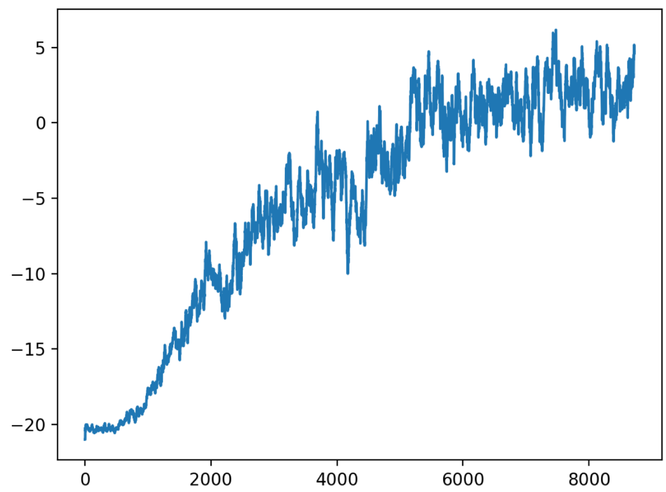
HW4

- HW4-1 Policy Gradient
- Describe your Policy Gradient model (1%)

  Preprocess observation成一個80*80的vector，用兩層Fully connected neural network，hidden layer數為256，第一層activation function為Relu，第二層activation function為softmax，輸出3維action的機率。每回合episode結束更新一次參數，每個step的reward會對未來的reward乘上gamma視為對未來reward影響的降低，目標為對大化action所得的reward的期望值。實作上用網路output p和實際sample出來的action還有discounted reward計算loss和gradient，Optimizer用RMSPropOptimizer，learning rate=1e-3，decay=0.99。
- Plot the learning curve to show the performance of your Policy Gradient on Pong (1%)



X-axis: number of time steps

Y-axis: average reward in last 30 episodes

Training 8720 episodes

- Describe your tips for improvement (1%)

  用variance reduction，根據

$$\mathrm{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]$$

$$\mathrm{Var} = E_{\tau\sim\pi_\theta(\tau)}[(\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]^2}_{\substack{\text{this bit is just } E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\ \text{(baselines are unbiased in expectation)}}}$$

$$\frac{d\mathrm{Var}}{db} = \frac{d}{db} E[g(\tau)^2 (r(\tau) - b)^2] = \frac{d}{db}\left(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right)$$

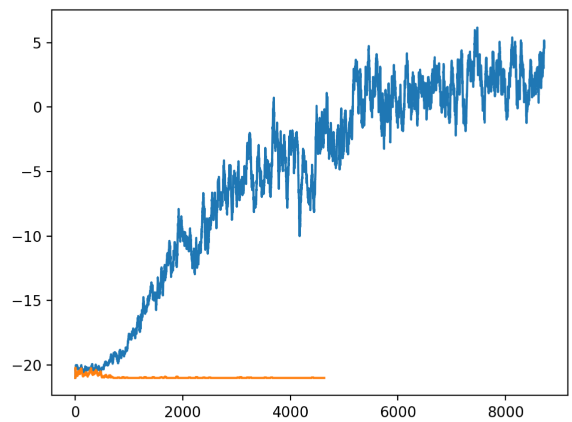
$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]} \longleftarrow$$

This is just expected reward, but weighted by gradient magnitudes!

計算出b，再將每回合reward減去b值，得到新的gradient。

- Learning curve (1%)



- Compare to the vallina policy gradient (1%)

  在其他參數都不變的情況下，Variance reduction像上圖橘色線一樣Train不起來，可能是減掉baseline後這項的range大幅改變，不適用原來的learning rate或optimizer，需要再進行實驗。

- HW4-2 Deep Q Learning
  - Describe your DQN model (1%)

    Model:

    input為(84, 84, 4)的資料，由四個frames所組成，先進入三層的conv層，第一層filter為[8, 8, 4, 32]，stride為[1, 4, 4, 1]；第二層filter為[4, 4, 32, 64]，stride為[1, 2, 2, 1]；第三層filter為[3, 3, 64, 64]，stride為[1, 1, 1, 1]1，VALID皆設為SAME，讓卷積可以停在圖片的邊緣。

    接著，在經Flatten，通過兩層Dense，各為512個neuron與action個數的neuron，判斷每個action的Q-value。

Hyper-parameters:

Replay Memory Size 10000

Perform Update Current Network Step 4

Perform Update Target Network Step 1000

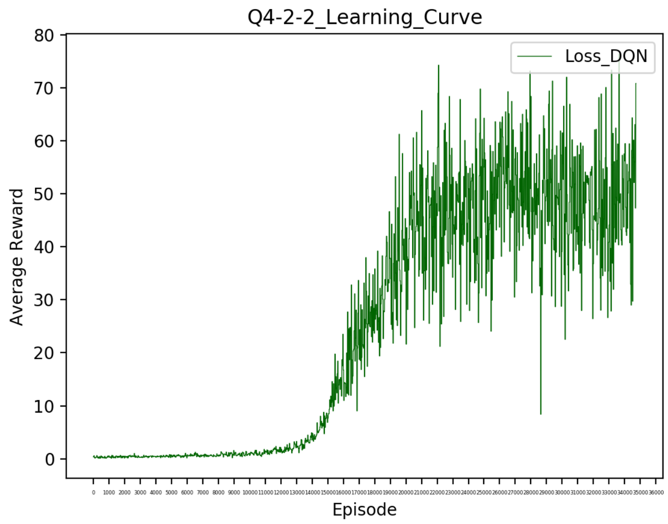
Learning Rate 1.5e-4

Batch Size 32

Discount gamma 0.99

RMSPropOptimizer(momentum=0, epsilon= 1e-8, decay=0.99)

- Plot the learning curve to show the performance of your Deep Q Learning on Breakout (1%)



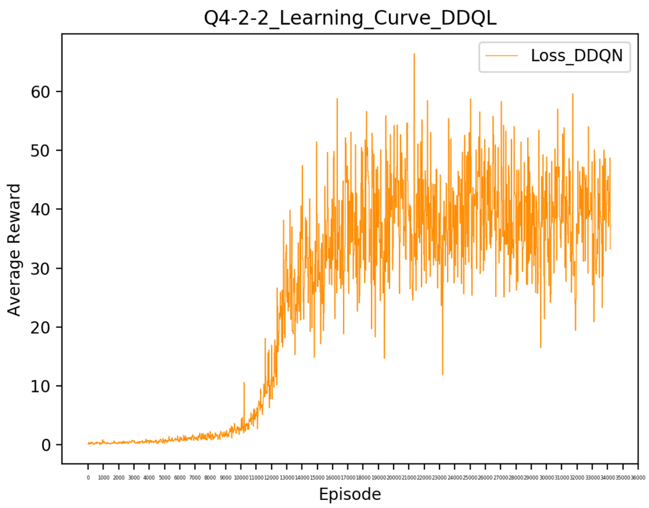
- Describe your tips for improvement (1%)

使用Double Deep Q Learning

因為在Deep Q Learning中，可能會高估了Q-value，而因此會傾向於選擇被高估的action，對後續的選擇也會造成影響。原來的loss function為比較$Q(s_t, a_t)$與$r_t + \max_a Q(s_{t+1}, a)$，而在Double Q Learning中，以比較$Q(s_t, a_t)$與$r_t + Q'(s_{t+1}, arg\ \max_a Q(s_{t+1}, a))$，由on-line Q-function選出action，再算出此action在Target net上的Q-value來當作最後的值，

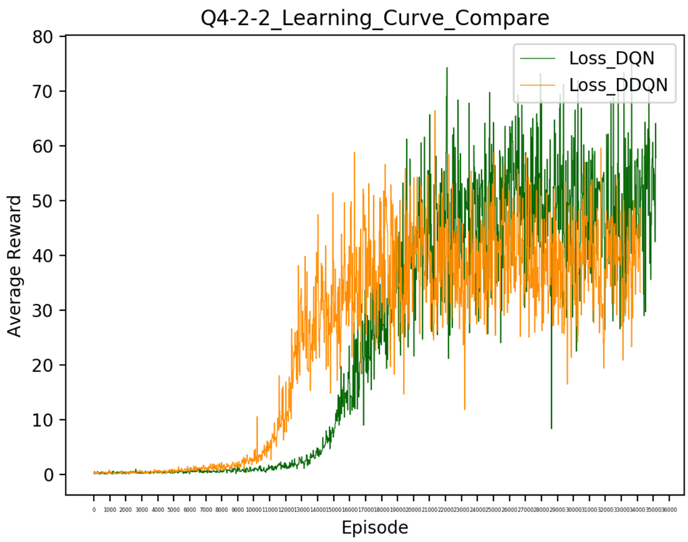
用以避免選擇到高估的action。

- Learning curve (1%)



- Compare to origin Deep Q Learning(1%)

下圖為比較DQL與DDQL。



由上圖可以發現，使用了Double Deep Q Learning之後，收斂的速度會較快，但是此次在訓練時，發現其實成效與Deep Q Learning差不了多少，似乎需再使用一些額外的tips才會達到更好的效果。

- HW4-3 Actor-Critic
- Describe your actor-critic model on Pong and Breakout (2%)

環境處理用到openai的env wrapper有：

NoopResetEnv(一開始隨機做0~30次Noop), MaxAndSkipEnv(重複4次一樣的動作), EpisodicLifeEnv(死掉時done設為1，但是會等到真正的episode結束才會reset), WarpFrame(resize

成84*84), ClipRewardEnv(把reward clip成{-1, 0, 1}), VecFrameStack(一次執行多個synchronized env，並且會把最近的4個frame疊起來當作observation)

Frame preprocessing：只有把frame normalize到[0, 1]

Model：

3 layers conv (n32k8s4, n64k4s2, n64k3s1), 1 layer fc(512), [action=fc(#action), value=fc(1)]

Hyper parameters：

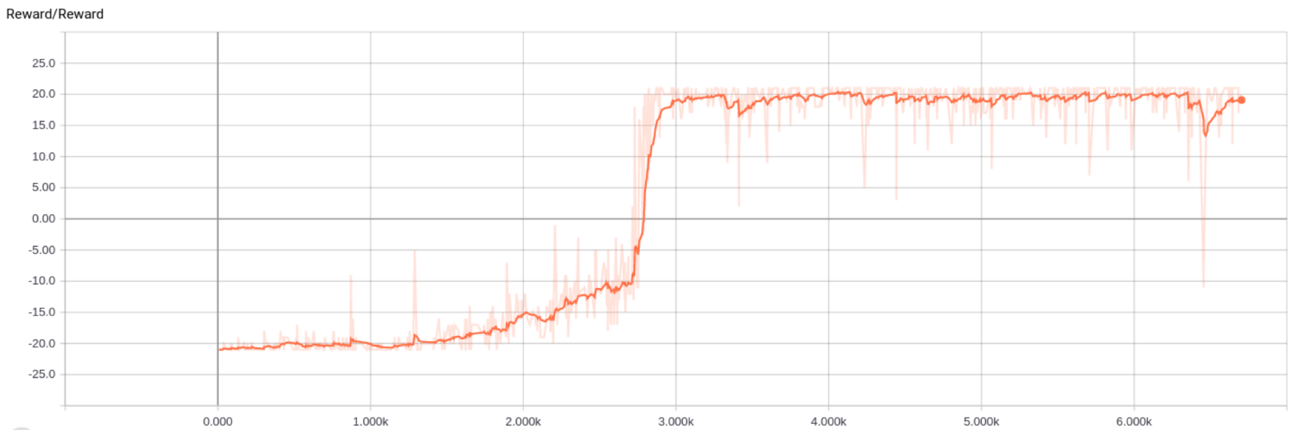
update_interval=5, entropy_regularizer=1e-2, learning_rate=7e-4, rmsprop_epsilon=1e-5, rmsprop_decay=0.99, max_gradient_norm=0.5, discount_gamma=0.99

整個訓練架構為A2C，因為只開一個env訓練過慢，故以下實驗皆使用4個thread(env)

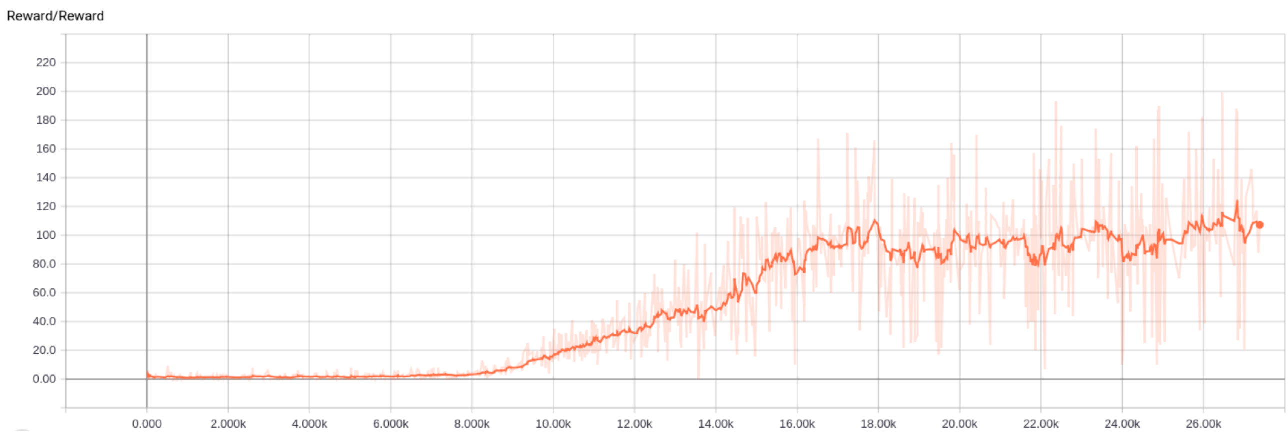
- Plot the learning curve and compare with 4-1 and 4-2 to show the performance of your actor-critic model on Pong & Breakout (2%)

Pong：大約只需要3000episode，即可以完全打敗電腦。

Breakout：大概會收斂在100分附近(clipped reward)，實際的reward可以把env render出來觀察大約會在350~400附近。



Pong (x軸為episode)



Breakout (x軸為episode, 5條命, clipped)

- Reproduce 1 improvement method of actor-critic (Allow any resource)

  Describe the method (1%)

  Improvement採用ACKTR (Actor Critic using Kronecker-Factored Trust Region)

  https://github.com/gd-zhang/ACKTR

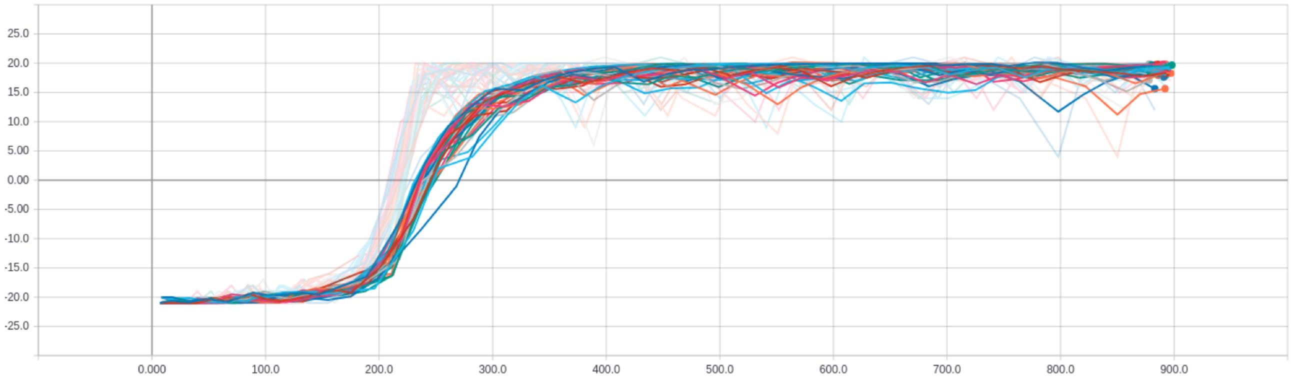
  ACKTR結合了3種不同的方法，分別為actor-critic、trust region optimization (training過程更穩定)、distributed Kronecker factorization (增加sample efficiency)。

  ACKTR會比A2C來的有效率的其中一個原因是，在更新參數的時候採用的是natural gradient，來避免新的policy的行為跟舊的差異過大(KL divergence)，進而避免performance collapse。

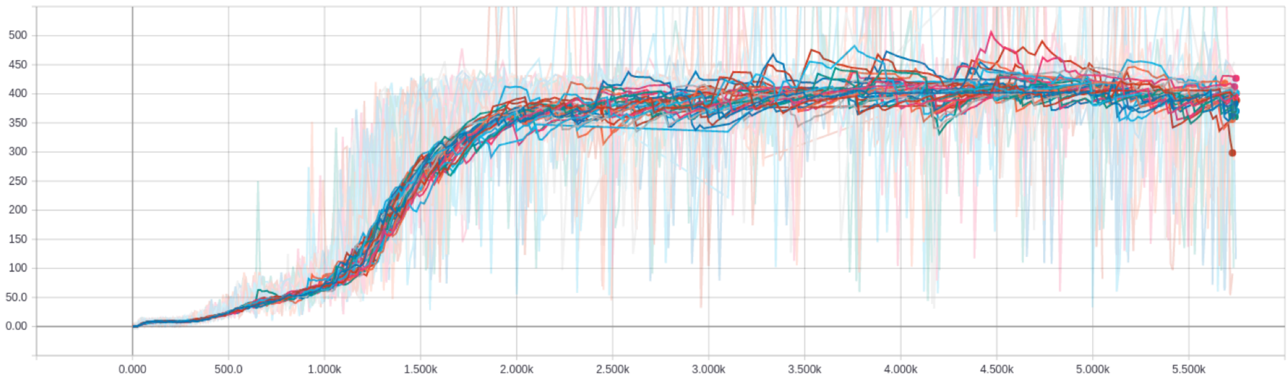
- Plot the learning curve and compare with 4-1 and 4-2, 4-3 to show the performance of your improvement (1%)

  Pong：跟上圖比較，可以觀察到打敗電腦所需的episode大幅降低

  Breakout：大約3000episode就可以達到跟上圖一樣的performance(unclipped)



Pong



Breakout (5條命, unclipped)