# DRL Model

GABRIEL PATTERSON

UDACITY

# DRL MODEL IMPLEMENTATION

## Vanilla DRL network implementation

The goal of this DRL model is to approximate the action value function Q(s,a) by minimising the squared distance of the model predicted value vs it equivalent Bellman Equation

The model's architecture for approximating the Q(s,a) is the one described by the Google Deep Mind research paper. In pseudo code it performs the following

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
**For** $t = 1$,T **do**
With probability $\varepsilon$ select a random action $a_t$
otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters $\theta$
Every $C$ steps reset $\hat{Q} = Q$
**End For**

.The model used for solving the banana collector game is a fully connected deep neural network with 2 hidden layers each containing 64 hidden nodes. This is the model that approximate the action value function Q(s,a) that guides the agent. Its output is a vector describing the respective "weight" the model assign to a given action. The model also has the following hyper parameters.

BUFFER_SIZE = int(1e5) # replay buffer size

BATCH_SIZE = 64 # minibatch size

GAMMA = 0.99 # discount factor

TAU = 1e-3 # for soft update of target parameters

LR = 5e-4 # learning rate

UPDATE_EVERY = 4 # how often to update the network

These hyper parameters are set at inception and are not dependent on data.

The model also uses a geometrically decreasing epsilon so that the epsilon greedy policy of selecting action is sure to converge

## Prioritized experience replay

When the model is updated using the loss function described in the previous section, each experience stored has an equal change of being selected. An alternative would be to select experiences to use for training according to their score on the cost function we are trying to optimise. Basically as described in the course these experiences would have a higher probability of being selected for training. This approach has been known to be more successful in many cases than the standard approach used in the vanilla model described in the last section. It is the simplest of all modifications suggested in the course and has the advantage of being not very much more computationally intensive than the vanilla approach so it has a good cost/benefit.

## Run more episodes

The model was run for 300 episodes until he reaches an average score of 13 over 100 consecutive episodes.  A better model could be obtained by running the model for many more episodes until no improvements are possible i.e. the average score doesn't change or only very marginally. This has the drawback of using more computing ressources.

## Rainbow (Kitchen sink approach)

The most powerful approach seems to be to combine many improvements together since it appears these improvements have synergetic effects between each other. Google Deep Mind has tested combining 6 well known improvements. Results indicates that it is the optimal improvement strategy for a DRL network. The drawback is that implementing all these changes is ressources intensive and technically difficult.