Acing simple games, an exploration of neural networks for reinforcement learning[1]

David Kleingeld

April 21, 2020

[1] A report for the Leiden Univesity Reinfocement Learning course

Contents

Introduction

Here

issue: speed and debugging, fast solution needed to be able to debug what is going on. issue: internet resources actually wrong, see: https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c (line 103 at the bottem. he quits if it succeeds once which could be by chance)

usefull: https://towardsdatascience.com/qrash-course-ii-from-q-learning-to-gradient-policy-actor-critic-in-12-minutes-8e8b47129c8c

Theory

Here we will give a short theoretical introduction to deep Q-networks (DQN). A DQN is capable of learning a diverse array of challanging tasks without hand coded domain specific knowledge. Recieving only the state of its envirement it returns one of the given possible actions, it then gets feedback in the form of a reward.

DQN is based on Q-learning or action-value learning. In Q-learning the Q function Equation 1 calculates the quality of a state-action combination. The outcome is then stored in the Q table and can be used to create more Q values. When an action is required the action with the highest Q value in the table is performed.

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of future value}} \Big) \quad (1)$$

Here Q^{new} is the new Q-value for the state s_t and action a_t. We calculate it by combining the old Q value with new information we learned as we transitioned to state s_{t+1} by action a_t. We weigh the combination with the learning rate α, the higher the learning rate the higher the impact of the new information. The new information consists of the reward r_t and Q value of the best action we can take from here[2]: $\max_a Q(s_{t+1}, a)$ We additionally weigh this by the discount factor γ which determines how importend future rewards are.

[2] as the best action is the action with the highest Q value we can add the maximum Q value corrosponding to the new state s_{t+1} we are now in.

Initially the Q table is empty and it will need training. The only way to fill the table is to performce random actions, using the reward to find the Q-value. A good strategy is to let the rate of random actions ϵ decrease throughout the training process. Starting at $\epsilon = 1$ and ending at $\epsilon = 0.1$.

This table based way of learning has only limited applications. For example if we where to apply Q-learning to the mountain car problem section we would immidiately run into a problem. The needed Q-table would need to be infinite[3]. While this can be solved by discretizing the input before the Q-learning algorithm there are many more complex problem where this is not feasable. One of these is our second problem, the atari game breakout, see **??**. For this we turn to DQN.

[3] ignoring the limited resolution of the `doubles` used to represent the state of the car

In DQN the optimal action-value is approximated by a neural network. When we need to perform an action we ask the neural network to make predictions for the Q-values for all the different possible actions. We then pick the action with the highest predicted Q-value.

Initially the neural network will behave like the empty table from Q-learning, not knowing the right Q value. However as we train the network it wil converge to the Q value. The network is trained with as input the state and action and as label the correct Q-value as calculated from Equation 1, getting $Q(s_{t+1}, a)$ and $Q(s_t, a)$ by asking the network to make a prediction for input $[s_{t+1}, a]$ and $[s_t, a]$ respectively.

There is a danger in using a neural network. The reinforcement learning can become unstable[4]. This is attributed to the *deadly triad*: function approximation, bootstrapping and off policy learning. The Q-function is approximated by the neural network based on features recognized in states states that should have a different Q-values may share the same features leading to loops in behavior. Bootstrapping, or basing the new Q-values partially on the old may exaggerate the problems introduced by the imperfect approximation. Finally DQN uses off policy learning it converges less well then on policy learning.

[4] John N. Tsitsiklis and Benjamin Van Roy. Analysis of temporal-difference learning with function approximation. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, page 1075–1081, Cambridge, MA, USA, 1996. MIT Press

Mnih, Koray Kavukcuoglu, and Silver[5] adress these instabilities using a replay buffer. By storing the states, actions and corrosponding reward the network can learn on results out of order. This breaks feedback loops solving the unstability described above. Another significant improvement made is infrequent weight updates. Here a seperate network is used for the predictions. This prevents the network from returning values based on newly learned behaviour while training causing those behaviours to be reinforced entering a feedback loop. Once in a while the network used for prediction is replaced with the current training network.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015

Mountain Car

Breakout

Conclusion

Appendices