Project Proposal

The goal of this project is to derive an algorithm which when given environment information and rewards, is able to learn (online) a policy which maximize expected rewards. The aim is first implement and verify state of the art algorithm followed by ways and effects of tweaking the algorithm. More specifically, we are going to look at Lample and Chaplot's agent who plays FPS game[1].

1 Reinforcement Learning with partial & complex observations

Probabilistic state and Markov assumptions

We assume our agent is acting in a semi-realistic universe where the full state of the universe (which exists) is unknown, yet of part which is reflected by observations made by our agent. We model the universe (environment) using probabilistic states S and Markovian transitions T wrt actions as a POMDP. This assumption intuitively make sense in the context of video games.

Observation dimensionality

With video games (or videos in general for that matter), apart from extracting feature from static image pixels, we cannot neglect temporal information between frames.

Here we will try difference ways of reducing dimensionalities:

1. last n frames + neural-network (as described in the original Deep Q-Learn article from DeepMind[2] where $n = 4$).

2. last 1 frame + recurrent LSTM neural-network (presented and analyzed by Hausknecht and Stone[3]). This is also the method Lample and Chaplot used for their FPS game agent.

3. Maybe adding an object tracker (contour tracking for example) to the above two methods.

2 Ambitions

Weighted Reward Function

Lample and Chaplot's FPS agent has different networks designated for gathering weapon as well as confronting enemy, this works nicely for this particular game, but has difficulty generalizing to other games. After all, there are so many other more interesting games. We propose an reward model where we augment Q, V and R to be vectors and we fix a single weight vector as hyper-parameter for them. Whether this weight can be learned (from, say, last t episodes), I haven't worked it out...

Emotional greedy

Rather than a fixed ϵ used for ϵ-greedy used in all above mentioned related works, we will test a self adjusting ϵ which is inversely proportional to discounted past reward. While still avoiding over-fitting, this approach will hopefully converge faster.

High-Level Actions

The premature idea is as follows, first we need to calculate the expected return over a certain action somehow, Maybe: $\sum_s P(s)E(R(s,a))$

Now if we group actions sequences which has high expected value together and denote the sequence of actions as a new action (we can denote a^n is an action consists of n actions), so we theoretically should be able to search in action space $\{A^1, A^2, ..., A^k\}$. To approximate our search efficiently, we can use a fringe algorithm and eliminate actions sequences which turns out to have low return while new sequences are generated.

We can also add small positive factor for increase exploratory for performing higher level actions, hence encouraging even higher level action sequences to be produced on top of high level actions.

3 Reality

3.1 Training time and state space

If it takes weeks to train, I'll probably have to switch to a simpler game, but maybe not as simple as Super Mario.

4 References

[1] Lample, Guillaume, and Devendra Singh Chaplot. "Playing FPS Games with Deep Reinforcement Learning." AAAI. 2017.

[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.

[3] Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." CoRR, abs/1507.06527 (2015).