Learning Algorithm

The algorithm that was leveraged for this assignment was Multi-agent deep deterministic policy gradient. MADDPG is very similar to DDPG in that they both use an actor-critic technique to make training faster and more efficient. What's interesting about this implementation is that it combines the data/knowledge of each agent during the training process.

Let me explain, usually, the critic network (of one agent) leverages the actions and observations of what it experiences in order to make a guess of the reward the agent might experience – which is then used to train the policy (actor). With MADDPG, all agents can share their own actions and experiences with each other via their critic network. So, the actors are still trained on their respective critics, but, the actions that each actor takes, and the observations (states) that they all experience are shared with all the critics.

Hyperparameters

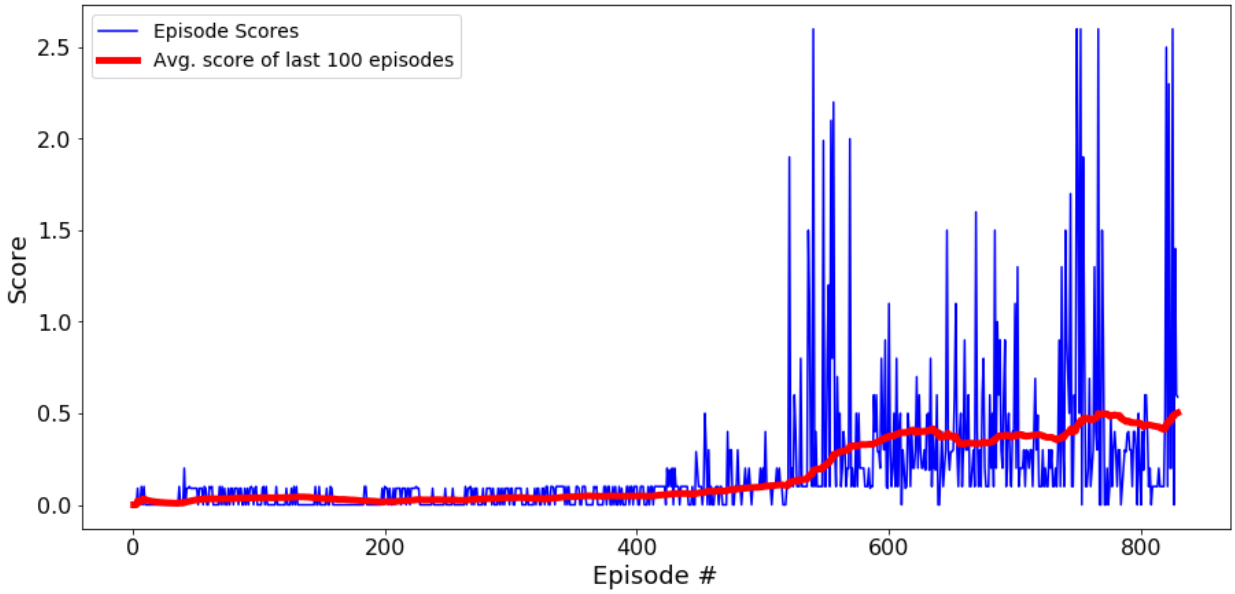
actor_lr = 3e-3
critic_lr = 4e-4
buffer_size = int(1e6)
batch_size = 64
gamma = 0.99
tau = 1e-3
noise_start = 0.9
noise_decay = 0.01
update_every = 1 #update every episode

Model Architecture

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

    def __init__(self, input_size, output_size, seed, fc1_units=256, fc2_units=256):
        """Initialize parameters and build actor model.
        Params
        ======
            input_size (int):  number of dimensions for input layer
            output_size (int): number of dimensions for output layer
            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(ActorModel, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(input_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, output_size)
        self.bn = nn.BatchNorm1d(fc1_units)
        self.reset_parameters()

    def reset_parameters(self):
        """Initialize weights with near zero values."""
        self.fc1.weight.data.uniform_(*hidden_init(self.fc1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state):
        """Build an actor network that maps states to actions."""
        if state.dim() == 1:
            state = torch.unsqueeze(state,0)
        x = F.relu(self.fc1(state))
        x = self.bn(x)
        x = F.relu(self.fc2(x))
        x = F.tanh(self.fc3(x))
        return x


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

    def __init__(self, input_size, seed, fc1_units=256, fc2_units=256):
        """Initialize parameters and build model.
        Params
        ======
            input_size (int): number of dimensions for input layer
            seed (int): random seed
            fc1_units (int): number of nodes in the first hidden layer
            fc2_units (int): number of nodes in the second hidden layer
        """
        super(CriticModel, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(input_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, 1)
        self.bn1 = nn.BatchNorm1d(fc1_units)
        self.reset_parameters()

    def reset_parameters(self):
        """Initialize weights with near zero values."""
        self.fc1.weight.data.uniform_(*hidden_init(self.fc1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, states, actions):
        """Build a critic network that maps (states, actions) pairs to Q-values."""
        xs = torch.cat((states, actions), dim=1)
        x = F.relu(self.fc1(xs))
        x = self.bn1(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = torch.sigmoid(x)
        return x
```

Plot of Rewards

Solves in 829 episodes.



Future Work

- Changing/tuning hyperparameters
- Implementing prioritized experiences