**Question 2**: Discuss your agent briefly, using the following questions as a guide:

* What learning algorithm(s) did you try? What worked best for you?
* What was your final choice of hyperparameters (such as 𝛼α, 𝛾γ, 𝜖ϵ, etc.)?
* What neural network architecture did you use (if any)? Specify layers, sizes, activation functions, etc.

**Answer**:

* Because I have so little experience with this type of project, when setting it up I initially relied closely to the project suggestions and sample code. The learning algorithm that I used is the Deep Deterministic Policy Gradients (DDPG) because it was suggested for the project and it worked well. Another reason to use the DDPG algorithm is that the Q-learning agent does not have the ability to estimate value for unseen states. In order to deal with this problem, the Deep Q-Learning Neural Network eliminates the two-dimensional array by introducing a Neural Network. Each Deep Q Network leverages a Neural Network to estimate the Q-value function. Training on the network will be based on the Q-learning update equation, where the loss function for the network is defined as the Squared Error between target Q-value and the Q-value output from the network. The input for the network is the current value of the state, or action, or both, while the output is the corresponding Q-value for each of the actions or states. In this implementation I will use a dual set of neural networks, one for the actor and one for the critic. The actor will be used to tune the parameter theta for the policy function, in other words, provide the best action for a specific state. The critic will be used for evaluating the policy function estimated by the actor according to the temporal difference (TD) error. For the DDPG, a replay buffer and a target network will be employed. This paper by Lillicrap, is a good reference: <https://arxiv.org/pdf/1509.02971.pdf>

reference: <https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287>

[Human-level control through deep reinforcement learning https://courses.engr.illinois.edu/cs546/sp2018/Slides/Apr05\_Minh.pdf](https://courses.engr.illinois.edu/cs546/sp2018/Slides/Apr05_Minh.pdf)

I also used the critic and actor codes almost as provided, and really only changed the neural network structure in small ways. Many of the hyperparameters, remained unchanged.

The Actor model has two dense layers with 512 and 256 cells. On both layers, I used 12 regularizers, batch normalization, and reLU activation funciton. The final layer has a dense layer with four cells and a sigmoid activation function. Adam optimizer was suggested in the example code, and that worked well with a learning rate of 0.001.

The Critic model is similar to the Actor model and also has two dense layers with 512 and 256 cells and I again used batch normalization and reLU activation function.

*I want to credit this GitHub repo for inspiration regarding regularizers and initializers <https://github.com/cipher813/quadcop2>*

* My final choices for the hyperparameters that were used are :
  + Learning rate for the actor = 0.0001
  + Learning rate for the critic = 0.001
  + Tau: (soft target update rate): = .001 This value was suggested in the paper "Continuous Control with Deep Reinforcement Learning"
  + Gamma: = 0.99, I tried a value of .7, but the learning curve was getting flat after a few tries
  + Noise: I tried a linear decrease of the noise every episode, but keeping it had much better reward / episode scores for the last 200 episode, before made better result.
  + Replay Buffer size / batch size: 100000 & 64 -> as set before
* For the neural network architecture I used Deep Deterministic Policy Gradients (DDPG) for the task. DDPG is an algorigthm well suited for continuous action space reinforcement learning. For the Actor and Critic uses somewhat different architectures, because the actor only takes states as inputs:

The Neural Network Architecture parameters that I used are:

* + Actor :
    - Dense(units=300) + BatchNorm + L2 Regularisation + ReLu Activation
    - Dense(units=300) + BatchNorm + L2 Regularisation + ReLu Activation
    - Dense( RandomUniform Weight initialisation ) + Sigmoid Activation
    - The optimizer used is the Adam optimizer, set with a learning rate of 0.0001 which has been suggested by the paper "Continuous Control with Deep Reinforcement Learning"
    - Also, dropout layers (30% dropout) and batch normalization was added to reduce overfitting

- Critic model is made up differently - mainly because this is the model to calculate Q values dependent on states AND actions whereas the Actor only takes states as inputs. The architecture looks like the following (remember: we have actions and states as input):

- Crtic :

- Same as actor for the state inputs as actor

- For the Input: actions:

- Dense(units=300) + L2 Regularization + ReLu Activation

- Combining : Add with ReLu Activation

- Adam optimizer was suggested in the example code, and that worked well with a learning rate of 0.001.

"Continuous Control with Deep Reinforcement Learning"

- I tried dropouts of .3 and .5 to both the actor and critic networks, but it made the reward results worst in the last 200 out of 500 episodes than compared without the dropout.