CONTINUOS CONTROL PROJECT –
Javier Tausía Hoyal

*** This is the explanatory file Report.pdf that summarizes the strategy that has been followed solving the Continuos Control problem (Reacher and Crawler) for the Deep Reinforcement Learning Nanodegree ***

First, the description of the environment and how it can be downloaded is explained in the README.md file, so here we will focus on the model used and the philosphy in the selection of the parameters.

Regarding the model, 2 neural networks have been used for the actor and the critic with shape:

```
class Actor(nn.Module):
    """Actor (Policy) Model."""

    def __init__(self, state_size, action_size, seed, fc_units=256):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
            fc1_units (int): Number of nodes in first hidden layer
            fc2_units (int): Number of nodes in second hidden layer
        """
        super(Actor, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc_units)
        self.fc2 = nn.Linear(fc_units, action_size)
        self.reset_parameters()

    def reset_parameters(self):
        self.fc1.weight.data.uniform_(*hidden_init(self.fc1))
        self.fc2.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state):
        """Build an actor (policy) network that maps states -> actions."""
        x = F.relu(self.fc1(state))
        return F.tanh(self.fc2(x))
```

for the **actor** and:

```
class Critic(nn.Module):
    """Critic (Value) Model."""

    def __init__(self, state_size, action_size, seed, fcs1_units=256, fc2_units=256, fc3_units=128):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
            fcs1_units (int): Number of nodes in the first hidden layer
            fc2_units (int): Number of nodes in the second hidden layer
        """
        super(Critic, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fcs1 = nn.Linear(state_size, fcs1_units)
        self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
        self.fc3 = nn.Linear(fc2_units, fc3_units)
        self.fc4 = nn.Linear(fc3_units, 1)
        self.reset_parameters()

    def reset_parameters(self):
        self.fcs1.weight.data.uniform_(*hidden_init(self.fcs1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(*hidden_init(self.fc3))
        self.fc4.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state, action):
        """Build a critic (value) network that maps (state, action) pairs -> Q-values."""
        xs = F.leaky_relu(self.fcs1(state))
        x = torch.cat((xs, action), dim=1)
        x = F.leaky_relu(self.fc2(x))
        x = F.leaky_relu(self.fc3(x))
        return self.fc4(x)
```

for the **critic**.

with a very high numer of parameters. This model is very simple, and a more complex model could has been used, and although this model dind't work, we tought that complicating the model architecture will not be very useful in the improvement of learning.

Regarding the tune of the parameters, different combinations have been used, all leading to similar results at the end of the day. The final configuration of hyperparameters was:

```
BUFFER_SIZE = int(1e6)   # replay buffer size
BATCH_SIZE = 128         # minibatch size
GAMMA = 0.99             # discount factor
TAU = 1e-3               # for soft update of target parameters
LR_ACTOR = 1e-4          # learning rate of the actor
LR_CRITIC = 3e-4         # learning rate of the critic
WEIGHT_DECAY = 0.0001    # L2 weight decay
```

SUMMARY:

As a summary, simply say that I do not know why the code didn't work, and it will be very useful if some help is provided...