MAT501 Applied Mathematics & Artificial Intelligence

Using Deep Q-learning Networks in Tetris

Charlie Joshi 2104598

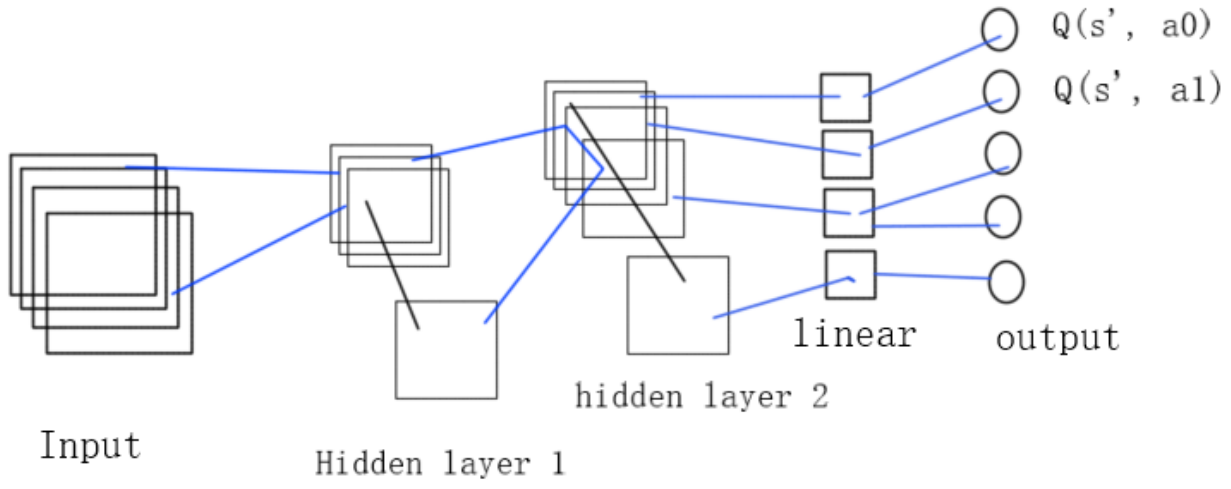


Fig 1. Diagram of our Deep Q-Learning Networkl

Introduction

Project Idea

The main idea behind the project was inspired by Deep Mind's implementation of deep reinforcement learning on Atari Game (Mnih et al., 2013). The implementation of deep reinforcement learning for this project was applied to the game of Tetris.

Various different approaches have been undertaken to solve Tetris using deep neural networks which have been observed to perform with astounding results for applications where little to no information was provided during preliminary set up conditions. Tetris requires strategy and seeing as neural networks can be used for strategy games it was a perfect fit for this project.

Although many different solutions for solving the game exist, with convolutional neural networks and other reinforcement learning approaches none have been able to create the perfect solution (Stevens and Pradhan 2016). The solution used for this project uses a learning network on q learning to find an optimal strategy for our simulation bot.

Deep Reinforcement learning

Main Concept :

The outline for setting up the algorithm for deep reinforcement learning was largely based on the approaches used in (Mnih et al., 2013) and (Mnih et al., 2015)).

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} \sum_{a' \in A} \pi(a'|s') Q_\pi(s',a')$$

$$= \mathbb{E}_\pi[r + \gamma Q_\pi(s_{t+1}, a_{t+1}) | S_t = s, A_t = a]$$

$$\downarrow \quad \pi(s_t) = \arg\max_{a'} Q(s_{t+1}, a')$$

$$Q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} \max_{a'} Q_*(s',a')$$

$$= \mathbb{E}_\pi[r + \gamma \max_{a'} Q_*(s_{t+1}, a') | S_t = s, A_t = a]$$

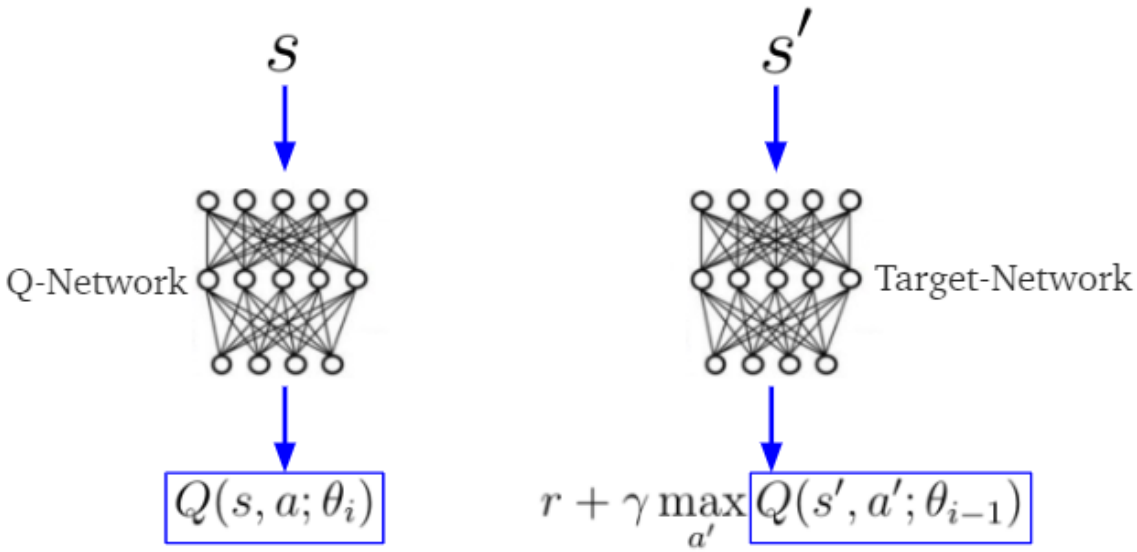
Figure : Simplified network function based on Bellman's Equation used in Q learning (Oppermann, 2018)

The above equations tell us the following:

1. Reward values are calculated on the basis of the states and action value pairs which are time dependent. π determines what policy is executed by the action a for a given state s at a time t. (Stevens and Pradhan 2016)
2. All future steps are taken with time increments of t+1

3. The calculation of the rewards, takes into consideration our discount factor, the current state-action pair. And since our reward calculation is the Q function that allows us to get the maximum possible reward, which in turn follows Bellman's equation, the formula was simplified to get the latter equation in the above figure.

Fig 2 . Target and Q networks: (Oppermann, 2018)



Within Deep Q learning we have the concept of target networks and Q networks which are functionally abstracted neural networks. Therefore we will have two networks one that is trained and another that isn't but whose values are updated at each iterated pass allowing us to further stabilise our networks. The article (Doshi, 2020) explained that since the values of Q networks were updated on the basis of the weights that we provide it can further make changes to the direction for calculation of target networks reward calculation. We would then constantly be looking for an ever changing value which is less than ideal for our learning network.

To minimize the distance of the above mentioned networks we use a loss function (squared) which can be achieved by any of the gradient descent programs. (Oppermann, 2018).

A high level diagram to assist in visualising how the network functions is as follows (Doshi, 2020)

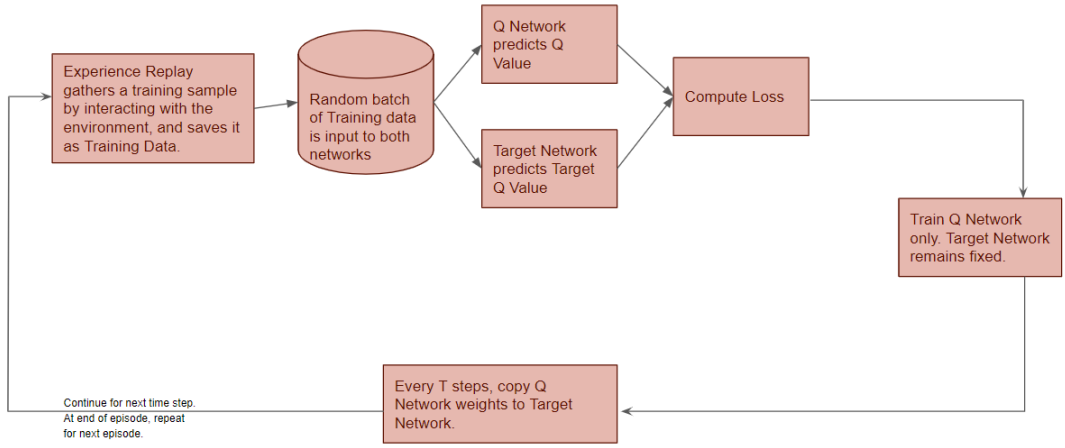


Fig 3 . High level network diagram (Doshi, 2020)

Method

Code Structure

The initial code was modularised for simplicity while setting up the project. It consists of the following sections :

1. Dqn agent : Here the conditions of our learning agent with hidden layers, number of neurons in networks, epsilon, training conditions and calculation for finding optimal state are defined.
2. Run train : This is where we actually train the models with parameters we've outlined in agent. Such as actual epsilon value, discount values, hidden layers, replay memory and so on.
3. Logs : This was the code where we outlined a custom tensorboard onto which the values generated in training would be written into to finally output the chart results.
4. Tetris : This was where the actual game of tetris that we would be rendering along with the rules for tetris were defined. It was sourced from a repository wherein the game rules were already predefined for us and has been referenced in the code.

5. Run evaluation : This is our custom function that has the ability to cross validate the models that we generate from our training. It takes in the hdf model file and compares it across the specified episode by randomly selecting 128 values to output what maximum score we can achieve. It has the ability to compare any number of model files stored in logs be it 1, 2, 3 or all.

Conditions

When setting up the structure for this system, it was decided to be a non greedy approach since many greedy approaches had already been covered and proven to not have the desired results.

In order to do so we had to remember the base algorithm this was set upon, i.e Q learning.

Q learning is an off-policy algorithm (Oppermann, 2018) where our final state is based on potential rewards. In this implementation we have rewards of +1 for every block placed, conversely for every game lost we deduct a point (-1) this becomes our penalty. And for every row cleared we have a reward set as width of board times the square of the rows cleared.

The code implemented has been tested out for multiple discount epsilon values which is set at 0.99, this will ensure that we have the opportunity to choose a best decision since this discount value determines the worth of the potential reward over the current one for the AI. The penalty here is set simply so that it's a non zero value to signal that it’s a “bad” move, encouraging the AI to go for an increased length of game regardless of the positive rewards amassed.

It is understood here that results can vary and not be optimal since a greedier choice will be adopted

Game Working

During the initial round of training, ai takes random actions from a mix back of seven (number of tetrominoes). Each state corresponds to a position action eg: left, turn clockwise etc and position associated with the largest value is chosen. As mentioned earlier

for some rounds best actions can be taken as all combinations of permutations that are probable. This is then fed to the network and finally the highest score is given as output.

Training was done with the following 4 attributes : height, rows cleared, number of holes and bumpiness as outlined in (Stevens and Pradhan 2016).

The very first figure of the report showcases the design of the deep neural network used in the project. There are two hidden layers each with 32 neurons and have an activation function of ReLU (rectified linear activation function) - which is especially used since it doesn't fire all neurons concurrently when training. The last layer is with a linear activation function. We use Mean Squared Error (mse) to calculate our loss values and a dynamic set for epsilon with a range between 1 to 0, allowing for an explorative model. We also add a discount value to this epsilon as discussed in our conditions which is triggered when episodes are ¾ th of the way done.

Results

Training results:

For training the network was run for 2000 episodes, replay size was 25000 with a sample of 512 episodes taken at random. The graphs were generated in tensorboard as follows

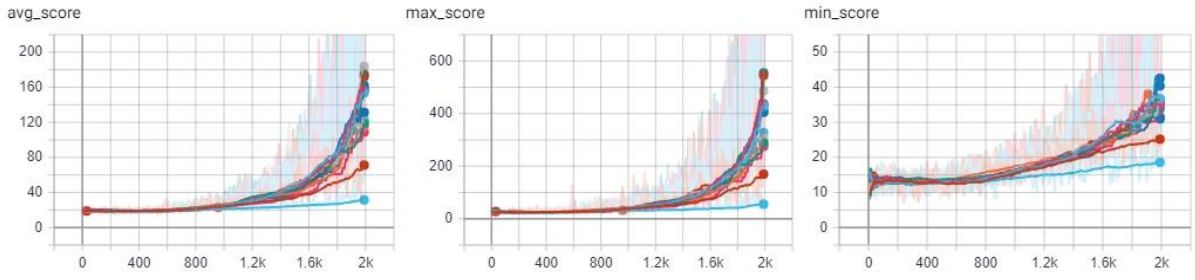


Fig 4 . Average , minimum and maximum scores for 2k episodes per training over time

Evaluating / testing results :

For evaluation purposes the pretrained models were tested on an epoch of 1 for 512 random samples and a comparison graph of the first vs the very last trained networks to find the minimum, maximum and average score values whose results are as follows:

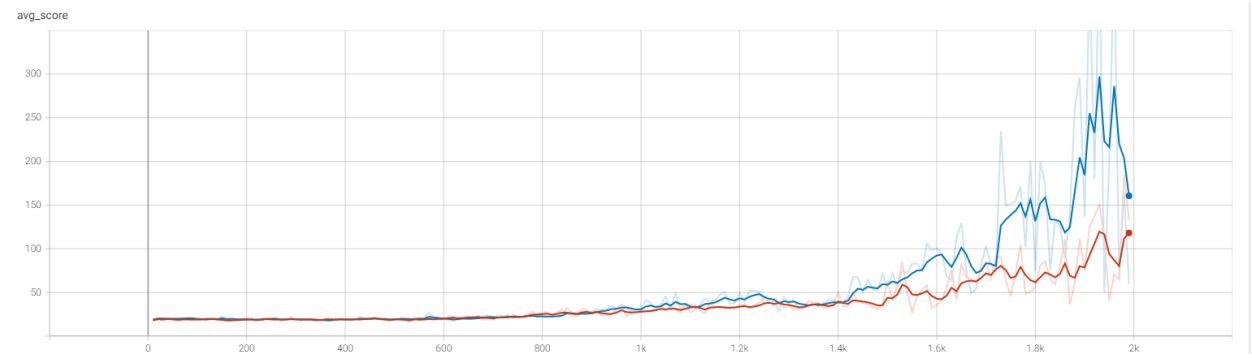


Fig 5.a: Average Score after 2000 episodes

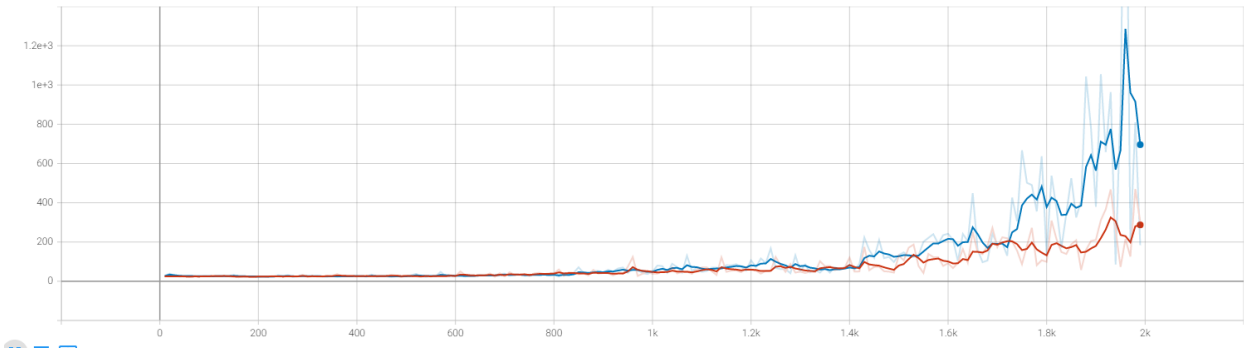


Fig 5.b: Maximum Score after 2000 episodes

Min score:

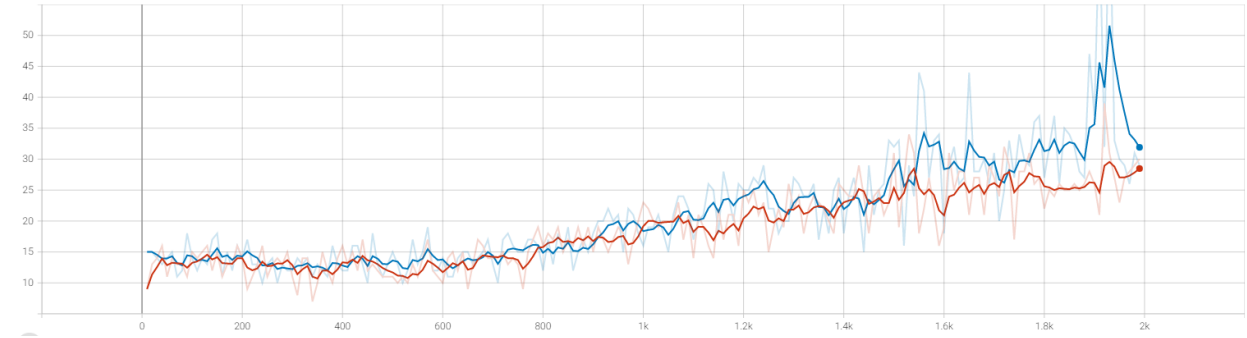


Fig 5.c: Minimum Score after 2000 episodes

Note: In figures:

Red line represents: The very first training

Blue line represents:The very last one

Sr. No	Graph results for scores	Values caused due to discount value 0.99
1	Average score (Ref Fig 1.a)	132 (red) VS 1044 (blue)
2	Maximum score (Ref Fig 1.b)	304 (red) VS 4293 (blue)
3	Minimum Score (Ref Fig 1.c)	30 (red) VS 42 (blue)

From the above graphs and values in charts shown to us on Tensorboard we can easily tell that the model was definitely learning to play for a longer game as well as a higher score. The learning is still quite slow but is showing improvements over time.

Conclusion

This project allowed for a deeper understanding of Deep Q networks in a classic game of Tetris. We can tell from the results that our modelled AI does actually learn, even though it might at times give us random results due to our set conditions. Some further advancements to this would be to have the AI play indefinitely. It would also be interesting to see how it fares against other modified versions of Tetris such as Hatetris (HATETRIS @ Things Of Interest, 2010). Furthermore I would have liked to play around with the hyperparameters and actions set had time allowed for it to see the variety of results that could have been generated to compare even lesser or improved score values.

Special thanks to Dr. Kean Lee Kang and Chris Acornley for their teaching and assistance during this project and module.

References

1. 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*.
2. Stevens, M. and Pradhan, S., 2016. Playing tetris with deep reinforcement learning.
3. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. nature, 518(7540), pp.529-533.

4. Oppermann, A., 2018. *Self Learning AI-Agents Part II: Deep Q-Learning*. [online] Medium. Available at: <https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning-b5ac60c3f47> [Accessed 8 October 2018].
5. Doshi, K., 2020. Reinforcement Learning Explained Visually (Part 5): Deep Q Networks, step-by-step. [online] Medium. Available at: <https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b> [Accessed 19 December 2020].
6. Qntm.org. 2010. *HATETRIS @ Things Of Interest*. [online] Available at: <https://qntm.org/hatetris> [Accessed 3 April 2010].