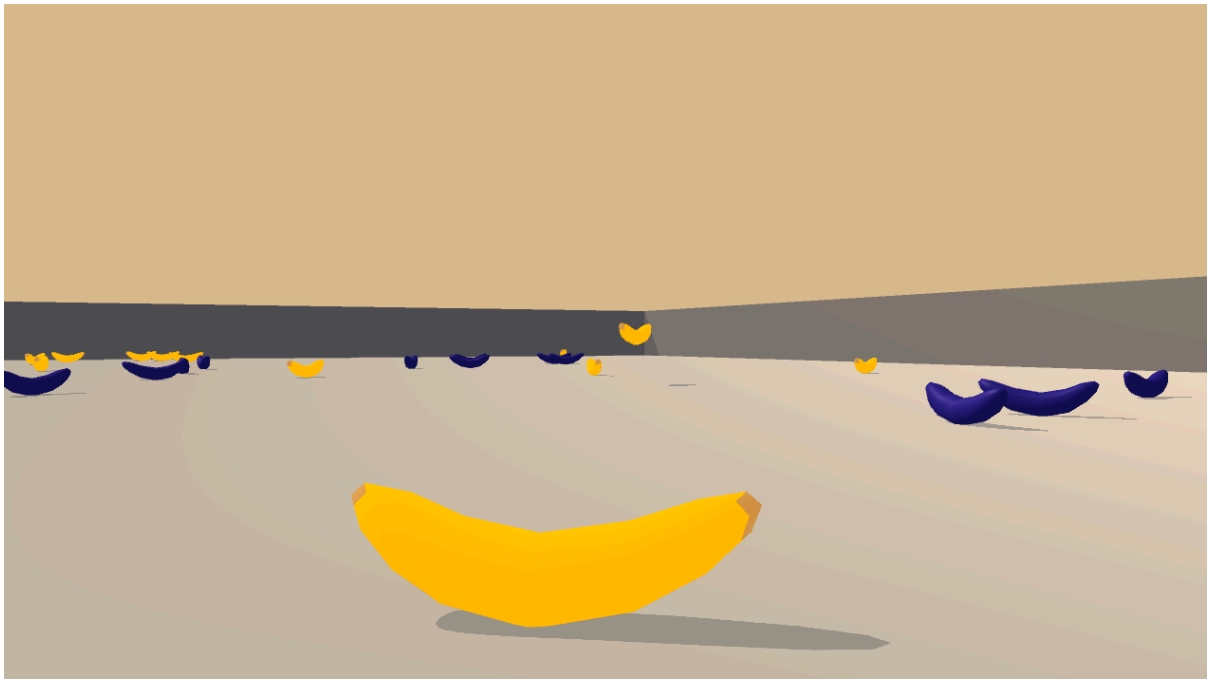
Udacity – Navigation DQN – Report 170319
Rens ter Weijde

In the navigation project we used Deep Q Learning to find the optimal policy. The environment used was from Unity, the corresponding task was collecting bananas (+1 reward for yellow ones, -1 reward for blue ones). Specifics of the environment: a continuous state space of 37 dimensions and 4 possible actions (left, right, forward, backward).



Google Images: the environment we were training the agent for.

On the algorithm: made famous by implementations on Atari games, DQN combines the idea of Q learning (where Q values represent the value of a certain actions in a certain states) with deep learning. Deep learning is in essence pattern recognition where layers of neurons can embody non-linear functions and learn through backpropagation. In essence, you could say that Deep Learning is a function approximation technique. The part that learns is embodied in the weights (*theta* in the model).

DQN is further a form of TD learning, which in contract to Monte Carlo methods doesn't need to complete episodes to learn. What is particularly interesting is the fact that it models the delta between the expected value and the target value; in this sense it becomes a bit like supervised learning which allows for training. The target network in this case creates more stability when training.

Note that the 'brain' of the agent is technically comprised of a small NN in this case, with only a few layers and 64 neurons per layer. This network is surprisingly simple given the inherent complexity of the task. Deepdive: the network is in the end trained through a loss function and backpropagation.

On the tweaks made to the algorithm: in the code I used a *Replay Buffer*. This is interesting from a psychological point of view and I personally compare it to daydreaming; it allows to

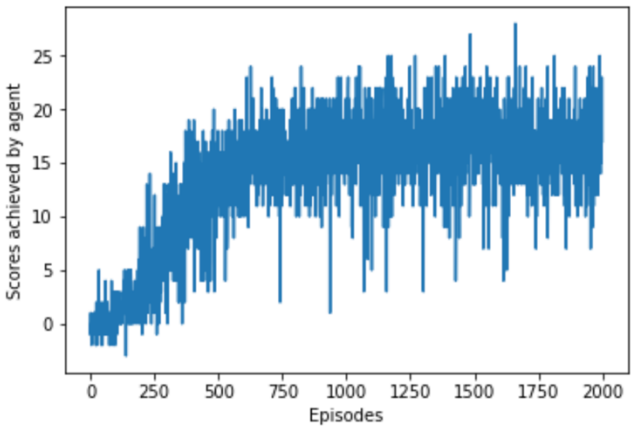
'decorrelate' input and reward, by learning from batches stored in the memory that were seen as relevant (rewarding or not). This is probably a bit similar to how humans/rhodents relive their experiences, and specific research links thalamic activity and dopamine spykes to this kind of activity.

In addition, I used the *Double-DQN:* this algorithm is a tweak where one network chooses the best action and the other evaluates it (David Silver et al. 2015). The purpose of this is mostly so the Q-learning algorithm does not overestimate action values.

On the hyperparameters used: most of the hyperparams used where quite normal in terms of range. Gamma 0.99, Epsilon decay 0.995, number of episodes 2000 (1000 was enough).

On the rewards: with a random setup (no longer in the notebook, as I got errors combining them for some reason) the average reward was 0. Now the reward is 16-17, which is a significant improvement. A training graph is shown below, as well as the result of every 100th episode.

```
Episode 100     Average Score: 0.14
Episode 200     Average Score: 1.90
Episode 300     Average Score: 5.41
Episode 400     Average Score: 9.48
Episode 500     Average Score: 11.89
Episode 600     Average Score: 14.04
Episode 700     Average Score: 15.64
Episode 800     Average Score: 15.90
Episode 900     Average Score: 16.03
Episode 1000    Average Score: 16.52
Episode 1100    Average Score: 16.21
Episode 1200    Average Score: 17.14
Episode 1300    Average Score: 16.72
Episode 1400    Average Score: 17.12
Episode 1500    Average Score: 17.19
Episode 1600    Average Score: 16.92
Episode 1700    Average Score: 16.74
Episode 1800    Average Score: 16.15
Episode 1900    Average Score: 16.41
Episode 2000    Average Score: 16.60
```



Ideas for improvement: most of the tweaks I could come up with (experience replay, double QN) are already in here. In later training I know we will use actor-critic models that can be more effective. Also, some form of early stopping could be useful – as I only needed ~1000 episodes, but chose 2000. A smart form of hyperparameter search (as in AWS SageMaker) could also be helpful, as most hyperparams are just set to a 'normal' range right now.