ECS 170: Learning to Play Pong

Ethan He

1 Problem Representation

1. Since we are using a neural network to replace the Q-table, it is easier to use image as state input. As it did in the `QLeaner` class:

```
self.features = nn.Sequential(
    nn.Conv2d(self.input_shape[0], 32, kernel_size=8, stride=4),
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=4, stride=2),
    nn.ReLU(),
    nn.Conv2d(64, 64, kernel_size=3, stride=1),
    nn.ReLU()
)

self.fc = nn.Sequential(
    nn.Linear(self.feature_size(), 512),
    nn.ReLU(),
    nn.Linear(512, self.num_actions)
)
```

2. The purpose of the neural network in Q-Learning is to replace the lookuo table. In such way, we can train a network for each action, where we use state as input to the netword and get $\widehat{Q}$ as output.

3. ϵ is the identifier for choosing an action. We generate a random number, if it is greater then ϵ, we do exploitation, so choose the best known action that the Q learner tell us; otherwise, we do exploration, then do random action.

4. See function `act` of `dpn.py`.

2 Making a Q-Learner Learn

The loss function is the square error formula. In the function, we get the current environment, where `batch_size` controls the sample size we fetch from `replay_buffer` (tensor). And from the random samples, we find the source to calculate the square error. The parameter `gamma`γ is to control the Q-learner's behavior: as γ get closer to 1, future rewards are given greater emphasis relative to the immediate rewards.

```
q_val = torch.gather(model(state.squeeze(1)), 1, action.unsqueeze(1)).
    squeeze(1) # y_i
q_val_target = target_model(next_state.squeeze(1)).max(1)[0]
expectations = reward + gamma * q_val_target * (1 - done)
loss = (q_val - Variable(expectations.data)).pow(2).mean()
```

3 Extend the Deep Q-Learner

See program `dqn.py`.

```
def sample(self, batch_size):
    # Randomly sampling data with specific batch size from the buffer
    state, action, reward, next_state, done = zip(*random.sample(self.
   buffer, batch_size))

    state = Variable(torch.FloatTensor(np.float32(state)))
    next_state = Variable(torch.FloatTensor(np.float32(next_state)))

    return state, action, reward, next_state, done
```

4 Learning to Play Pong

1. See program `run_dqn_pong.py` around line 90.

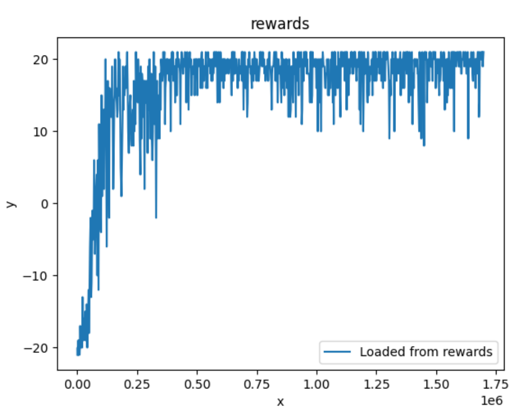
```
torch.save(model.state_dict(), model_file)
```

2. Use the `np.savetxt()` function to save `losses` and `all_rewards` in `.csv` files for use inquestion 4.

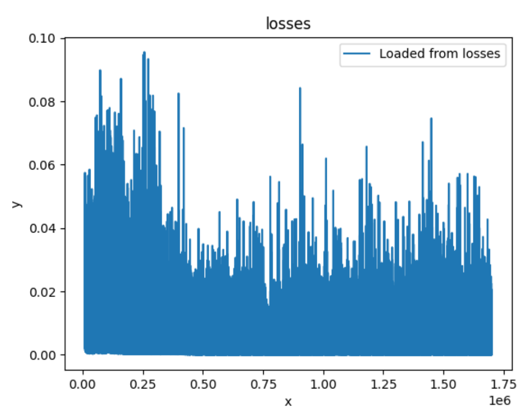
```
np.savetxt("rewards.csv", all_rewards, delimiter=",")
np.savetxt("losses.csv", losses, delimiter=",")
```

3. Use command `python3 run_dqn_pong.py` to train the model.

4. Plots.



(a) Rewards



(b) Losses

5 Bonus