Some nice title

Ville Toiviainen

ville.j.toiviainen@aalto.fi

Antti Partanen

antti.partanen@aalto.fi

Abstract

Our plan for this project was to explore neural networks, reinforcement learning and deep reinforcement learning algorithms. We set our focus on two algorithms: Policy Gradient and Deep Q-Networks (DQN), and their use to solve the OpenAI Gym environments. Initially, we selected Tennis on Atari 2600 as the environment to solve, but as we encountered difficulties that we originally didn't expect, we had to try out our algorithms on other OpenAI Gym's Atari 2600 environments.

While we didn't exactly reach our initial goal: solving the Atari Tennis environment, we did manage to create working algorithms of policy gradient and DQN. The algorithms are able to solve the CartPole-v0 environment from OpenAI Gym. In the results section, we discuss the performance of both algorithms.

I. Introduction

In recent year biggest development steps in reinforced learning have been related to solutions that use deep learning. The major step was taken by DeepMind team that used Deep Q-Networks with Atari games. For this reason, we wanted to experience with Deep Q-Networks in Atari environments. For comparison, we implemented Policy gradient, which has been an interesting topic of research lately as well.

I. Reinforced Learning

In reinforced learning, the algorithm learns the environment by interacting with it. It's a goal-driven way to learn - good actions are often rewarded and bad ones receive a penalty. No wonder that games are a very popular way to test reinforced learning. To learn the algorithm needs to evaluate its performance - how good decision it was to choose this action? In many cases, the algorithm takes multiple actions before it ends up in the terminal state. This forms multiple distinct action-state sequences that have their own rewards. Markov decision process (MDP) is an important property which means that these sequences can be used as a state. In Q-learning, another important property is that when iteration count approaches to infinity Q-function approaches the optimal Q*. This formalism is called Bellman equation. [1]

II. Q-Learning

In Q-Learning the model learns to estimate the optimal action-value function defined in function 1.

$$Q*(s,a) = \mathbf{E}_{s'}\left[r + \gamma \max_{a'} Q*(s',a')|s,a\right] \quad (1)$$

If the environment fulfills the requirements of MDP and Bellman equation eventually when $i \rightarrow \infty$, $Q \rightarrow Q*$. Q-Learning is an off-policy algorithm which in this case is mandatory since experience replay is used to boost the learning. Experience replay is explained in methods and experiments section. [1] [2]

III. Deep Learning

Deep Learning is an area of machine learning that uses deep neural networks for the learning task. "Neural networks have emerged as an important tool for classification. The recent vast research activities in neural classification have established that neural networks are a promising alternative to various conventional classification methods. The advantage of neural networks lies in the following theoretical aspects. First, neural networks are data-driven self- adaptive methods in that they can adjust themselves to the data without any explicit specification of functional or distributional form for the underlying model. Second, they are universal functional approximators in that neural networks can approximate any function with arbitrary accuracy [3]." This powerful way to approximate function is especially useful in reinforced learning where a system needs to learn to approximate very complex environment and finding the true optimal function can be too demanding time-wise and computationally.

IV. OpenAI Gym

OpenAI Gym is a toolkit for testing and training reinforcement learning algorithms. OpenAI Gym provides "environments" or test problems that have a generalized interface that allows people to write reinforcement algorithms to try to solve those problems. OpenAI Gym includes different environments, ranging from different games (in 2D and 3D) to various algorithmic problems.

One of the environment we used was Cart Pole. Cart Pole environment outputs four values: cart position, cart velocity, pole velocity and pole velocity at the top. These four values are given as an input to the neural network. Network's outputs are two values - Q-values for actions "go left" and "go right". The maximum number of actions to take is 200. Every action is rewarded with a reward of +1.

II. Methods

I. Deep Q-Networks

Deep Q-networks approximate Q-function in Q-learning with neural networks. Value from the action-value function is compared to the correct one with the equation 1. This way network eventually learns the optimal Q-function. Backpropagation can be done after every action or after a number of iterations. The later makes the learning process smoother. [1]

$$y_j = \begin{cases} r_j, \ if\, j+1 \, is\, terminal \\ r_j + \gamma \max_{a'} Q^*(\phi_{j+1}, a'; \theta^*) \end{cases} \tag{2}$$

$$Loss = (y_j - Q(\phi_j, a_j; \theta))^2 \tag{3}$$

In many environments, actions that are next to each other have a strong correlation. If the minibatches used in training would only contain correlated samples learning would be slow and even ineffective. Experience replay is a technique that solves this problem. Previous state-action pairs are stored in the memory and these pairs are used in the training randomly. This means that minibatches contain a lot of samples from different state-action pairs observed from the environment. This removes the strong correlation and makes the learning more efficient and smoother. Only requirement is that the learning algorithm is off-policy, which Q-learning is.

One dilemma in reinforced learning is exploration vs exploitation. In Q-learning, the model takes the action that has the highest reward unless there's stochasticity in the process. In DQN ϵ-greedy is used for that.

II. Policy Gradient

III. Experiments

In this section we cover different experiments that we did. With DQN we experienced with different network architectures, ϵ-greedy strategies and experience replay strategies.

I. Different Network architectures

We used feedforward neural networks as our deep learning method. Input and output sizes are determin by the environment - e.g Cart Pole has 2 input values and two output values. Everything between can be altered. We decided to limit our network depth two maximum layer size of two. That meant that we experienced with networks that contained one or two layers. Neuron count in layers the second architectural thing that we altered. Three experienced architectures are listed in the table 1.

Table 1: *Different network architectures used with DQN.*

	Neuron count in	
Layers	1st hidden layer	2nd hidden layer
1	24	
2	24	24
2	24	48

In all the experiments activation function was ReLU and loss was measured with mean squered error. Also all the hyperparameters where the same.

II. ϵ-greedy strategies

Two different linear ϵ-greedy strategies, equation 4 and 5, were tested. In all the cases number of steps (s) effected the value with decay (d) parameter. Maximum $epsilon_{max}$ value was 1 and minimum ϵ_{min} 0.01.

$$\epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{(-\frac{s}{d})} \tag{4}$$

$$max\{\epsilon_{min}, \frac{\epsilon_{max}}{max\{1, (sd)\}}\} \tag{5}$$

III. Experience replay strategies

Essential part of DQN is experience replay. The more observations we can store the better [1]. Unfortunately storing huge amount of observations introduces memory limits and using huge minibatch sizes means very heavy computations. For this reason we experienced with different Experience replay strategies to learn which could be good enough setup for this problem.

We started with dummy one that included just the last two observations. Second experience contained last 1000 observations and used 64 of those randomly. Third one stored last 100000 observations and used 64 of those randomly.

IV. Results

I. DQN with Cart Pole

Different network architectures had clearly performance differences. The most simple network that contained one hidden layer and 24 neurons was not able to learn the task in 1000 episodes. From the figure 1 we can see that it was close to reaching the average reward of 200 and it had strong upwards trend in the end. Learning only spiked upwards so the model avoided sudden drops in reward which happened more often with the other architectures.

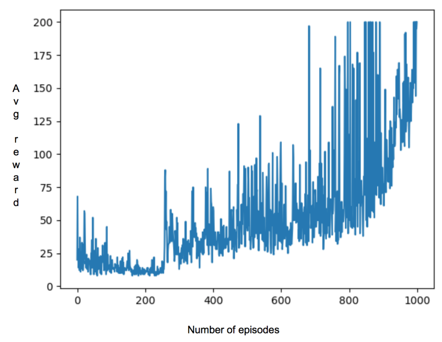

Figure 1: *Episode rewards for architecture with one hidden layer with 24 neurons.*

The two more complex networks learned the task (the average reward was over 195) under 350 episodes. From figures 2 and 3 we can see that network with 24 neurons in both layers learned the task more generally. It didn't have huge drops in the reward that the more complex network had even after it had multiple successful episodes that achieved the maximum reward.

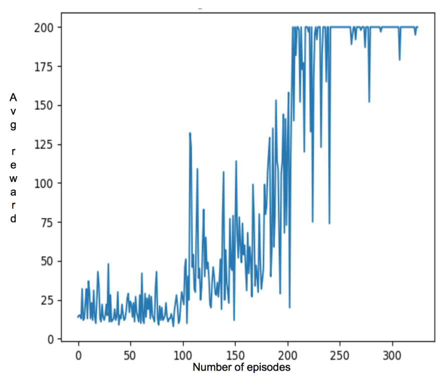

Figure 2: *Episode rewards for architecture with two hidden layers with 24 neurons.*

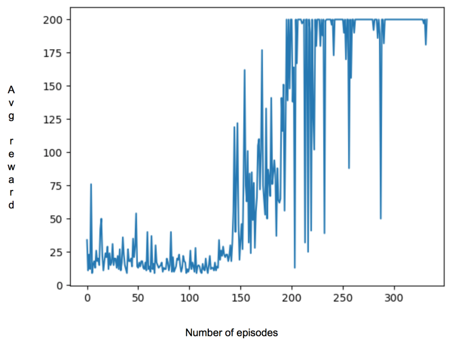

Figure 3: *Episode rewards for architecture with two hidden layers from which the first had 24 neurons and the second 48.*

Different ϵ-greedy strategies had the results we expected. When the algorithm was not able to explore enough it exploited a strategy that was a local minimum. As a result, the model was not able to reach maximum score. With really eager ϵ-greedy strategy, where *epsilon* value was over 0.1 still after 1000 episodes, the model was not able to learn optimal strategy. Reasons are obvious since there is very high probability 0.1 that action is taken randomly. Results for this epsilon we can see from the figure 4. Best ϵ-greedy strategy was achieved with the formula 5 that had the decay of 0.1.

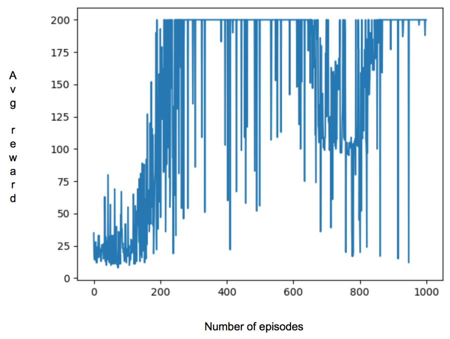

Figure 4: *Episode rewards for ϵ-greedy strategy that had ϵ value over 0.1 for 1000 episodes. Decay value was 0.01 for the epsilon strategy in the formula 5.*

Different experience replay strategies had also the results we expected. With a very small size of 2 model was not able to learn the task at all. With really high memory and batch size, it learned it really well. The only noticeable result was with the memory size of 1000. Intuitively we felt that 64 observations chosen randomly from 1000 would enough variation to break the strong correlation between close-by observations. This turned out to be not true like we can see from the figure 5. Based on the figure model hasn't learned the task. It can achieve maximum score, but only for a short time. Then reward drops to almost zero. It seems that it learns a strategy, it utilizes it until it learns a completely new one. This cycle repeats and at least with 1000 episodes it hasn't learned any steady strategy.

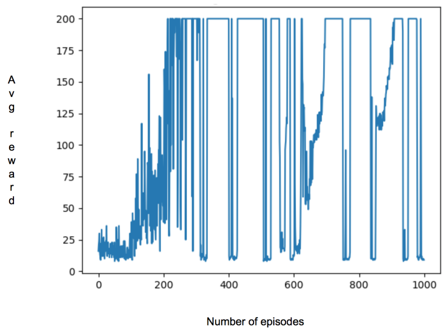

Figure 5: *Episode rewards when replay memory stored only the 1000 observations and had the batch size of 64.*

V. Discussion

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2016, book in preparation for MIT Press.

[3] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

VI. Appendix A: Dataset insights

VII. Appendix B: Pipeline