МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
ХЭРЭГЛЭЭНИЙ ШИНЖЛЭХ УХААН, ИНЖЕНЕРЧЛЭЛИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ

Батбаярын Бямбаням

DDPG бататгасан сургалтын үйлдлийн шуугианыг турших, харьцуулах

(Effect of action space noise for DDPG RL algorithm)

Мэдээллийн технологи (D061304)
Бакалаврын судалгааны ажил

Улаанбаатар

2021 оны 02 сар

МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
ХЭРЭГЛЭЭНИЙ ШИНЖЛЭХ УХААН, ИНЖЕНЕРЧЛЭЛИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ

DDPG бататгасан сургалтын үйлдлийн шуугианыг турших, харьцуулах

(Effect of action space noise for DDPG RL algorithm)

Мэдээллийн технологи (D061304)
Бакалаврын судалгааны ажил

Удирдагч: _____ Г.Гантулга

Хамтран удирдагч: _____

Гүйцэтгэсэн: _____ Б.Бямбаням (17B1NUM1479)

Улаанбаатар

2021 оны 02 сар

Зохиогчийн баталгаа

Миний бие Батбаярын Бямбаням ”DDPG бататгасан сургалтын үйлдлийн шуугианыг турших, харьцуулах ” сэдэвтэй судалгааны ажлыг гүйцэтгэсэн болохыг зарлаж дараах зүйлсийг баталж байна:

- Ажил нь бүхэлдээ эсвэл ихэнхдээ Монгол Улсын Их Сургуулийн зэрэг горилохоор дэвшүүлсэн болно.
- Энэ ажлын аль нэг хэсгийг эсвэл бүхлээр нь ямар нэг их, дээд сургуулийн зэрэг горилохоор оруулж байгаагүй.
- Бусдын хийсэн ажлаас хуулбарлаагүй, ашигласан бол ишлэл, зүүлт хийсэн.
- Ажлыг би өөрөө (хамтарч) хийсэн ба миний хийсэн ажил, үзүүлсэн дэмжлэгийг дипломын ажилд тодорхой тусгасан.
- Ажилд тусалсан бүх эх сурвалжид талархаж байна.

Гарын үсэг: _____

Огноо: _____

ГАРЧИГ

ЗУРГИЙН ЖАГСААЛТ

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

УДИРТГАЛ

Reinforcement learning буюу бататгасан сургалтад тасралттай үйлдлийн хувьд суралцах үйл явц нь санамсаргүй үйлдлийг сонгох замаар явагддаг. Харин үргэлжилсэн үйлдлийн хувьд суралцах үйл явц нь үйлдэлд шуугианыг нэмэх замаар явагддаг. Deep Deterministic Policy Gradient (DDPG) бол тасралтгүй, үргэлжилсэн үйлдлүүдийг сурахад зориулагдсан алгоритм тул action space noise буюу үйлдлийн шуугиан ашиглагдана. Энэ судалгааны ажлаар энэхүү үйлдлийн шуугианыг туршин, харьцуулах бөгөөд ямар үр нөлөөтэй болохыг тодорхойлно.

0.1 Зорилго

DDPG бататгасан сургалтын үйлдлийн шуугианыг туршин, харьцуулж энэ шуугиан моделийг сургах үйл явцад хэрхэн нөлөөлж буйг ажиглан дүгнэлт гаргах

0.2 Зорилтууд

- Моделийг DDPG алгоритмыг ашиглан BiPedalWalker-V3 орчныг дуустал алхаж чаддаг болгох
- 3 төрлийн шуугианыг амжилттай хэрэгжүүлэх
- Хэрэгжүүлэлт хийхдээ кодыг аль болох ойлгомжтой DDPG алгоритын pseudo кодын дагуу хэрэгжүүлэх

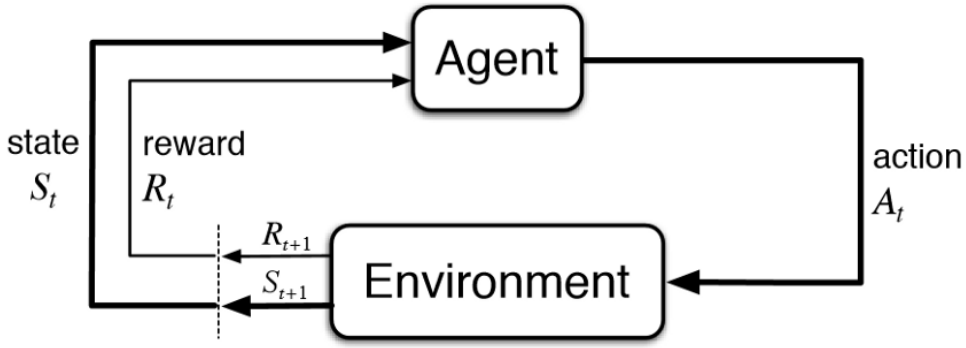
1. СУДАЛГАА

1.1 DDPG алгоритм

Алгоритмыг тайлбарлахаас өмнө Reinforcement Learning буюу бататгасан сургалтын талаар бага зэрэг тайлбарлая. Цаашдаа RL гэж товчлон бичнэ.

1.1.1 Бататгасан сургалтын алгоритм

RL нь агент болон орчин гэсэн хоёр хэсгээс тогтдог. Орчин гэдэг нь агент ажиллаж байгаа объектыг, агент гэдэг нь RL алгоритмыг илэрхийлнэ.



Зураг 1.1: Бататгасан сургалтын бүтэц

Орчин нь агентруу төлөвийг илгээх байдлаар эхэлдэг бөгөөд агент нь мэдлэг дээрээ тулгуурлан тухайн нөхцөл байдалд хариу үйлдэл үзүүлнэ. Үүний дараа орчин агент руу дараагийн төлөв болон reward-ыг илгээнэ. Агент нь орчноос хүлээн авсан reward-аар мэдлэгээ шинэчилнэ. Энэ давталт нь орчноос дуусгах төлөв илгээх хүртэл үргэлжилнэ. Ихэнх RL алгоритмууд дээрх байдлаар ажилладаг.

1.1.2 Бататгасан сургалттай холбоотай нэр томъёо

- Action (A): Агент-ийн хийх боломжтой бүх алхмууд
- State (S): Орчноос буцаж ирэх тухайн нөхцөл байдал
- Reward (R): өмнөх үйлдлийн үр дүнд гарсан ололт
- Policy (π): Агент одоогийн төлөв байдалд үндэслэн дараагийн үйлдлийг тодорхойлоход ашигладаг стратеги.
- Value (V): урт хугацааны ололт
- Q-value эсвэл action-value(Q): Value-тай төстэй. Гэхдээ одоогийн үйлдлийг нэмэлт параметрээр авдаг.

Model-free болон Model-based

Model нь орчны динамик загварчлалыг илэрхийлдэг. Загвар нь шилжилтийн магадлалыг T(s1 | (s0, a)) одоогийн төлөв s0 ба үйлдэл a хоёроос сурч, дараагийн s1 төлөвт шилждэг.

Model-free гэдэг нь мэдлэгээ шинэчлэхийн тулд туршилт ба алдаанд тулгуурладаг. Төлөвүүд болон үйлдлүүдийг хадгалах шаардлагагүй.

On-policy болон off-policy

On-policy агент нь value-г одоогийн policy-г ашигласан одоогийн үйлдэлд тулгуурлан сурдаг. Харин off-policy агент өөр нэг policy-г ашигласан үйлдэл a*-д тулгуурлан сурдаг.

1.1.3 DDPG алгоритм

Deep Deterministic Policy Gradient (DDPG) бол үргэлжилсэн, тасралтгүй үйлдлүүдийг сурахад зориулагдсан model-free off-policy алгоритм юм. Q-функц ба policy-ыг зэрэг сурдаг алгоритм

юм. Q-функцийг сурахын тулд off-policy өгөгдөл болон Bellman тэгшитгэлийг ашигладаг. Мөн policy-г сурахын тулд Q-функцыг ашигладаг.

DDPG алгоритм дараах 4 неороны сүлжээг ашигладаг:

- θ^Q:Q-network
- θ^μ:Deterministic policy function
- $\theta^{Q'}$:target Q network
- $\theta^{\mu'}$:target policy network

Q-network болон policy network хоёр нь Actor-critic аргатай маш төстөй.

- Actor (Deterministic policy function) - төлөвөөс хамааран үйлдлийг санал болгоно. Төлөвийг оролтоор авч үйлдлийг гаргана.
- Critic (Q-network) - төлөвөөс хамаарсан үйлдэл нь сайн эсвэл муу болохын урьдчилан таамагладаг. Төлөв болон үйлдлийг оролтоор авч Q-value-г гаргадаг.

Target network нь суралцсан сүлжээнүүдийг хянаж байдаг эх сүлжээнүүдийнхээ цагийн хоцрогдолтой хуулбарууд юм. Эдгээр сүлжээг ашиглан тогтвортой сурах байдлыг сайжруулдаг.

Доорх зурагт DPDG алгоритмын pseudo-code-ыг харууллаа. Үүнийг 4 хэсэгт задлан тайлбарлаж болно.

- Туршлагаа хадгалах (Experience replay)
- Actor болон critic сүлжээг шинэчлэх
- Target сүлжээг шинэчлэх
- Судалгаа, шинжилгээ хийх (Exploration)

DDPG алгоритмын pseudo code

θ^Q болон θ^μ жинтэйгээр critic сүлжээ $Q(s_i, a_i|\theta^Q)$ болон actor сүлжээ $\mu(s|\theta^m u)$ -г үүсгэнэ

$\theta^{Q'} \longleftarrow \theta^Q$, $\theta^{\mu'} \longleftarrow \theta^\mu$ жинтэйгээр target сүлжээ $Q^{'}$ болон $\mu^{'}$-г үүсгэнэ

Replay buffer-аа үүсгэнэ

for episode = 1, M do

Анхны төлөв болох s1-г авна

for t = 1, T do

Тухайн policy болон шуугиан дээрээ үндэслэн үйлдлээ сонгоно $a_t = \mu(s_t|\theta^\mu) + N_t$

Үйлдэл a_t-гээ гүйцэтгээд reward r_t болон шинэ төлөв $s_t + 1$-ээ авна

Replay buffer-даа төлөв, үйлдэл, reward, шинэ төлөвөө $(s_t, a_t, r_t, s_t + 1)$ хадгалж авна

Replay buffer-аасаа N тооны санамсаргүй утгыг авна

$y_i = r_i + \gamma Q^{'}(s_i + 1, \mu^{'}(s_i + 1|\theta^{\mu^{'}})|\theta^{Q'})$ утгыг онооно

Loss-ыш багасгаж critic сүлжээг шинэчилнэ: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

Actor policy-г шинэчлэнэ:

$$\bigtriangledown_\theta \mu J(\theta) \approx \frac{1}{N}\sum_i [\bigtriangledown_a Q(s, a|\theta^Q)|_s = s_i, a = \mu(s_i) \bigtriangledown_\theta \mu\mu(s|\theta^\mu)|_s = s_i]$$

Target сүлжээнүүдийг шинэчлэнэ:

$$\theta^{Q'} \longleftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$
$$\theta^{\mu'} \longleftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

end for

end for

Replay buffer

DDPG алгоритм нь replay buffer-ыг туршлагыг цуглуулахад ашигладаг. Цуглуулсан туршлагаа неороны сүлжээний параметрүүдийг шинэчлэхэд ашигладаг. Value болон policy сүлжээг шинэчлэхдээ

replay buffer дахь туршлагуудаас санамсаргүй байдлаар цуглуулан ашигладаг.

Яагаад replay buffer-ыг ашиглаж байгаа вэ гэхээр алгоритмд хамааралгүй байдлаар тархсан өгөгдөл хэрэгтэй. Ийм өгөгдлүүдийг replay buffer дахь туршлагуудаас санамсаргүй байдлаар сонгон авах байдлаар цуглуулж болно.

Actor (Policy) болон Critic (Value) сүлжээг шинэчлэх

Value сүлжээг шинэчлэх үйл явц нь Q-learning-тэй төстэй байдлаар хийгддэг. Шинэчлэгдсэн Q value-г Bellman-ны тэгшитгэлээс гарган авна:

$$y_i = r_i + \gamma Q^{'}(s_i + 1, \mu^{'}(s_i + 1|\theta^{\mu^{'}})|\theta^{Q'})$$

DDPG-д дараагийн төлөв Q утгуудыг target value network, target policy network ашиглан тооцдог. Дараа нь шинэчлэгдсэн Q утга ба анхны Q утга хоорондын дундаж квадрат алдааг хамгийн бага хэмжээнд хүртэл бууруулна:

$$Loss = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

Анхны Q value нь target network-ооc биш value network-ооc бодогдон гарна.

Policy функцийн хувьд гол зорилго нь буцан ирэх үр дүн хамгийн дээд хэмжээнд байх юм:

$$J(\theta) = E[Q(s, a)|_s = s_t, a_t = \mu(s_t)]$$

Policy алдагдлыг тооцоолохын тулд зорилгын функцийн деривативыг авна:

$$\bigtriangledown_\theta \mu J(\theta) \approx \bigtriangledown_a Q(s, a) \bigtriangledown_\theta \mu \mu(s|\theta^\mu)$$

Policy-гоо off-policy байдлаар шинэчилж байгаа учир санамсаргүй байдлаар авсан туршлагуудынхаа градиентүүдийн нийлбэрийн дундаж утгыг авна:

$$\bigtriangledown_\theta \mu J(\theta) \approx \frac{1}{N} \sum_i [\bigtriangledown_a Q(s, a|\theta^Q)|_s = s_i, a = \mu(s_i) \bigtriangledown_\theta \mu \mu(s|\theta^\mu)|_s = s_i]$$

Target сүлжээг шинэчлэх

Target сүлжээний параметрүүдийг хуулбарлаад, тэдгээрээр дамжуулан сурсан сүлжээнүүдээ хянана:

$$\theta^{Q'} \longleftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \longleftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

τ бол ихэвчлэн 1-тэй ойролцоо байхаар сонгосон параметр юм (жишээлбэл: 0.999).

Судалгаа шинжилгээ

Reinforcement learning буюу бататгасан сургалтад тасралттай үйлдлийн хувьд суралцах үйл явц нь санамсаргүй үйлдлийг сонгох замаар явагддаг. Харин үргэлжилсэн үйлдлийн хувьд суралцах үйл явц нь үйлдэлд шуугианыг нэмэх замаар явагддаг. DDPG алгоритмын баримт бичиг зохиогчид үйлдэлд шуугиан нэмэхийн тулд N:Ornstein-Uhlenbeck Process-г ашигласан байна:

$$\mu^{'}(s_t) = \mu(s_t|\theta_t^{\mu}) + N$$

Ornstein-Uhlenbeck процесс нь өмнөх шуугиантай уялдаатай холбоотой шуугианыг бий болгодог.

1.2 Ашигласан технологи

Gym, pytorch etc

1.2.1 Gym

Gym бол reinforcement learning буюу бататгасан сургалтын алгоритмуудыг хөгжүүлэх болон харьцуулахад зориулагдсан хэрэгсэл юм. Үүнийг ашиглан агентдаа алхах, тоглоом тоглох зэрэг бүх зүйлийг зааж болно.

Яагаад үүнийг ашигладаг вэ?

Бататгасан сургалт (RL) нь шийдвэр гаргахтай холбоотой машин сургалтын дэд талбар юм. Энэ нь агент нарийн төвөгтэй, тодорхойгүй орчинд хэрхэн зорилгодоо хүрч болохыг судалдаг. RL нь доорх 2 шалтгааны улмаас ихээр ашиглагдаж байна:

- RL нь дараалсан шийдвэр гаргахтай холбоотой бүхий л асуудлыг багтаасан байдаг. Жишээ нь роботын хөдөлгүүрийг удирдаж түүнийг үсрэх чадвартай болгох, үнэ, бараа материалын менежмент гэх мэт бизнесийн шийдвэр гаргах, видео тоглоом, ширээний тоглоом тоглох гэх мэт
- RL алгоритмууд олон хүнд хэцүү орчинд сайн үр дүнд хүрч эхэлсэн

Гэсэн хэдий ч RL судалгааны ажлыг RL-ын open-source орчин хангалттай олон янз байдаггүй бөгөөд тэдгээрийг тохируулах, ашиглахад хэцүү байдал болон орчны стандартчилал дутмаг гэсэн хоёр хүчин зүйл удаашруулж байна. Gym нь эдгээр 2 асуудлыг шийдэхийг зоридог.

1.2.2 BiPedalWalker-v2

Энэ бол gym-ын Box2D симулятор дахь нэг орчин юм. Гол зорилго нь bipedal роботыг алхаж сургах. Урагш алхах бүрд reward өгдөг. Нийтдээ төгсгөл хүртэл 300+ оноог өгдөг. Хэрэв робот унавал -100 оноо өгдөг. Илүү сайн агент нь илүү сайн оноо авах болно.

Төлөв нь их биений өнцгийн хурд, өнцгийн хурд, хэвтээ хурд, босоо хурд, хөлний байрлал, хөлний өнцгийн хурд, хөлтэй газар шүргэлцэх, 10 лидарын зай хэмжигч хэмжигдэхүүнээс бүрдэнэ. Мужийн векторт координат байхгүй байна.

1.2.3 Pytorch

Pytorch бол Torch сан дээр тулгуурласан open-source машин сургалтын сан юм. Python хэлэнд ихээр ашиглагддаг ч мөн C++ програмчлалын хэлд ашиглагддаг. Энэ нь GPU ашигладаг. PyTorch нь өндөр түвшний хоёр онцлог шинж чанарыг агуулдаг:

- GPU-г ашиглан тензорын тооцоолол хийх (NumPy гэх мэт)
- Гүнзгий неороны сүлжээг (Deep neural network) бий болгох

2016 оны 1 сард гарсан бөгөөд үүнээс хойш олон судлаачид үүнийг ашигласаар байна. Учир нь маш нарийн төвөгтэй неороны сүлжээг хялбараар бий болгодог. Мөн кодоо шалгахдаа заавал бүхлээр нь ажиллуулах шаардлагагүй болсон. Шаардлагатай тохиолдолд Pytorch-ын функцуудыг NumPy, SciPy, Cython зэргээр өргөтгөж болно.

2. ХЭРЭГЖҮҮЛЭЛТ

Хэрэгжүүлэлтийг python програмчлалын хэлийг ашиглан гүйцэтгэсэн. Орчинг бэлдэхдээ gym openai хэрэгслийг ашигласан. Pytorch санг тооцоолол хийх, неороны сүлжээ үүсгэх зэрэгт ашигласан. Дээр дурдсан DDPG алгоритмын pseudo кодын дагуу кодыг бичсэн. Чухал кодын хэсгийг доор орууллаа. Бүтэн кодыг хавсралт хэсэгт оруулсан.

BipedalWalker-v3 орчин дээр алгоритмыг ажиллуулан турших болно.

Зураг 2.1: BiPedalWalker-V3 орчин

Орчинг үүсгэж төлөвийн хэмжээс, үйлдлийн хэмжээс, хийж болох үйлдлийн тоог авна.

```
env = gym.make('BipedalWalker-v3')


state_dimension = env.observation_space.shape[0]
action_dimension = env.action_space.shape[0]
```

```
action_max = env.action_space.high[0]
```

Код 2.1: Орчин үүсгэх

Actor, target actor, critic болон target critic сүлжээг optimizer (сургагч)-ын хамт үүсгэх. Үүсгэхдээ төлөвийн хэмжээс, үйлдлийн хэмжээс, хийж болох үйлдлийн тоо зэргийг ашиглана.

```
actor = models.Actor(state_dimension, action_dimension, action_max)
target_actor = models.Actor(state_dimension, action_dimension,
   action_max)
actor_optimizer = torch.optim.Adam(actor.parameters(), lr=0.001)

critic = models.Critic(state_dimension, action_dimension)
target_critic = models.Critic(state_dimension, action_dimension)
critic_optimizer = torch.optim.Adam(critic.parameters(), lr=0.001)

for target_param, param in zip(target_actor.parameters(), actor.
   parameters()):
  target_param.data.copy_(param.data)

for target_param, param in zip(target_critic.parameters(), critic.
   parameters()):
    target_param.data.copy_(param.data)
```

Код 2.2: Actor critic сүлжээ үүсгэх

Replay buffer үүсгэх

```
ram = memory.ReplayBuffer(1000000))
```

Код 2.3: Replay buffer үүсгэх

Action noise-г үүсгэхдээ Ornstein-Uhlenbeck Process-г ашигласан

```
noise = utilities.OrnsteinUhlenbeckActionNoise(action_dimension)
```

Код 2.4: Шуугиан үүсгэх

Үйлдэл дээр correleted шуугианыг нэмэх

```
action_with_noise = action_without_noise.data.numpy() + (noise.sample()
    * action_max)
```

Код 2.5: Үйлдэл дээр шуугиан нэмэх

Үйлдэл дээр uncorreleted шуугианыг нэмэх

```
action_with_noise = action_without_noise.data.numpy() + (random.uniform
   (-0.2, 0.2) * action_max)
```

Код 2.6: Үйлдэл дээр шуугиан нэмэх

Үйлдлийг хийж шинэ төлөв, reward-г авах

```
new_observation, reward, done, info = env.step(action_with_noise)
```

Код 2.7: Үйлдэл хийх

Critic сүлжээг сургаад, шинэчлэх

```
# Critic network-g surgah

predicted_action = target_actor.forward(next_states).detach()
next_val = torch.squeeze(target_critic.forward(next_states,
   predicted_action).detach())
y_expected = rewards + 0.99*next_val
y_predicted = torch.squeeze(critic.forward(states, actions))

# Critic network-g shinechleh, critic loss-g tootsooloh
```

```
9
10 critic_loss = F.smooth_l1_loss(y_predicted, y_expected)
11 critic_optimizer.zero_grad()
12 critic_loss.backward()
13 critic_optimizer.step()
```

Код 2.8: Critic сүлжээг сургах шинэчлэх

Actor сүлжээг сургах

```
1 predicted_action = actor.forward(states)
2 actor_loss = -1*torch.sum(critic.forward(states, predicted_action))
3 actor_optimizer.zero_grad()
4 actor_loss.backward()
5 actor_optimizer.step()
```

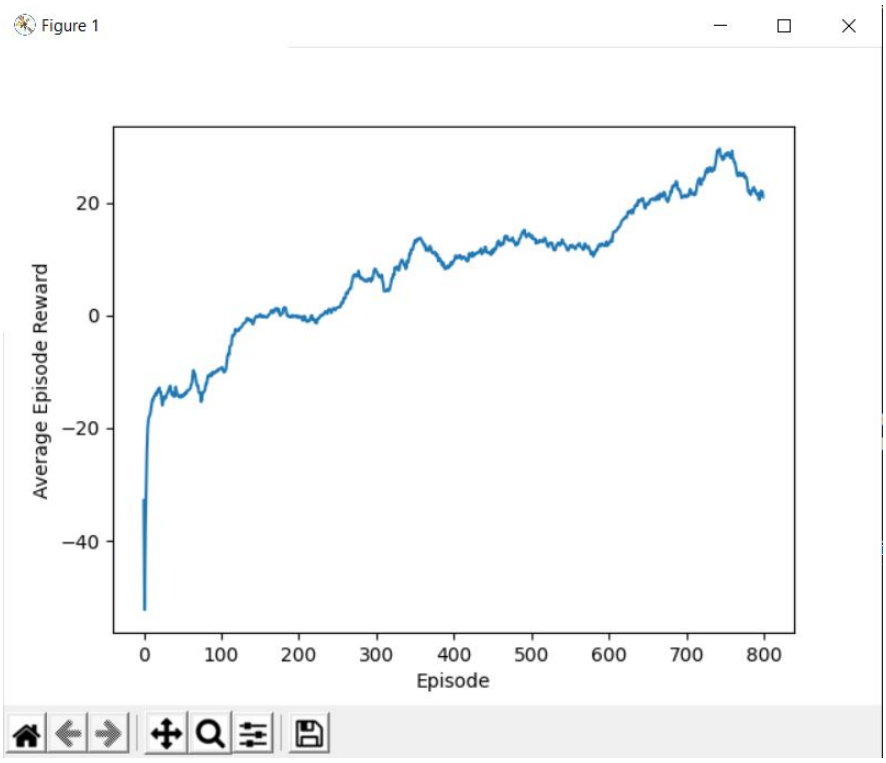
Код 2.9: Actor сүлжээг сургах

3. ҮР ДҮНГИЙН БОЛОВСРУУЛАЛТ

3.1 Туршилтын үр дүн

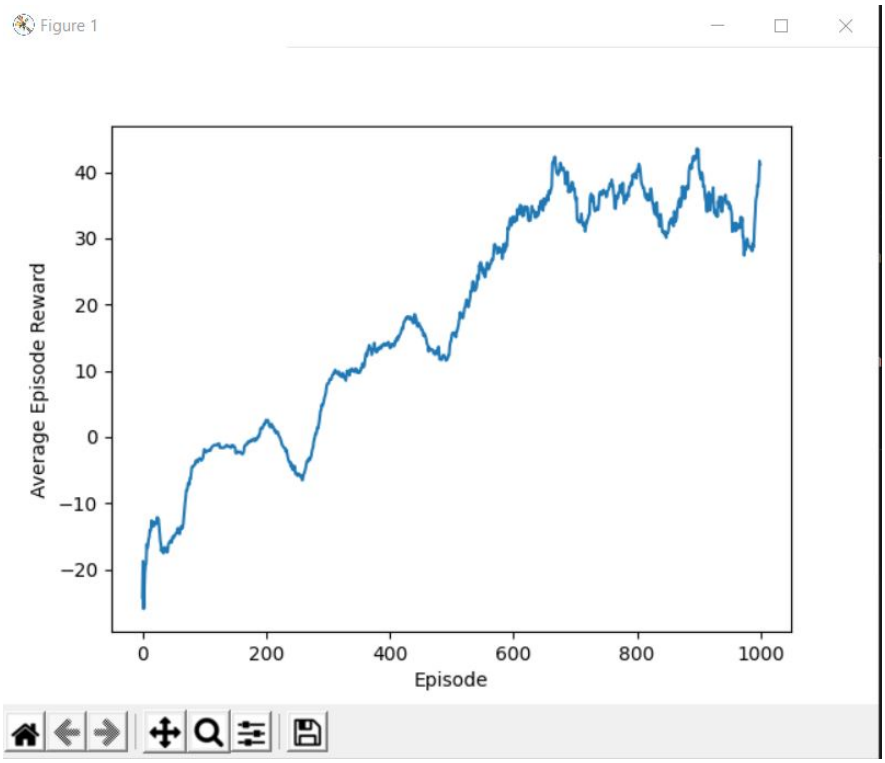
Хэрэв алгоритм зөв ажиллаж байгаа бол reward нь өсөж байх ёстой байдаг. Дундаж reward-ыг авахдаа episode болгоны нийлбэр reward-г олоод үүнийгээ list-д хадгалан аваад энэ list-ээс сүүлийн 40 үр дүнгийн дундажыг олж графикийг зурсан.

Доорх графикийн хувьд 800 episode ажилласан бөгөөд босоо тэнхлэгийн дагуу дундаж reward, хэвтээ тэнхлэгийн дагуу ажилласан episode-г авч байна. Орчны шуугианыг нэмэхдээ Ornstein-Uhlenbeck Process-г ашигласан. Энэ процесс нь өмнөх шуугиантай уялдаа хамааралтай буюу correlated шуугианыг гаргаж өгнө.



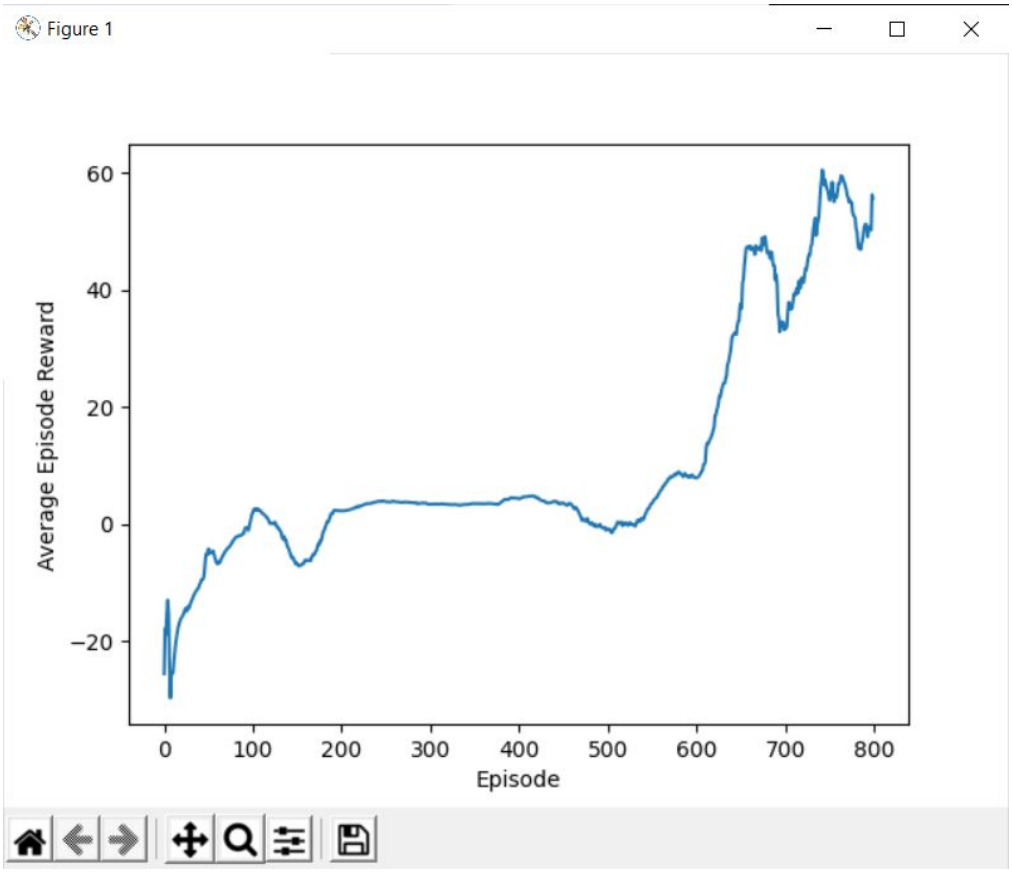
Зураг 3.1: Correlated noise

1000 episode ажиллуулсны дараа:



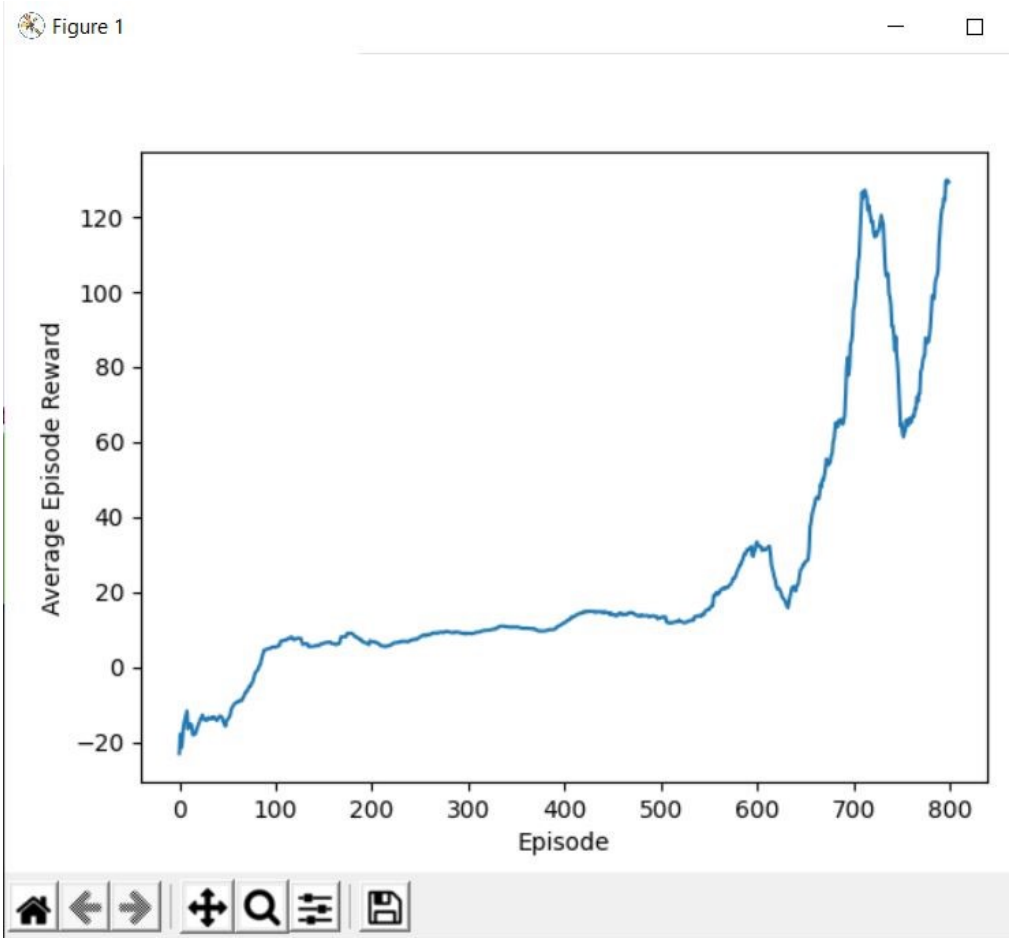
Зураг 3.2: Correlated noise

Доорх хоёр графикийн хувьд 800 episode ажилласан бөгөөд орчны шуугианыг -0.2 оос 0.2-ын хооронд санамсаргүй байдлаар сонгон авч үйлдэл дээрээ нэмж байгаа. Энэ нь өмнөх шуугиантай уялдаа хамааралгүй буюу uncorrelated шуугиан гэсэн үг юм.



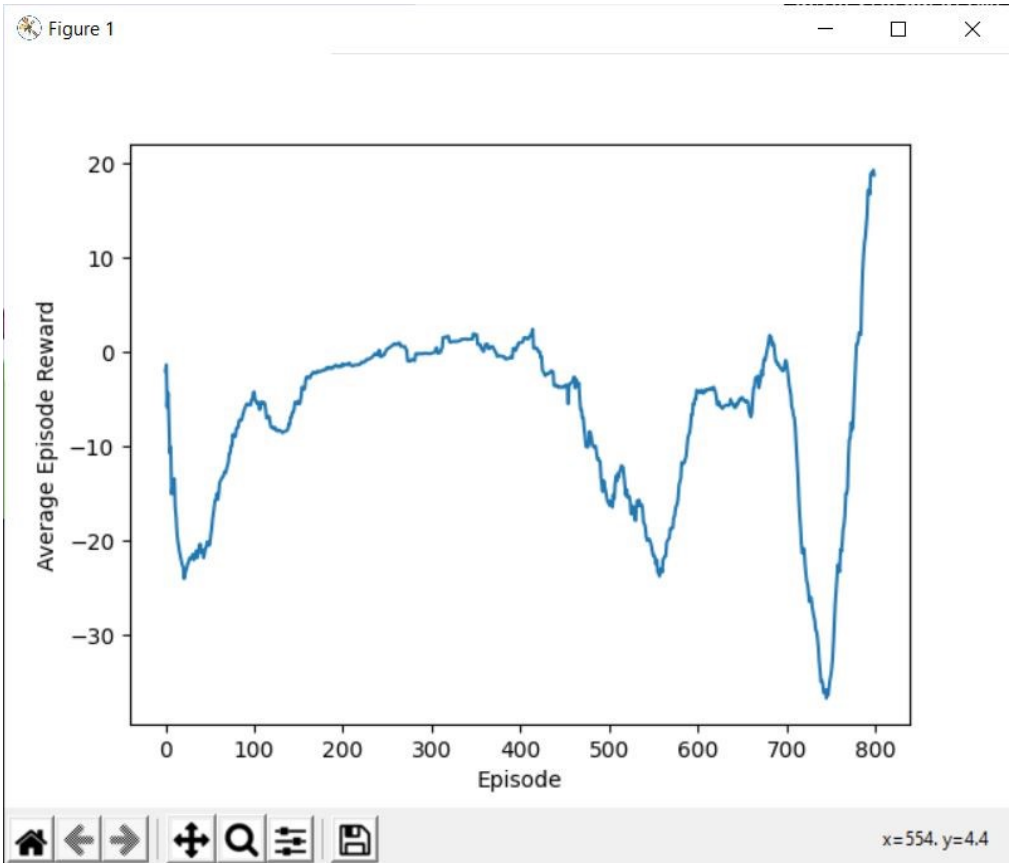
Зураг 3.3: Uncorrelated noise

Uncorreleted шуугианы өөр нэг график



Зураг 3.4: Uncorrelated noise 2

Доорх графикийн хувьд 800 episode ажилласан бөгөөд орчны шуугианыг -0.4-оос 0.4-ын хооронд санамсаргүй байдлаар сонгон авч үйлдэл дээрээ нэмж байгаа. Энэ нь мөн адил өмнөх шуугиантай уялдаа хамааралгүй буюу uncorrelated шуугиан юм. Графикаас харахад 0.4 өөр сонгон авсан үед нь үр дүн муу гарч байна.



Зураг 3.5: Uncorrelated noise 0.4

3.2 Үр дүнгийн харьцуулалт

Дүгнэлт

Дүгнэлтийг энд бич

Bibliography

[1] Deep Deterministic Policy Gradients Explained, TowardsDataScience, https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b

[2] Deep Deterministic Policy Gradient (DDPG), Keras, https://keras.io/examples/rl/ddpg_pendulum/

[3] Deep Deterministic Policy Gradient, Spinning Up, https://spinningup.openai.com/en/latest/algorithms/ddpg.html

[4] Continuous Control With Deep Reinforcement Learning, Lillicrap et al 2015, https://arxiv.org/pdf/1509.02971.pdf

A. A

Хавсралтын агуулга

В. КОДЫН ХЭРЭГЖҮҮЛЭЛТ