

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.Бямбаям (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

УДИРТГАЛ	1
0.1 Зорилго	1
0.2 Зорилтууд	1
1. СУДАЛГАА	2
1.1 Бататгасан сургалтын алгоритм	2
1.2 Гүн неороны сүлжээ	4
1.3 DDPG алгоритм	5
1.4 Хиперпараметруудийн тохируулга	8
1.5 Орчны шуугиан	8
1.6 Ашигласан технологи	11
2. ХЭРЭГЖҮҮЛЭЛТ	14
2.1 Алгоритмын хэрэгжүүлэлт	14
2.2 Алгоритмын турших орчин	14
2.3 Чухал кодын хэсгүүд	16
3. ҮР ДҮНГИЙН БОЛОВСРУУЛАЛТ	19
3.1 Туршилтын үр дүн	19
3.2 Үр дүнгийн харьцуулалт	23
ДҮГНЭЛТ	24
НОМ ЗҮЙ	24
ХАВСРАЛТ	25
А. А	26
В. КОДЫН ХЭРЭГЖҮҮЛЭЛТ	27

ЗУРГИЙН ЖАГСААЛТ

1.1	Бататгасан сургалтын бүтэц	2
1.2	Неороны сүлжээний жишээ	3
1.3	2 төрлийг орчны шуугиан	9
1.4	OU процессын график	10
1.5	Бие биенээсээ хамааралгүй шуугианы график	11
2.1	BiPedalWalker-V3 орчин	14
3.1	Correlated noise	19
3.2	Correlated noise	20
3.3	Uncorrelated noise	21
3.4	Uncorrelated noise 2	22
3.5	Uncorrelated noise 0.4	23

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

2.1	Орчин үүсгэх	15
2.2	Actor critic сүлжээ үүсгэх	16
2.3	Replay buffer үүсгэх	16
2.4	Шуугиан үүсгэх	16
2.5	Үйлдэл дээр шуугиан нэмэх	17
2.6	Үйлдэл дээр шуугиан нэмэх	17
2.7	Үйлдэл хийх	17
2.8	Critic сүлжээг сургах шинэчлэх	17
2.9	Actor сүлжээг сургах	18

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 3 төрлийн шуугианыг харьцуулах, турших бөгөөд ямар үр нөлөөтэй, аль шуугиан нь илүү 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

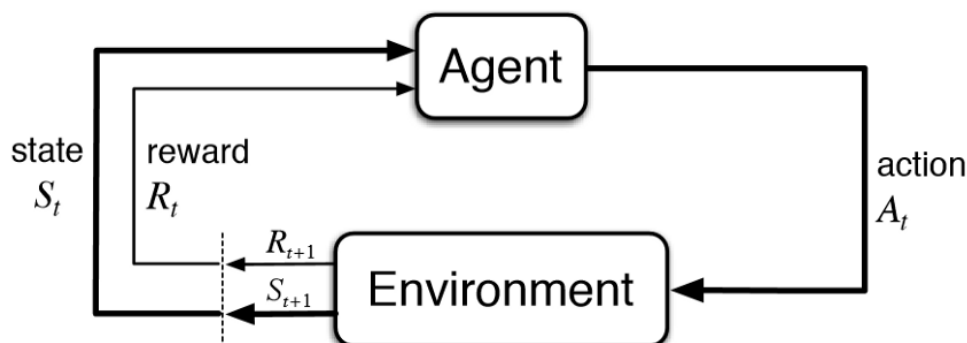
- Моделийг DDPG алгоритмыг ашиглан BiPedalWalker-V3 орчныг дуустал алхаж чаддаг болгох
- Бие биенээсээ хамааралтай шуугиан, бие биенээсээ хамааралгүй шуугиан, параметр шуугиан гэх гурван төрлийн шуугианыг амжилттай хэрэгжүүлэх
- Хэрэгжүүлэлт хийхдээ кодыг аль болох ойлгомжтой DDPG алгоритын pseudo кодын дагуу хэрэгжүүлэх

1. СУДАЛГАА

1.1 Бататгасан сургалтын алгоритм

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

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

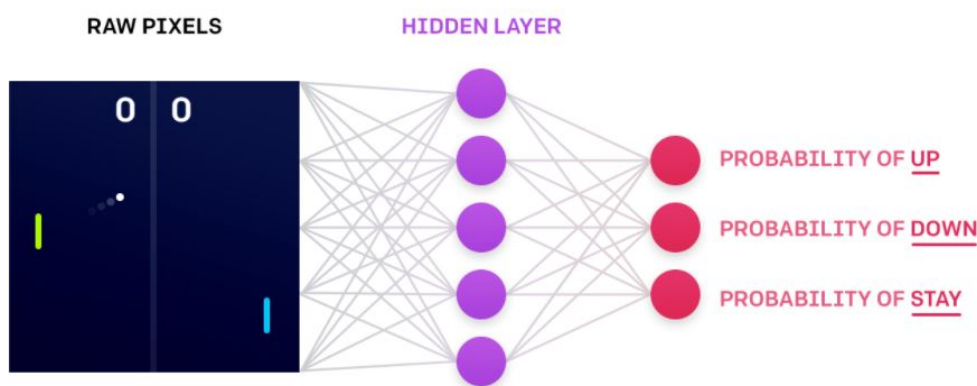


Зураг 1.1: Бататгасан сургалтын бүтэц

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.

Агентын төлөв байдлыг дүрслэхийн тулд тухайн нөхцөл байдалд агент хэрхэн ажиллах ёстойг тооцоолох полиси (policy) функцыг (агентын тархи гэсэн үг) тодорхойлдог. Парактик дээр полиси (policy) нь ихэвчлэн тоглоомын одоогийн төлөвийг оролт болгон авч зөвшөөрөгдсөн үйлдлүүдийн аль нэгийг хийх магадлалыг тооцдог неороны сүлжээ юм.

Доорх зурагт үзүүлсэн Понг тоглоомонд полиси нь дэлгэцийн пикселийг авч тоглогчийн цохиурыг (ногоон) дээш, доош эсвэл хоёуланг нь хөдөлгөх магадлалыг тооцоолж болно. Доорх зурагт полиси неороны сүлжээ буюу Hidden layer юм.



Зураг 1.2: Неороны сүлжээний жишээ

1.1.1 Бататгасан сургалттай холбоотой 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.

1.1.2 Model-free

Model-free гэдэг нь мэдлэгээ шинэчлэхийн тулд шагналд тулгуурладаг. Төлөвүүд болон үйлдлүүдийг хадгалах шаардлагагүй.

1.1.3 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.

1.1.4 Суралцах үйл явц

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.

1.2 Гүн неороны сүлжээ

Хиймэл неороны сүлжээ (ANN) нь тархи хэрхэн ажилладагаас санаа авсан функцийг ойролцоологч (function approximator) юм. ANN нь нэг давхарга дахь нейронууд бүгд урд давхарга дахь нейронуудтай холбогддог хэд хэдэн давхарласан хиймэл неороноос бүрдэнэ. Бүх хиймэл неоронууд нь идэвхжүүлэх функцтэй байдаг бөгөөд энэ нь ихэвчлэн ReLu функц байдаг.

$$f(x) = \max(0, x)$$

X_j нь неоронтой холбогдсон тохиолдолд w_{ij} жинтэй байдаг ба нейрон бүр хэвийх утгатай b_i байдаг. Неороны гаралтыг дараах томъёогоор тооцоолж болно:

$$y = f(b_i + \sum_{h=1}^n w_{ij}x_j)$$

Эхний давхаргыг оролт болгон ашигласнаар оролтыг өөр давхаргаар дамжуулж, сүлжээний сүүлийн давхаргаас гарах утгыг авах боломжтой. ANN нь илүү нарийн төвөгтэй функцуудыг ойролцоолох боломжтой тохиромжтой олон давхарга, неоронуудтай бол гүн неороны сүлжээ (DNN) гэж үздэг.

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 : Actor сүлжээ
- θ^μ : Critic сүлжээ
- $\theta^{Q'}$: target Actor сүлжээ
- $\theta^{\mu'}$: target Critic сүлжээ

Actor сүлжээ нь төлөвөөс хамааран үйлдлийг санал болгоно. Төлөвийг оролтоор авч үйлдлийг гаргана. Critic сүлжээ нь 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 хуулбарууд юм. Эдгээр сүлжээг ашиглан тогтвортой сурах байдлыг сайжруулдаг.

Доор DDPG алгоритмын pseudo-code-ыг харууллаа. Үүнийг 4 хэсэгт задлан тайлбарлаж болно.

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

DDPG алгоритмын pseudo code

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

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

Replay buffer-аа үүсгэнэ

for episode = 1, M do

Анхны төлөв болох s_1 -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

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

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

Actor policy-г шинэчлэнэ:

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

Target сүлжээнүүдийг шинэчлэнэ:

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

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

end for

end for

Replay buffer

DDPG алгоритм нь replay buffer-ыг туршлагыг цуглуулахад ашигладаг. Цуглуулсан туршлагаа

неороны сүлжээний параметруудийг 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 болон Critic сүлжээг шинэчлэх

Critic сүлжээг шинэчлэх үйл явц нь Q-learning-тэй төстэй байдлаар хийгддэг. Шинэчлэгдсэн Q утгыг Беллманы 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 critic network, target actor 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 утга нь target network-оос биш critic network-оос бодогдон гарна.

Actor сүлжээний хувьд гол зорилго нь буцан ирэх үр дүн хамгийн дээд хэмжээнд байх юм:

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

Actor алдагдлыг тооцоолохын тулд зорилгын функцийн деривативыг авна:

$$\nabla_{\theta} \mu J(\theta) \approx \nabla_a Q(s, a) \nabla_{\theta} \mu \mu(s | \theta^{\mu})$$

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

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

Target сүлжээг шинэчлэх

Target сүлжээний параметруудийг хуулбарлаад, тэдгээрээр дамжуулан сурсан сүлжээнүүдээ хянана. Target сүлжээний параметруудийг тодорхой хугацааны алхам хийсний дараа дараах томъёогоор шинэчилдэг:

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

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

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

Шуугиан нэмэх

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.

1.4 Хиперпараметруудийн тохируулга

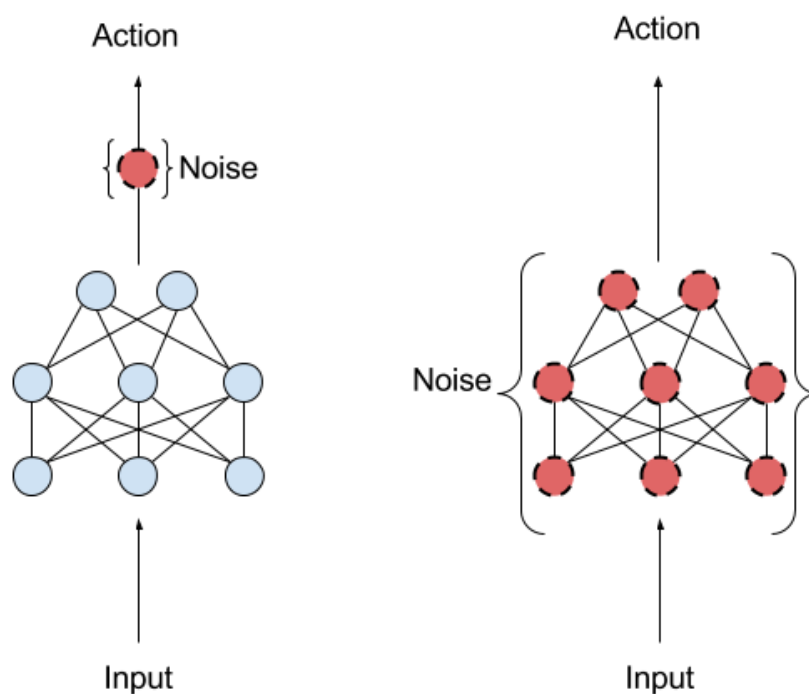
RL алгоритмууд нь алгоритм хэрхэн ажиллахыг өөрчилдөг параметрууд болох хиперпараметруудтэй байдаг. Алгоритм нь тодорхой орчинд сайн ажиллаж байгаа эсэхийг баталгаажуулахын тулд хиперпараметрын өөр өөр утгыг ашиглан туршиж аль хиперпараметрууд нь хамгийн сайн ажиллаж байгааг болохыг олж мэдэхийн тулд хиперпараметрын тохируулга хийх шаардлагатай байдаг.

1.5 Орчны шуугиан

RL алгоритмд ашигладаг ихэнх полиси нь стохастик буюу санамсаргүй байдлаа тодорхойлогдсон байдаг. Энэ нь зөвхөн аливаа үйлдэл хийх магадлалыг л тооцоолоно гэсэн үг бөгөөд сургалтын

явцад агент тодорхой төлөв байдалд олон удаа орж болох бөгөөд тухайн төлөв бүрт түүвэрлэлтийн (sampling) улмаас өөр өөр үйлдэл хийж болно гэсэн үг юм. Эдгээр үйлдлүүдийн зарим нь оновчтой зарим нь оновчгүй байх бөгөөд оновчгүй үйлдлийг багасгахын тулд агентийн үйлдэлд шуугианыг нэмж байгаа.

Бид үйлдлийн орчны шуугиан (Action space noise), параметр шуугиан (Parameter noise) гэх 2 төрлийн шуугианыг авч үзэх болно. Доорх зургын зүүн талынх нь үйлдлийн орчны шуугиан, баруун талынх нь параметр шуугиан юм.



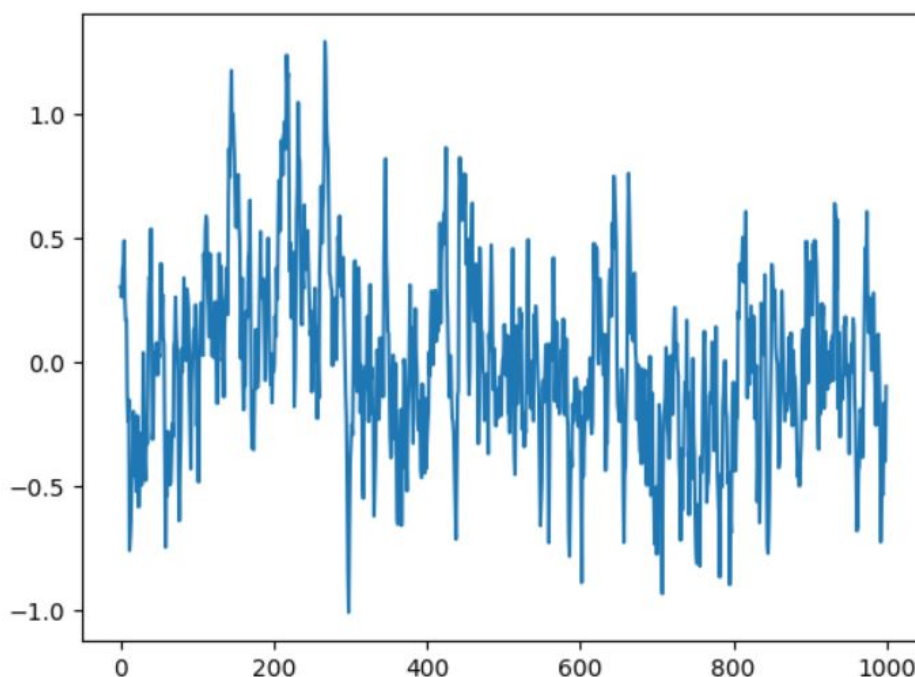
Зураг 1.3: 2 төрлийг орчны шуугиан

1.5.1 Үйлдлийн орчны шуугиан

Үйлдлийн орчны шуугиан дотор бие биетэйгээ хамааралтай шуугиан, бие биенээсээ хамааралгүй шуугиан гэх 2 шуугианыг авч үзнэ.

Бие биетэйгээ хамааралтай шуугиан

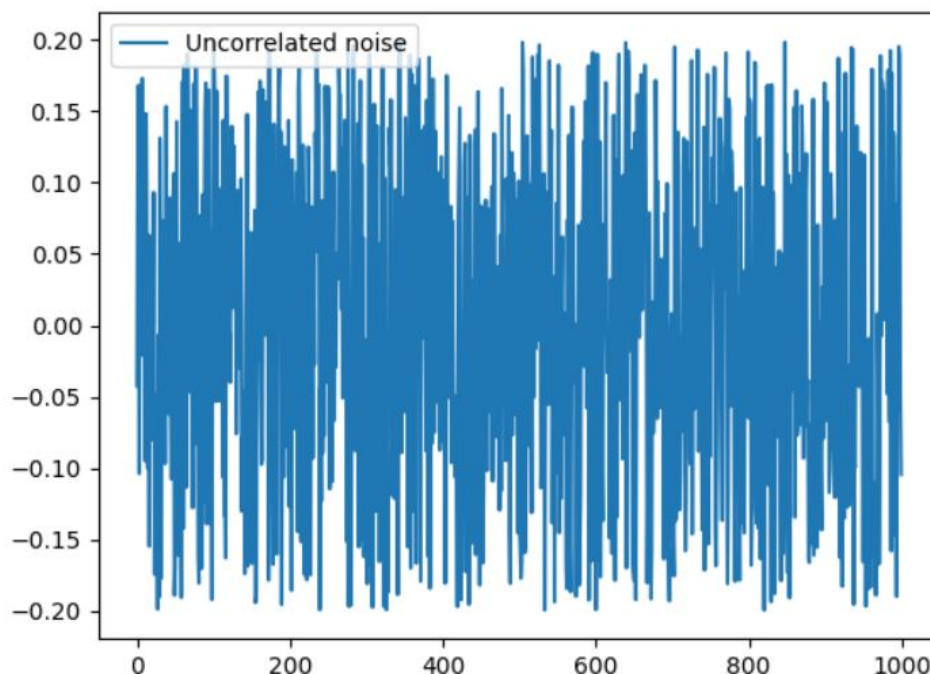
Correlated noise буюу бие биенээсээ хамааралтай шуугианыг үүсгэхдээ Ornstein–Uhlenbeck-ын процессыг ашиглаж байгаа. Энэ процесс нь бие биетэйгээ хамааралтай шуугианыг үүсгэж өгч байгаа. Доорх графикт ямархуу тархац, савалгаатай шуугиан үүсгэж байгаа харууллаа. Хэвтээ тэнхлэгийн дагуу хэдэн удаа үүсгэсэн, босоо тэнхлэгийн дагуу ямар утга авах нь харгалзаж байна:



Зураг 1.4: OU процессын график

Бие биенээсээ хамааралгүй шуугиан

Uncorrelated noise буюу бие биенээсээ хамааралгүй шуугианыг үүсгэхдээ $[-0.2, 0.2]$ -ын хооронд санамсаргүй тоо авах байдлаар үүсгэж байгаа. Яагаад $[-0.2, 0.2]$ -ын хооронд авч байгаа вэ гэхээр 3 төрлийн шуугианаа хооронд нь харьцуулах учираас адилхан параметртэй хэрэгжүүлж байгаа. Доорх графикт ямархуу тархац, савалгаатай шуугиан үүсгэж байгаа харууллаа (1000 удаа үүсгэсэн):



Зураг 1.5: Бие биенээсээ хамааралгүй шуугианы график

1.5.2 Параметр шуугиан

Параметр шуугиан нь бусад аргуудаас илүү үр дүнтэй арга юм. Параметр шуугиан нь үйлдэлд шуугиан нэмэх бус неороны сүлжээн параметр байдлаар шуугианыг нэмдэг. Шуугиан нь дасан зохицдог шуугиан (Adaptive noise) байна. Уламжлалт RL нь үйлдлийн орчны шуугианыг ашиглан агент нэг алхамаас нөгөө алхамд хийх үйлдэл бүртэй холбоотой магадлалыг өөрчилдөг.

1.6 Ашигласан технологи

1.6.1 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.

1.6.2 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.

1.6.3 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

2.1 Алгоритмын 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. Кодыг хавсралт хэсэг оруулсан. Github-аас үзэхийг хүсвэл дараах холбоосоор хандана уу https://github.com/bbyambanyam/ddpg_algorithm.git

2.2 Алгоритмын турших орчин

BipedalWalker-v3 орчин дээр алгоритмыг ажиллуулан туршилаа.



Зураг 2.1: BiPedalWalker-V3 орчин

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

```
1 env = gym.make('BipedalWalker-v3')
2
3 state_dimension = env.observation_space.shape[0]
4 action_dimension = env.action_space.shape[0]
5 action_max = env.action_space.high[0]
```

Код 2.1: Орчин үүсгэх

2.2.1 Хипер параметрууд

Доорх DDPG-ийн хипер параметрууд нь BiPedalWalker-V3 орчныг шийдвэрлэхэд тохируулагдсан болно.

- Actor learning rate: 0.0001 (Adam optimizer)
- Critic learning rate: 0.0001 (Adam optimizer)
- Memory buffer size: 1000000
- Minibatch size: 128
- OU-noise-theta: 0.15
- OU-noise-sigma: 0.2
- OU-noise-mu: 0
- normal-noise: 0.2
- Steps: 1600
- Target update: 0.001

2.3 Чухал кодын хэсгүүд

Actor, target actor, critic болон target critic сүлжээг optimizer (сургагч)-ын хамт үүсгэх.

Үүсгэхдээ төлөвийн хэмжээс, үйлдлийн хэмжээс, хийж болох үйлдлийн тоо зэргийг ашиглана.

```
1 actor = models.Actor(state_dimension, action_dimension, action_max)
2 target_actor = models.Actor(state_dimension, action_dimension,
    action_max)
3 actor_optimizer = torch.optim.Adam(actor.parameters(), lr=0.001)
4
5 critic = models.Critic(state_dimension, action_dimension)
6 target_critic = models.Critic(state_dimension, action_dimension)
7 critic_optimizer = torch.optim.Adam(critic.parameters(), lr=0.001)
8
9 for target_param, param in zip(target_actor.parameters(), actor.
    parameters()):
10     target_param.data.copy_(param.data)
11
12 for target_param, param in zip(target_critic.parameters(), critic.
    parameters()):
13     target_param.data.copy_(param.data)
```

Код 2.2: Actor critic сүлжээ үүсгэх

Replay buffer үүсгэх.

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

Код 2.3: Replay buffer үүсгэх

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

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

Код 2.4: Шуугиан үүсгэх

Үйлдэл дээр correlated шуугианыг нэмэх

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

Код 2.5: Үйлдэл дээр шуугиан нэмэх

Үйлдэл дээр uncorrelated шуугианыг нэмэх

```
1 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

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

Код 2.7: Үйлдэл хийх

Critic сүлжээг сургаад, шинэчлэх

```
1 # Critic network-g surgah  
2  
3 predicted_action = target_actor.forward(next_states).detach()  
4 next_val = torch.squeeze(target_critic.forward(next_states,  
    predicted_action).detach())  
5 y_expected = rewards + 0.99*next_val  
6 y_predicted = torch.squeeze(critic.forward(states, actions))  
7  
8 # 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

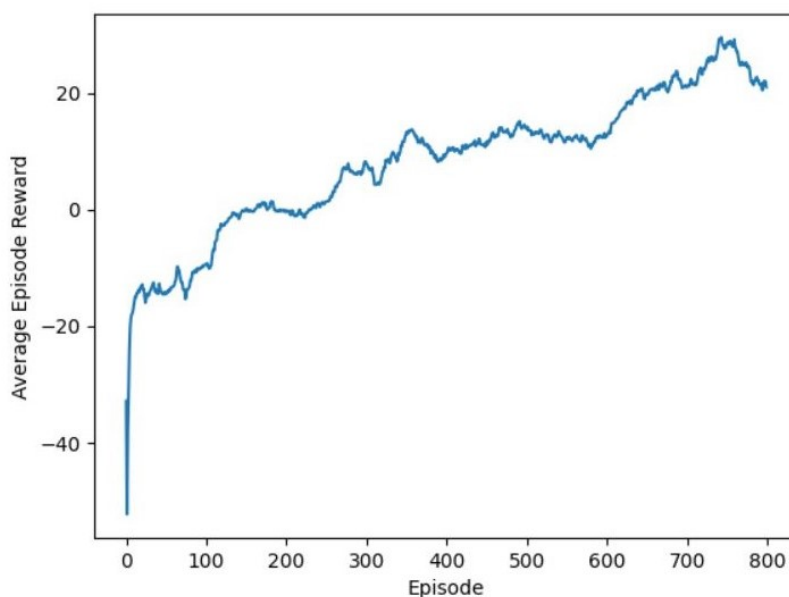
3.1 Туршилтын үр дүн

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.

3.1.1 Үйлдлийн орчны шуугиан

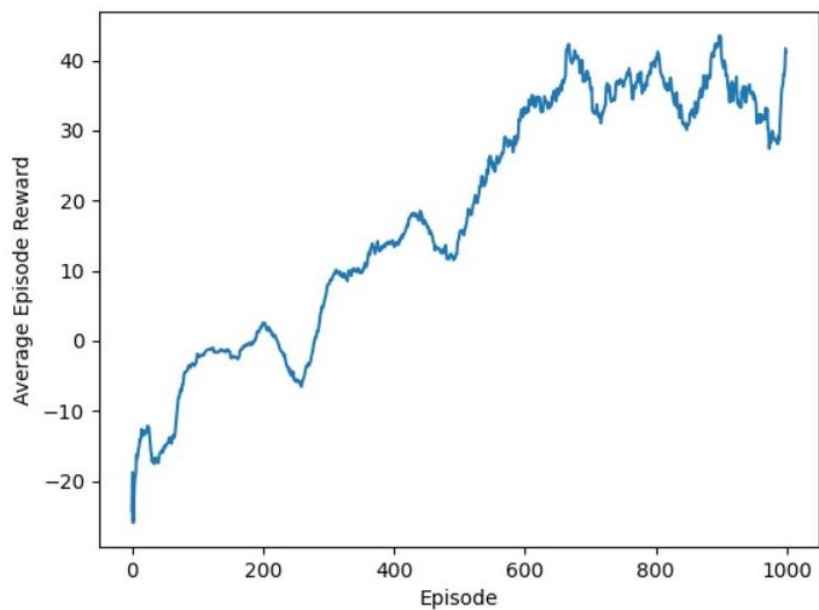
Correlated шуугиан

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

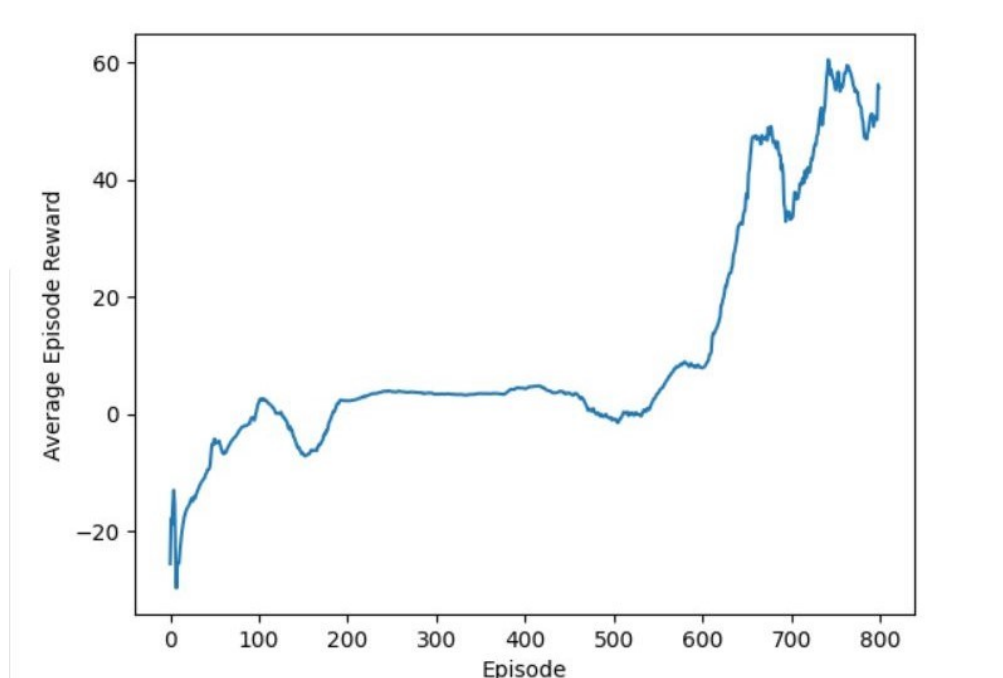
1000 episode ажиллуулсны дараа:



Зураг 3.2: Correlated noise

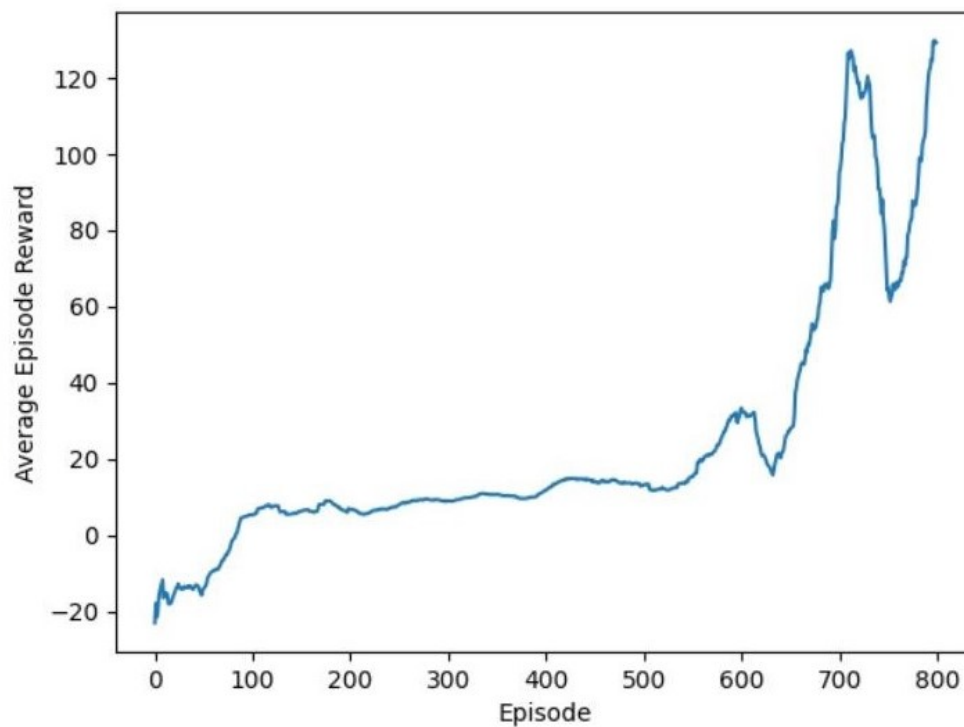
Uncorrelated шуугиан

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.



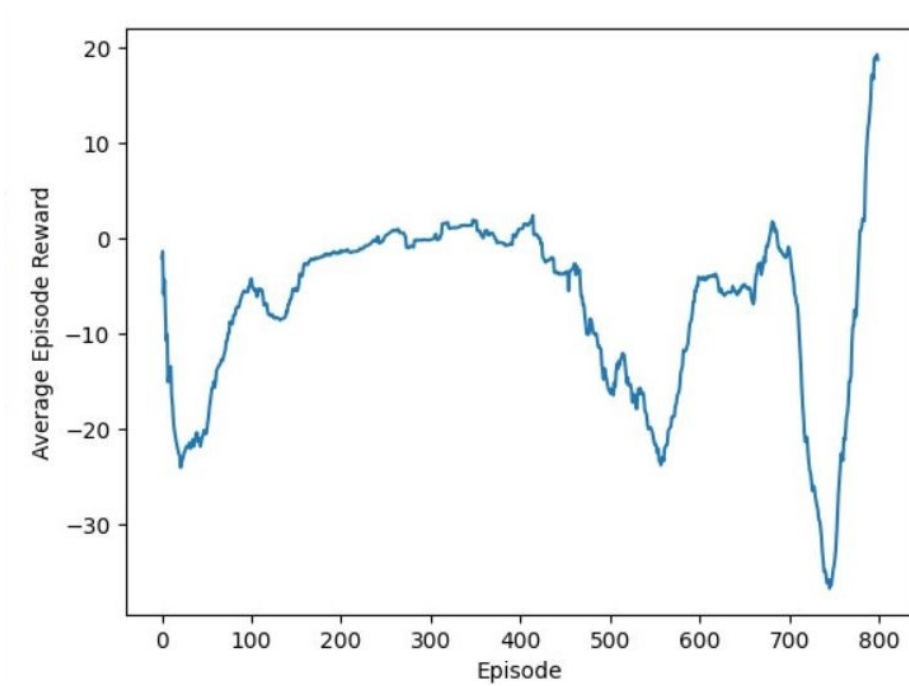
Зураг 3.3: Uncorrelated noise

Uncorrelated 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

3.1.2 *Parameter Space Noise*

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>
- [5] Parameter noise, Openai, <https://openai.com/blog/better-exploration-with-parameter-noise/>
- [6] Evolution strategies, Openai, <https://openai.com/blog/evolution-strategies/>

A. A

Хавсралтын агуулга

В. КОДЫН ХЭРЭГЖҮҮЛЭЛТ