Algorithm

I selected DDPG (Deep Deterministic Policy Gradient) [1] to solve this problem. The DDPG is the combination of Deterministic Policy Gradient (DPG) [2] and Deep Q-Network (DQN) [3]. The DDPG trains Actor and Critic models simultaneously. The Actor model ($\mu(s;\theta^{\mu})$) specifies the current policy by deterministically mapping states to an action. The Critic model ($Q(s,a;\theta^{Q})$) provides the value of the state-action combination and be learned by using Bellman equation. The Actor model is then learned with the policy gradient theorem given by the following [1, 2].

$$\nabla_{\theta^{\mu}} J = \mathrm{E}_{s_t}\left[\nabla_a Q(s,a;\theta^{Q})|_{s=s_t,a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s;\theta^{\mu})|_{s=s_t}\right]$$

The Actor model and the Critic model are given by a Deep Neural Network (DNN), and the Critic model is, in particular, trained with the DQN algorithm. The DDPG algorithm is given as follows.

DDPG Algorithm

Initialize Actor model (θ^{μ}) and Critic model (θ^{Q})
Copy the Actor/Critic models to the target Actor/Critic models ($\theta^{\mu'}/\theta^{Q'}$),respectively:
  $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$
Initialize Replay Buffer R
for episode = 1, M **do**
  Initialize random process $\mathcal{N}$ for action exploration
  Receive initial observation from environment
  for t = 1, termination of environment **do**
    Select action: $a_t = \mu(s_t;\theta^{\mu}) + \mathcal{N}_t$
    Execute action: $(s_{t+1}, r_t) = env(s_t, a_t)$
    Store transaction (s_t, a_t, r_t, s_{t+1}) in R
    Sample random batch transactions (N) from R
    Compute target value: $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1};\theta^{\mu'});\theta^{Q'})$
    Update Critic model by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i;\theta^{Q}))^2$
    Update Actor model using sampled policy gradient:
      $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a;\theta^{Q})|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s;\theta^{\mu})|_{s_i}$
    Soft-update target networks:
      $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
  end for
end for

Here, the question is what Actor/Critic models (DNN model) should be used. I started from a simple DNN. However the model did not learn at all at the beginning. Then the model was modified many times by referring the DDPG paper [1] and the slack channel discussions [4]. Finally, after many tries-

and-errors, and with a lot of luck, I found a model that is trained quickly and solve the environment. The following describes the details of these attempts. For all attempts, the 20 agent environment was used.

Attempt 1:

At the beginning, the Actor and Critic models were implemented with a simple multi-layer DNN. A linear dense network and a ReLU function are used in all layers except the output layer. Figure 1 shows the outline of the model. I tested models with different number of layers. I also tested different learning rates including the method where the learning rate was decreased periodically in the learning. Some other hyperparameters such as discount and OU-noise parameters (σ, θ) were also tweaked. However, none of these showed any sign of learning. Figure 2 shows an example of the score history (average score over 20 agents per episode) in those attempts. The score stayed around 0.1 even after 100 episodes. This would be due to the covariate shift in each layer which makes it difficult for network to learn efficiently.

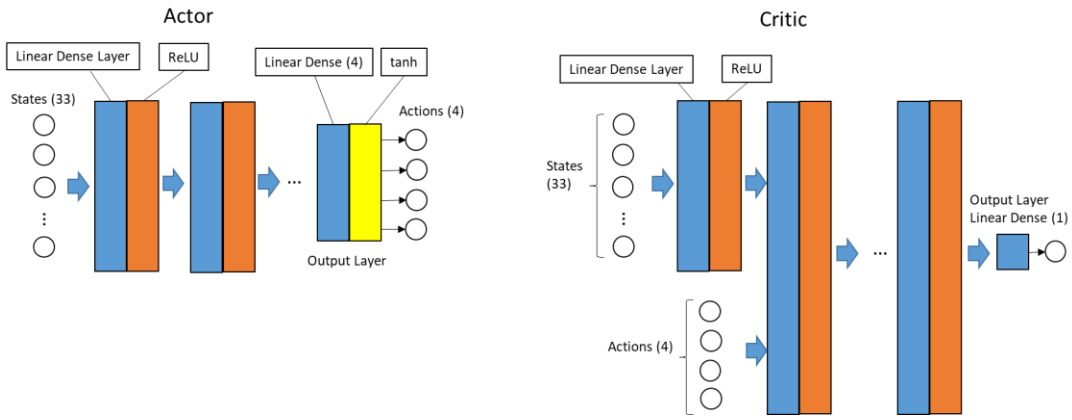

Figure 1: Actor/Critic models with only Linear-Dense and ReLU activation

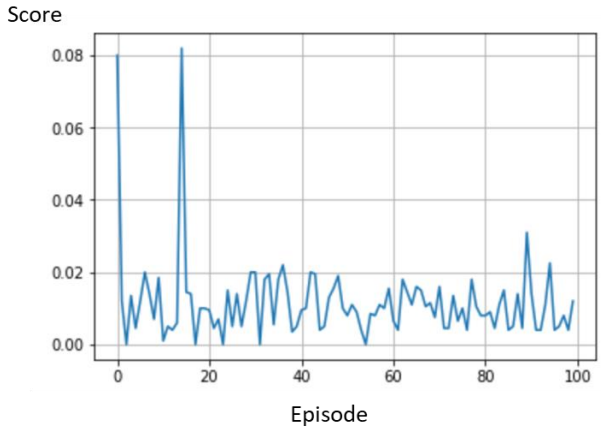

Figure 2: Typical score history of training with models having only Linear-Dense and ReLU

Attempt 2:

To handle the effect of covariant shift, I added Batch-Normalization in every layer including the input. Figure 3 shows the outline of the model. Then some sing of learning showed up in the score history. Figure 4 is an example of the score history for this attempt. The model started providing the score higher than 20 after 20 episodes. However the score largely varied episode by episode, and the model did not stably show the score of 30 or higher. In order to stabilize the learning, I tweaked hyperparameters such as number of layers, learning rate, and discount. None of these attempts however satisfied the criteria which is getting average score of +30 over 100 episodes.

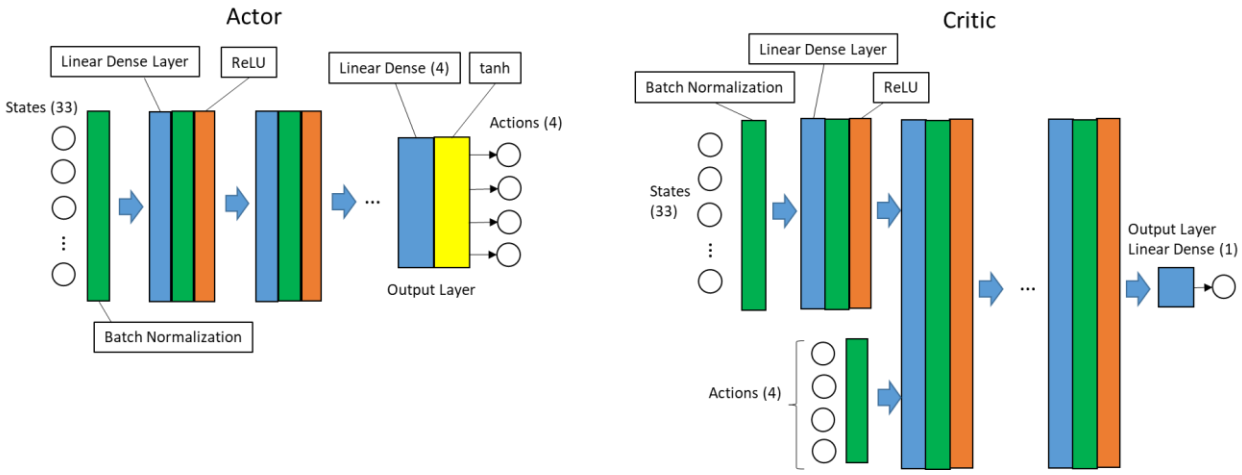

Figure 3: Action/Critic models with Batch-Normalization

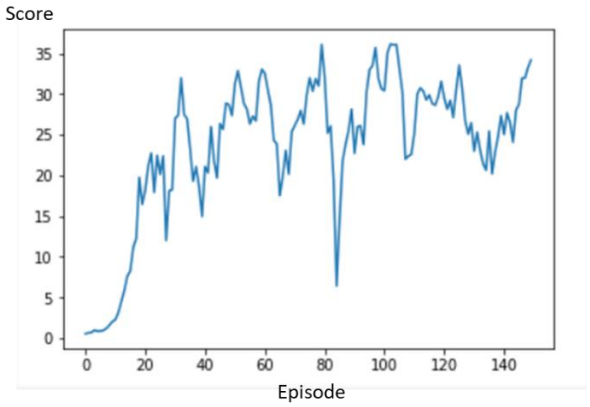

Figure 4: Score history of training with model hiving Batch-Normalization in each layer

Attempt 3:

Finally, referring the DDPG paper [1], I removed the Batch Normalization from the higher layers of the Critic model so that only layers before adding the action input have the Batch Normalization. Although I do not know why, this modification seemed to be critical. The score quickly increased to a high value like 35 and stayed there with a small variation after that. Figure 5 shows the score history for this attempt. The score reached 30 at episode 20 and did not get lower than that afterward except only one episode (episode 80). In fact, the average score over 100 episodes reached 30 for the average over episode 3 to episode 102, and did never get below 30 after that. Figure 6 shows the detail architecture of the Actor/Critic models used for this training, and Table 1 shows hyperparameters.

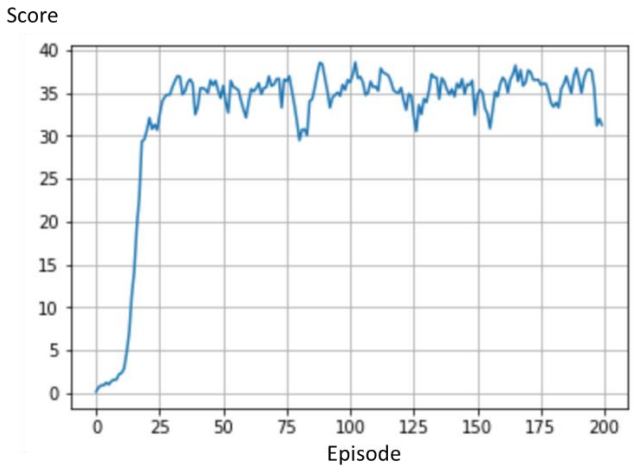

Figure 5: Score history of training with final model

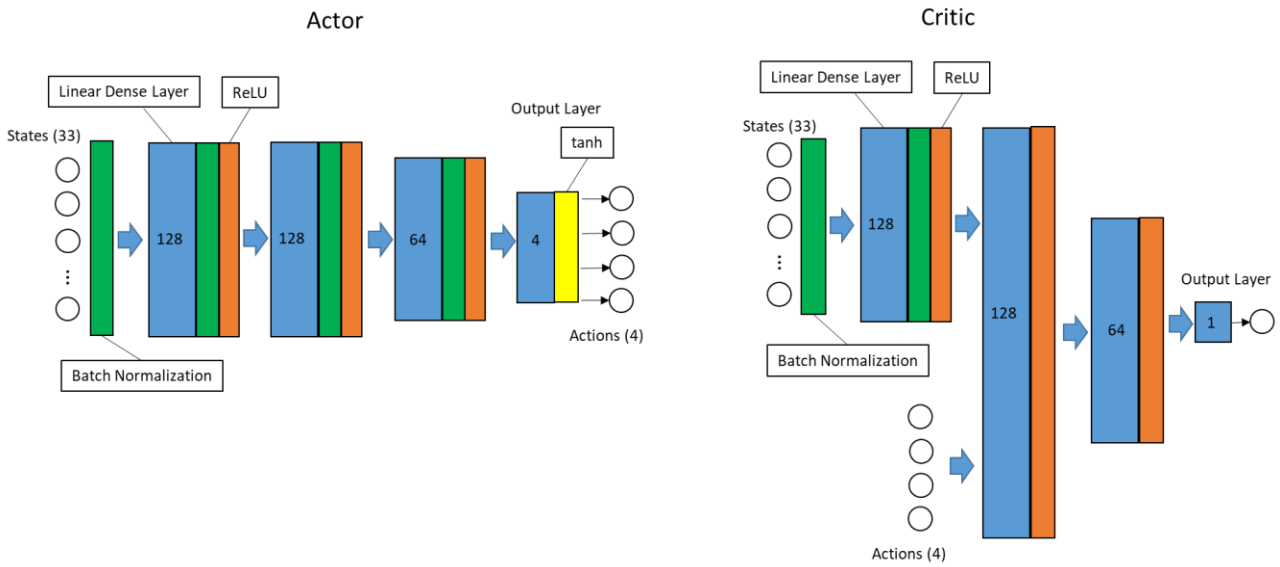

Figure 6: Final Actor/Critic models

Table 1: Hyperparameters

Replay Buffer Size	10,000
Batch Size	512
Discount	0.9
Actor Learning Rate	0.001
Critic Learning Rate	0.001
Learning Rate Reduction	No
Soft Update τ	0.001
OU Noise θ	0.15
OU Noise σ	0.01

Summary and Future Work

The attempts explained above showed me quite interesting insights about DDGP. In particular, it was reaffirmed that the Batch-Normalization significantly improve the efficiency of learning. However, at the same time, putting the Batch-Normalization in every layer resulted in unstable learning. Although the reason was unsure, removing the Batch-Normalization from the higher layers in the Critic model stabilized the learning, and resulted in a good Actor/Cretic models that leaned very quickly.

As future works, it is of great interest to try some other policy-based methods such as PPO. The model introduced above has almost optimal learning efficiency. Hence, in addition to the learning efficiency, it will be interesting to design a model that obtains higher score (like over 40) without sacrificing the learning efficiency.

It is still mysterious why removing Batch-Normalization from the higher layer in the Critic model drastically improved the stability of learning. Any suggestion or discussion?

References

[1] Timothy P. Lillicrap, *et al.*, "Continuous Control With Deep Reinforcement Learning", ICLR, 2016.

[2] Shilver, *et al.*, "Deterministic Policy Gradient Algorithm", ICML, 2014

[3] Hado van Hasselt, et al., "Deep Reinforcement Learning with Double Q-learning", AAAI on AI, 2016.

[4] Udacity Deep Reinforcement Learning Nanodegree, Slack channel #project-2_continuous