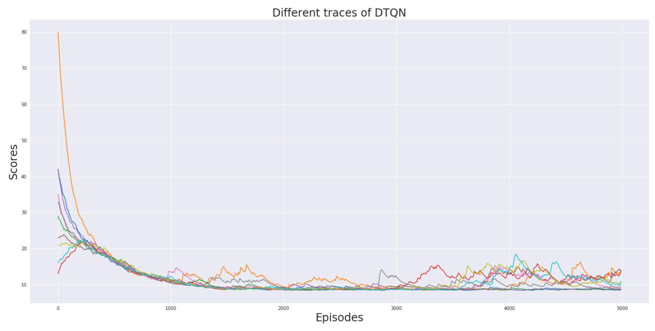




(a) traces of DQN



(b) traces of DRQN



(c) traces of DTQN

Fig. 5: Scores vs Episodes for multiple runs of different algorithms. (a) DQN, (b) DRQN, (c) DTQN.

the encoder module (since our task is not Seq2Seq) to extract the features from input sequence which are further used to predict the Q-values. The multi-head attention module in the encoder refers to the self-attention layer which is a mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Figure 4 shows the multi-head attention module used in the encoder, V, K, Q denotes the value vector, key vector, query vector. Figure 2c shows a representative architecture for DTQN.

3. RESULTS AND DISCUSSIONS

We did multiple experiments for each of the following DQN, DRQN and DTQN based reinforcement learning algorithms. Each algorithm was trained for 5000 episodes and we ran 10 different instances for each of the algorithms with random initialization. Figure 5 shows the value of the scores over episodes for different runs. We see that DQN and DTQN performed worse compared to our DRQN model. While on average the scores for the DQN and DTQN are decreasing, we see that there are a few experiments where the scores improve during training as shown in the graph, but for a majority of the traces, the scores did not improve when using DQN or DTQN. However, the results show significant improvement in scores when using the DRQN model.

As we used location and angle as the only inputs to our neural networks determine the next action, the scores and loss function did not show very promising trends in all three cases. Still, it was possible to make a distinction as to which one of the three: DQN, DRQN, or DTQN performed better by analyzing their relative scores and loss values. DRQN gave the best score results with the maximum reaching to 135 in one of the test cases. DTQN performed the worst on average.

We believe that a transformer-based approach is indeed not suitable for solving the reinforcement learning problem at least in cases where input is taken in the form of a pair of location value and angle value. Reason for the failure can be pointed out as follows: LSTM based approach (DRQN) captures the temporal attributes of the sequence of inputs whereas transformer-based approach tries to put attention on different time-steps of the sequence to obtain the representation for input sequence and does not explicitly tries to capture the temporal aspect of the sequence. As our problem is more of temporal related as the next state is the state at the next time step, LSTM achieves better results. We would also like to add that during our course of work we found that training RL algorithm, for video games, based on DQN or neural networks, in general, is difficult to train and the performance greatly depends upon the random initialization. As can be seen from the traces, some of the random initialization perform too bad while a few perform well.

4. CONCLUSIONS AND FUTURE WORK

In this work, we propose a new transformer based model for reinforcement learning, the inspiration for the same is derived from the fact that the recent advancements in NLP have been achieved by moving away from RNNs and introducing the new transformer based model. Even though the standard transformer consists of both encoder and decoder modules as it's designed to perform Seq2Seq task, we decided to use only the encoder module with the multi-head attention mechanism from the transformer to extract important features from the states to learn the Q-values. We perform experiments on a POMDP and compared multiple algorithms. We conclude that using the state representation provided by the RAM of the game (X-coordinate of the cart and the angle made by pole with the cart), the conventional DRQN (based on RNNs such as LSTM and GRU) perform better than DQN or the proposed DTQN. As part of future work, we intend to compare different algorithms using different state representation like images instead of RAM values.

5. REFERENCES

[1] Julian Togelius Jialin Liu Ruben Rodriguez, Philip Bontrager, "Deep reinforcement learning for general video game ai," *arXiv preprint arXiv:1806.02448*, 2018.

[2] Michael I Jo Tommi Jaakkola, Satinder P Sin, "Reinforcement learning algorithm for partially observable markov decision problems," *In Advances in Neural Information Processing Systems 7 (NIPS), pages 345-352*, 1995.

[3] Devendra Singh Chaplot Guillaume Lample, "Playing fps games with deep reinforcement learning," *arXiv preprint arXiv:1609.05521*, 2018.

[4] Li Dong Jianpeng Cheng and Mirella Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.

[5] David Grangier Denis Yarats Jonas Gehring, Michael Auli and Yann N. Dauphin, "Convolutional sequence to sequence learning," *arXiv preprint arXiv:1705.03122v2*, 2017.

[6] Jack W. Rae Emilio Parisotto, H. Francis Song, "Stabilizing transformers for reinforcement learning," *arXiv preprint arXiv:1910.06764v1*, 2019.

[7] KyungHyun Cho Junyoung Chung, Caglar Gulcehre and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[8] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.0480*, 2019.

[9] Naren Ramakrishnan Chandan K. Reddy Yaser Keneshloo, Tian Shi, "Deep reinforcement learning for sequence-to-sequence models," *arXiv preprint arXiv:1805.09461v4*, 2019.

[10] John N. Tsitsiklis and Benjamin Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, 1997.

[11] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al., "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[13] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov 2017.

[14] Matthew Hausknecht and Peter Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, "Openai gym," 2016.