

**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 .....	2
1. СУДАЛГАА .....	4
1.1 Бататгасан сургалтын алгоритм .....	4
1.2 Гүн неороны сүлжээ .....	6
1.3 DDPG алгоритм .....	7
1.4 Гиперпараметруудийн тохируулга .....	10
1.5 Орчны шуугиан .....	11
1.6 Ашигласан технологи .....	15
2. ХЭРЭГЖҮҮЛЭЛТ .....	18
2.1 Алгоритмын хэрэгжүүлэлт .....	18
2.2 Алгоритмын турших орчин .....	18
2.3 Чухал кодын хэсгүүд .....	21
3. ҮР ДҮНГИЙН БОЛОВСРУУЛАЛТ .....	26
3.1 Туршилтын үр дүн .....	26
3.2 Үр дүнгийн харьцуулалт .....	31
ДҮГНЭЛТ .....	33
НОМ ЗҮЙ .....	33
ХАВСРАЛТ .....	34
А. КОДЫН ХЭРЭГЖҮҮЛЭЛТ .....	35

## ЗУРГИЙН ЖАГСААЛТ

1.1	Бататгасан сургалтын бүтэц . . . . .	4
1.2	Неороны сүлжээний жишээ . . . . .	5
1.3	2 төрлийг орчны шуугиан . . . . .	11
1.4	OU процессын график . . . . .	12
1.5	2 өөр OU процессын график . . . . .	13
1.6	Бие биенээсээ хамааралгүй шуугианы график . . . . .	14
2.1	BiPedalWalker-V3 орчин . . . . .	19
3.1	Correlated noise . . . . .	26
3.2	Correlated noise . . . . .	27
3.3	Uncorrelated noise . . . . .	28
3.4	Uncorrelated noise 2 . . . . .	28
3.5	Uncorrelated noise 0.4 . . . . .	29
3.6	Параметр орчны шуугиан . . . . .	30
3.7	Параметр орчны шуугиан 800 episode . . . . .	30
3.8	Харьцуулалтын график . . . . .	31

## ХҮСНЭГТИЙН ЖАГСААЛТ

2.1	LR параметр шуугиан .....	20
2.2	LR Correlated .....	21
2.3	LR Uncorrelated .....	21

# Кодын жагсаалт

2.1	Орчин үүсгэх . . . . .	19
2.2	Actor critic сүлжээ үүсгэх . . . . .	21
2.3	Replay buffer үүсгэх . . . . .	22
2.4	Шуугиан үүсгэх . . . . .	22
2.5	Үйлдэл дээр шуугиан нэмэх . . . . .	22
2.6	Үйлдэл дээр шуугиан нэмэх . . . . .	22
2.7	Үйлдэл дээр шуугиан нэмэх . . . . .	23
2.8	Үйлдэл хийх . . . . .	23
2.9	Үйлдэл хийх . . . . .	23
2.10	Үйлдэл хийх . . . . .	24
2.11	Critic сүлжээг сургах шинэчлэх . . . . .	24
2.12	Actor сүлжээг сургах . . . . .	25
A.1	models.py . . . . .	35
A.2	memory.py . . . . .	36
A.3	utilities.py . . . . .	37
A.4	main.py . . . . .	39

# Товчилсон үг

DDPG - Deep Deterministic Policy Gradient

RL - Reinforcement learning

DNN - Deep Neural Network

ANN - Artificial Neural Network

LR - Learning rate



## УДИРТГАЛ

Сүүлийн жилүүдэд бататгасан сургалтын (RL) алгоритмуудыг компьютерын шинжлэх ухааны судлаачид ихээр сонирхон судалж байна. Ялангуяа Model-free RL алгоритмуудыг олон янзын төвөгтэй асуудлуудыг маш өндөр үр дүнтэйгээр шийдвэрлэхэд ихээр ашиглаж байгаа бөгөөд гүн неорал сүлжээг (DNN) нэмж ашигласнаар үр дүнг илүү сайжруулж байна.

RL алгоритмуудыг судлах, үнэлэх зориулалттай олон төрлийн сангууд бий болсоор байна. Эдгээрийн нэг нь OpenAI бөгөөд энэ нь нээлттэй эх сурвалж бүхий сан юм. Үүнд Gym буюу бататгасан сургалтын алгоритмуудыг хөгжүүлэх, харьцуулахад зориулсан хэрэгсэл, Baseline сан буюу сүүлийн үеийн бататгасан сургалтын алгоритмуудыг Tensorflow ашиглан гүйцэтгэсэн хэрэгжүүлэлт бүхий сан зэрэг багтана. Deep Deterministic Policy Gradient (DDPG) бол Baseline санд хэрэгжүүлэгдсэн алгоритмуудын нэг бөгөөд олон янзын нарийн төвөгтэй даалгавруудыг үр дүнтэй шийдвэрлэж чадах алгоритм юм.

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 бол тасралтгүй, үргэлжилсэн үйлдлүүдийг сурахад зориулагдсан алгоритм тул шуугиан ашиглагдана. Энэ судалгааны ажлаар 3 төрлийн шуугианыг харьцуулах, турших бөгөөд ямар үр нөлөөтэй, аль шуугиан нь илүү болохыг тодорхойлно.

## Зорилго

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

## Асуудал

PyTorch санг ашиглан хэрэгжүүлсэн DDPG алгоритмын үр дүн ямар төрлийн шуугиан ашиглахаас хамааран хэрхэн өөрчлөгдөж байна вэ? Аль шуугиан нь илүү үр дүнтэй байна вэ?

## Хамрах хүрээ

Орчны шуугиануудыг хооронд нь харьцуулж, үнэлэхийн тулд ижил орчинд DDPG алгоритмыг ашиглан хэд хэдэн удаа туршин үзнэ. Энэхүү судалгааны ажлын бүх туршилтыг BipedalWalker-V3 симуляцийн орчинд хийнэ. Мөн бүх шуугианы хувьд  $\mu=0$ ,  $\sigma=0.2$  байна.  $\mu$  нь дундаж утга,  $\sigma$  нь савлах утга.  $[-0.2, 0.2]$  гэсэн хязгаарын дотор шуугианы утгыг авна гэсэн үг.

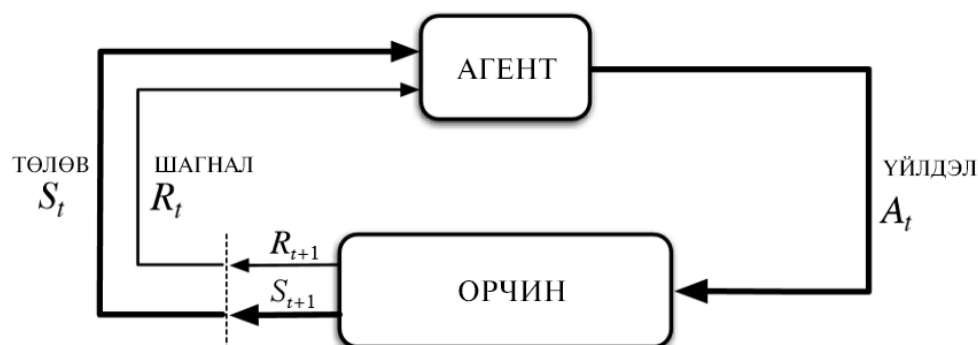
# 1. СУДАЛГАА

Энэ хэсэгт бататгасан сургалтын (RL) цаад санаануудыг болон DDPG алгоритмыг ойлгоход шаардлагатай зарим онолыг тайлбарлалаа. Мөн ашигласан технологийн тайлбарыг орууллаа.

## 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.



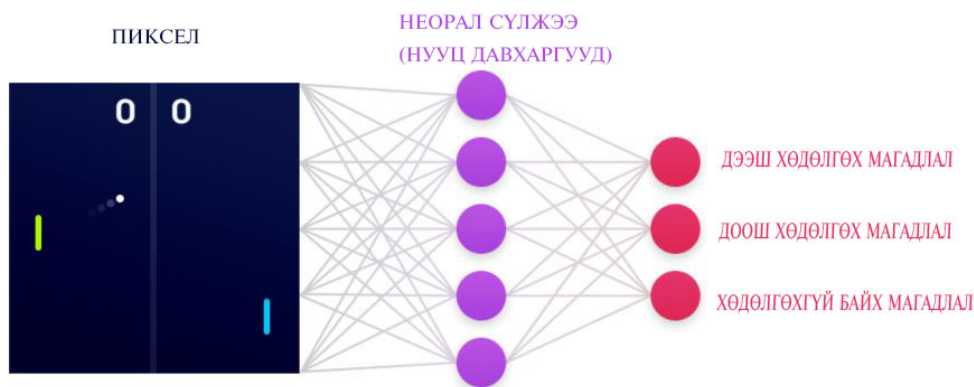
Зураг 1.1: Бататгасан сургалтын бүтэц

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) функцийг (агентын тархи гэсэн үг) тодорхойлдог. Практик

дээр полиси (policy) нь ихэвчлэн тоглоомын одоогийн төлөвийг оролт болгон авч зөвшөөрөгдсөн үйлдлүүдийн аль нэгийг хийх магадлалыг тооцдог неорал сүлжээ юм.

Доорх зурагт үзүүлсэн Понг тоглоомд полиси нь дэлгэцийн пикселийг авч тоглогчийн цохиурыг (ногоон) дээш, доош эсвэл хоёуланг нь хөдөлгөх магадлалыг тооцоолж болно. Доорх зурагт полиси нь неорал сүлжээ буюу нууц давхаргууд юм. [6]



Зураг 1.2: Неороны сүлжээний жишээ

### 1.1.1 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, шагнал
- Policy ( $\pi$ ): Агент одоогийн төлөв байдалд үндэслэн дараагийн үйлдлийг тодорхойлоход ашигладаг стратеги.
- Value (V): урт хугацааны шагнал
- 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 полисиг ашигласан одоогийн үйлдэлд тулгуурлан сурдаг. Харин off-policy агент өөр нэг полисиг 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 неорал сүлжээ (Artificial Neural Network - ANN) нь тархи хэрхэн ажилладгаас санаа авсан функцийг ойролцоологч (function approximator) юм. ANN нь нэг давхарга дахь неоралууд бүгд урд давхарга дахь неоралуудтай холбогддог хэд хэдэн давхарласан хиймэл неоралаас бүрдэнэ. Бүх хиймэл неоралууд нь идэвхжүүлэх функцтэй байдаг бөгөөд энэ нь ихэвчлэн ReLU функц байдаг.

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

$X_j$  нь неоралтай холбогдсон тохиолдолд  $w_{ij}$  жинтэй байдаг ба неорал бүр хэвийх утгатай  $b_i$  байдаг. Неоралын 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 функцуудийг

ойролцоолох боломжтой тохиромжтой олон давхарга, неоралуудтай бол гүн неорал сүлжээ (Deep Neural network - DNN) гэж үздэг.

### 1.3 DDPG алгоритм

DDPG бол үргэлжилсэн, тасралтгүй үйлдлүүдийг сурахад зориулагдсан model-free off-policy алгоритм юм. Q-функц ба полисиг зэрэг сурдаг алгоритм юм. Q-функцийг сурахын тулд off-policy өгөгдөл болон Bellman-ны тэгшитгэлийг ашигладаг. Мөн полисиг сурахын тулд Q-функцыг ашигладаг. [3]

DDPG алгоритм дараах 4 неорал сүлжээг ашигладаг:

- $\theta^Q$ :Актор сүлжээ
- $\theta^\mu$ :Критик сүлжээ
- $\theta^{Q'}$ :таргет Актор сүлжээ
- $\theta^{\mu'}$ :target Критик сүлжээ

Критик сүлжээ нь төлөвөөс хамааран үйлдлийг санал болгоно. Төлөвийг оролтоор авч үйлдлийг гаргана. Актор сүлжээ нь төлөвөөс хамаарсан үйлдэл нь сайн эсвэл муу болохын урьдчилан таамагладаг. Төлөв болон үйлдлийг оролтоор авч Q-value-г гаргадаг. Актор сүлжээ нь байнгын суралцаж явдаг бол Критик сүлжээ нь аажмаар суралцаж явдаг.

Таргет сүлжээ нь суралцсан сүлжээнүүдийг хянаж байдаг эх сүлжээнүүдийнхээ хуулбарууд юм. Эдгээр сүлжээг ашиглан тогтвортой сурах байдлыг сайжруулдаг. [2]

Доор DDPG алгоритмын pseudo-code-ыг харууллаа. Үүнийг 4 хэсэгт задлан тайлбарлаж болно. [1]

- Туршлагаа хадгалах (Experience replay)
- Actor болон critic сүлжээг шинэчлэх

- Target сүлжээг шинэчлэх
- Судалгаа хийх (Exploration)

## DDPG алгоритмын хуурмаг код

---

### Algorithm 1 DDPG алгоритмын хуурмаг код

---

- 1:  $\theta^Q$  болон  $\theta^\mu$  жинтэйгээр critic сүлжээ  $Q(s_i, a_i|\theta^Q)$  болон actor сүлжээ  $\mu(s|\theta^\mu)$  -г үүсгэнэ
  - 2:  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$  жинтэйгээр target сүлжээ  $Q'$  болон  $\mu'$  -г үүсгэнэ
  - 3: Replay buffer-аа үүсгэнэ
  - 4: **for** episode = 1, M **do**
  - 5:   Анхны төлөв болох  $s_1$ -г авна
  - 6:   **for** t = 1, T **do**
  - 7:     Тухайн полиси болон шуугиан дээрээ үндэслэн үйлдлээ сонгоно  $a_t = \mu(s_t|\theta^\mu) + N_t$
  - 8:     Үйлдэл  $a_t$ -гээ гүйцэтгээд шагнал  $r_t$  болон шинэ төлөв  $s_t + 1$ -ээ авна
  - 9:     Replay buffer-даа төлөв, үйлдэл, шагнал, шинэ төлөвөө  $(s_t, a_t, r_t, s_t + 1)$  хадгалж авна
  - 10:    Replay buffer-аасаа N тооны санамсаргүй утгыг авна
  - 11:     $y_i = r_i + \gamma Q'(s_i + 1, \mu'(s_i + 1|\theta^{\mu'})|\theta^{Q'})$  утгыг онооно
  - 12:    Алдааг багасгаж критик сүлжээг шинэчилнэ:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
  - 13:    Актер полисиг шинэчлэнэ:  

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

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

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$
  - 15:   **end for**
  - 16: **end for**
-

## Replay buffer

DDPG алгоритм нь replay buffer-ыг туршлагыг цуглуулахад ашигладаг. Цуглуулсан туршлагаа неорал сүлжээний параметруудийг шинэчлэхэд ашигладаг. Актор болон Критик 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

Критик 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 таргет критик сүлжээ, таргет актор сүлжээ 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 таргет сүлжээнээс биш критик сүлжээнээс бодогдон гарна.

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

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

Актор алдагдлыг тооцоолохын тулд зорилгын функцийн деривативыг авна:

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



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

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

### Таргет сүлжээг шинэчлэх

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

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

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

$\tau$  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.

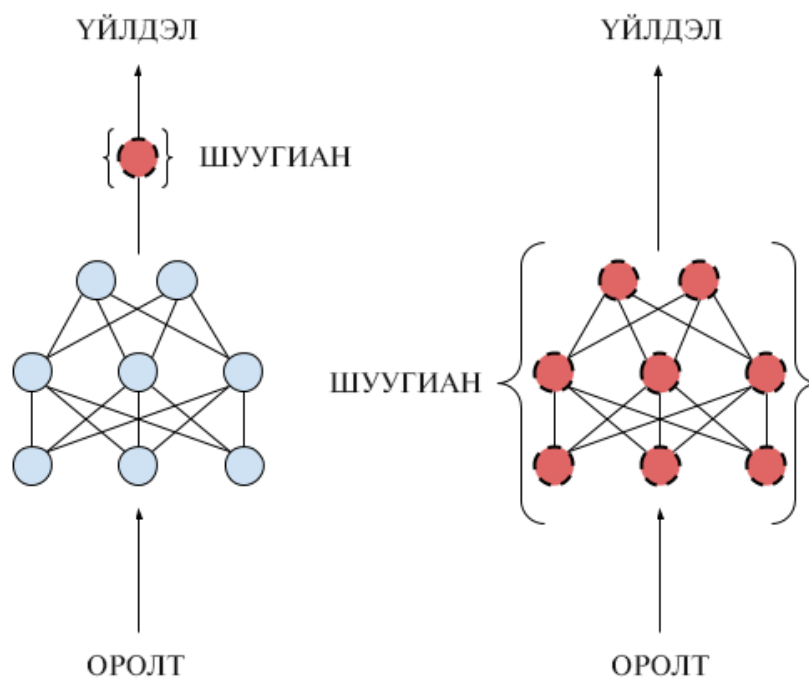
## 1.4 Гиперпараметруудийн тохируулга

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

## 1.5 Орчны шуугиан

RL алгоритмд ашигладаг ихэнх полиси нь стохастик буюу санамсаргүй байдлаар тодорхойлогдсон байдаг. Энэ нь зөвхөн аливаа үйлдэл хийх магадлалыг л тооцоолно гэсэн үг. Сургалтын явцад агент нь тодорхой нэг төлөв байдалд хэд хэдэн 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 агентын үйлдэлд шуугианыг нэмж байгаа. Ерөнхийдөө орчны шуугианыг агент нэг алхмаас нөгөө алхамд хийх үйлдэл бүртэй холбоотой магадлалыг өөрчлөхөд ашиглаж байна гэсэн үг.

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



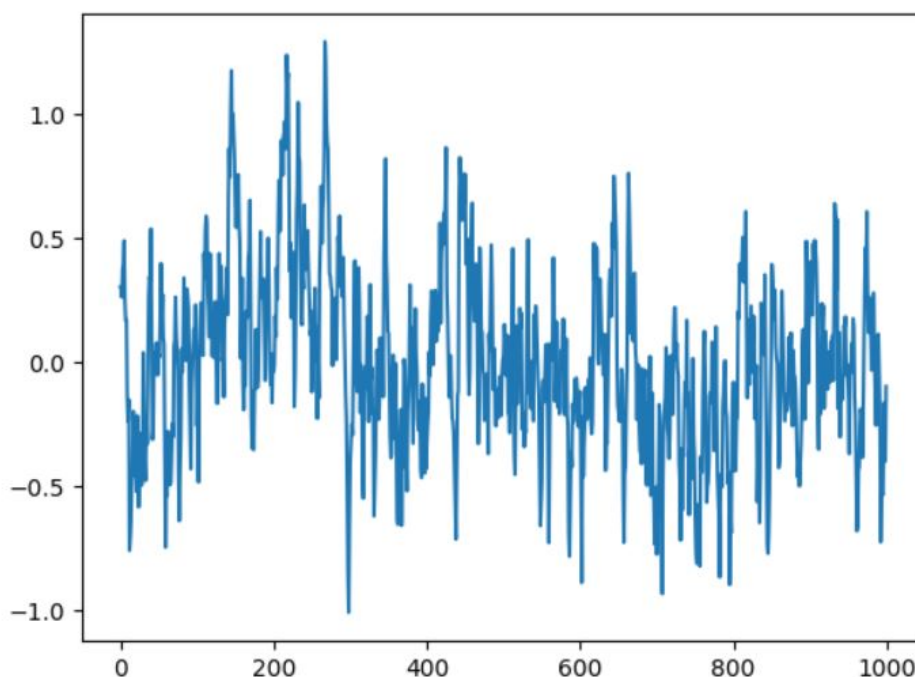
Зураг 1.3: 2 төрлийг орчны шуугиан

### 1.5.1 Үйлдлийн орчны шуугиан

Үйлдлийн орчны шуугиан гэдэг нь орчны шуугианыг үйлдэл дээр нэмэхийг хэлдэг. Үйлдлийн орчны шуугиан дотор бие биетэйгээ хамааралтай шуугиан, бие биенээсээ хамааралгүй шуугиан гэх 2 шуугианыг авч үзнэ. Доорх шуугиануудын графикыг туршилт хийн гарган авсан.

#### Бие биетэйгээ хамааралтай шуугиан

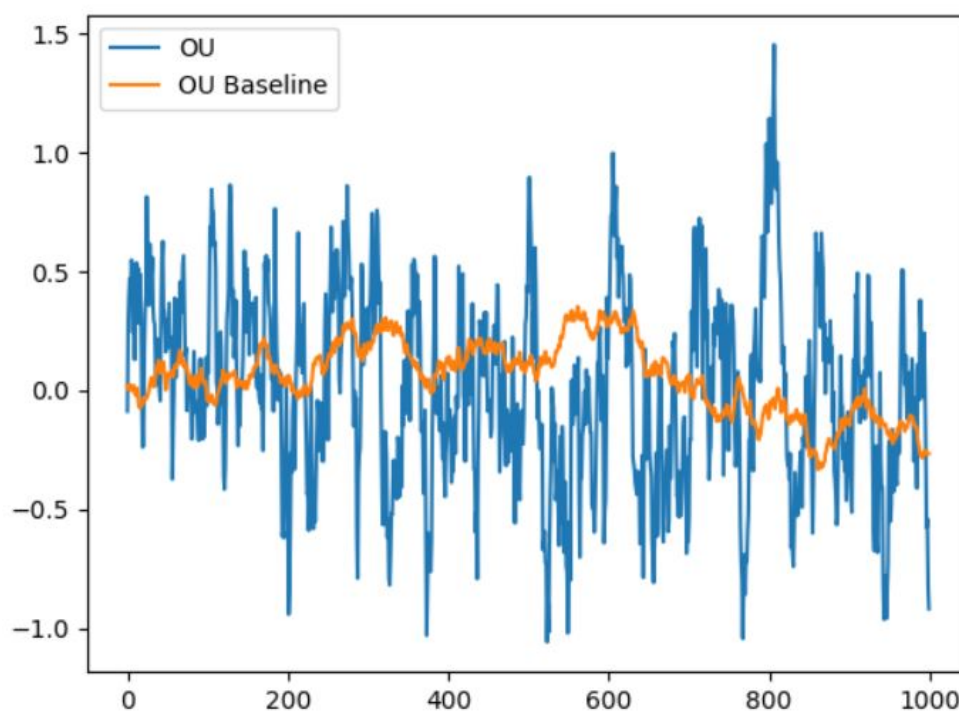
Correlated noise буюу бие биенээсээ хамааралтай шуугианыг үүсгэхдээ Ornstein–Uhlenbeck-ын процессыг ашиглаж байгаа. Энэ процесс нь бие биетэйгээ хамааралтай шуугианыг үүсгэж өгч байгаа. Доорх графикт ямархуу тархац, савалгаатай шуугиан үүсгэж байгаа харууллаа. Хэвтээ тэнхлэгийн дагуу хэдэн удаа үүсгэсэн,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



Зураг 1.4: OU процессын график

Туршилт хийхдээ хоёр өөр утга гаргах Ornstein–Uhlenbeck-ын процессын хэрэгжүүлэлтийг

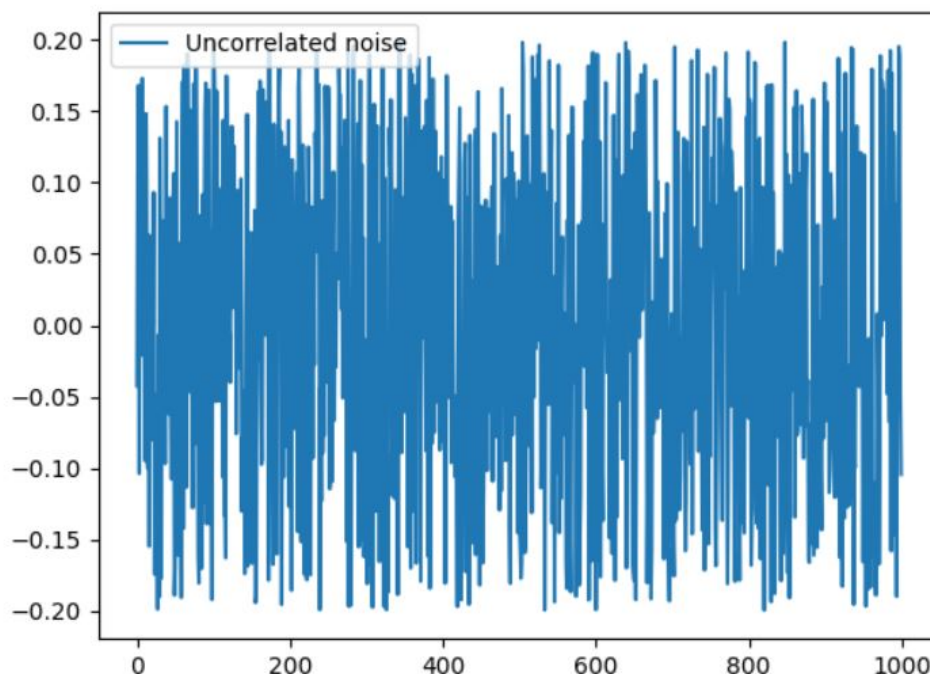
ашигласан. Нэг нь илүү тогтвортой алсуур аа савладаг, нөгөө тогтворгүй савлалт ихтэй байна. Доорх графикт ялгааг харууллаа:



Зураг 1.5: 2 өөр 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 учраас адилхан параметртэй хэрэгжүүлж байгаа. Доорх графикт ямархуу тархац, савалгаатай шуугиан үүсгэж байгааг харууллаа (1000 удаа үүсгэсэн):



Зураг 1.6: 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 неорал сүлжээн дээр параметр байдлаар шуугианыг нэмдэг. Шуугиан нь дасан зохицдог шуугиан (Adaptive noise) байна. Уламжлалт RL нь үйлдлийн орчны шуугианыг ашиглан агент нэг алхмаас нөгөө алхамд хийх үйлдэл бүртэй холбоотой магадлалыг өөрчилдөг. Параметр шуугиан нь алгоритмыг орчноо илүү үр дүнтэй судалж, илүү өндөр оноо авахад тусалдаг. [5]

### 1.5.3 Параметр шуугианыг хэрэгжүүлэх

Episode болгоны эхэнд актер сүлжээг хуулж аваад Гауссын шуугианыг нэмж шинэ шуугиантай актер сүлжээ үүсгэнэ. [7]

$$\tilde{\theta}^Q = \theta^Q + N(0, \sigma^2)$$

Тухайн episode шинэ үүссэн актор сүлжээн дээр сургагдана. Episode дууссаны дараа шинэ үүссэн актор сүлжээний үйлдлүүд болон энгийн актор сүлжээний үйлдлүүд хоорондын зай тооцно. Зайг тооцохдоо:

$$d(\theta^Q, \tilde{\theta}^Q) = \sqrt{1/N \sum_{i=1}^n E_s[(\theta^Q(s)_i - \tilde{\theta}^Q(s)_i)^2]},$$

$N$  = үйлдлийн хэмжээс

Шинэ үүссэн актор сүлжээний үйлдлүүд болон энгийн актор сүлжээний үйлдлүүд хоорондын зөрүүний квадратуудын нийлбэрийг олоод үүний язгуурыг олж байна.

Энэ нь ямар байгаагаас хамааран параметр шуугианыхаа сигмаг өөрчилнө:

$$\sigma_{k+1} = \begin{cases} \alpha \sigma_k & \text{Хэрвээ } d(\theta^Q, \tilde{\theta}^Q) \leq \delta, \\ \frac{1}{\alpha} \sigma_k & \text{Бусад тохиолдолд,} \end{cases}$$

Дээрх байдлаар шуугианыхаа  $\sigma$ -г өөрчилөн явна. Ийм байдлаар хэрэгжүүлэлтийг хийх болно.

## **Off-policy аргад зориулсан параметер шуугиан**

Off-policy үед суралцах үйл явцыг (exploration) шинэ үүсгэсэн актор сүлжээн дээр хийх бөгөөд сургах процессоо энгийн актор сүлжээн дээрээ хийнэ.

## **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, хэвтээ хурд, босоо хурд, хөлний байрлал, хөлний өнцгийн хурд, хөлтэй газар шүргэлцэх, 10 лидарын зай хэмжигч гэх 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, неорал сүлжээ үүсгэх зэрэгт ашигласан. Дээр дурдсан DDPG алгоритмын pseudo кодын дагуу кодыг бичсэн. 3 шуугианы гүйцэтгэлийг харьцуулах учраас ойролцоо хипер параметруудийг ашигласан.

DDPG алгоритмын хувьд Lillicrap et al.(2015)[4]-дээр тайлбарласантай төстэй actor, critic сүлжээний архитектурыг ашигласан. Target сүлжээнүүдийг  $\tau = 0.001$  гэсэн параметртэй soft-update хийсэн. Actor, critic сүлжээ хоёулаа Adam сургагчийг ашигласан.

Кодыг хавсралт хэсэг оруулсан. Github-аас харахыг хүсвэл дараах холбоосоор хандана уу [https://github.com/bbyambanyam/ddpg\\_algorithm.git](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 Гиперпараметрууд

Доорх DDPG-ийн гиперпараметрууд нь BiPedalWalker-V3 орчныг шийдвэрлэхэд тохируулагдсан бөгөөд гүйцэтгэлийн хувьд хамгийн сайн гарч байгаа нь юм. Зарим тохиолдолд өөр байж болно. Эдгээр параметруудийг олон удаагийн туршилтан дээр үндэслэн тохируулав.

- Actor learning rate: 0.0001 (Adam optimizer)
- Critic learning rate: 0.001 (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
- initial-stddev ( $\sigma$ ): 0.1
- desired-action-stddev ( $\delta$ ): 0.2
- adaptation-coefficient( $\alpha$ ): 1.01

Жишээ нь learning rate буюу сурах хурдыг доорх үр дүнгүүдээс үндэслэн тохируулсан:

Доорх хүснэгтэд LR буюу сурах хурдаас хамааран Параметр шуугиантай үед авсан дундаж шагналын хамгийн их утга хэрхэн гарч байгааг харууллаа.

Table 2.1: LR параметр шуугиан

Critic LR \ Actor LR	Critic LR	
	0.001	0.0001
0.001	145	112
0.0001	248	-

Доорх хүснэгтэд LR буюу сурах хурдаас хамааран Correlated буюу бие биетэйгээ хамааралтай шуугиантай үед авсан дундаж шагналын хамгийн их утга хэрхэн гарч байгааг харууллаа.

Table 2.2: LR Correlated

Critic LR \ Actor LR	0.001	0.0001
	0.001	0.0001
0.001	42	68
0.0001	198	-

Доорх хүснэгтэд LR буюу сурах хурдаас хамааран Uncorrelated буюу бие биенээсээ хамааралгүй шуугиантай үед авсан дундаж шагналын хамгийн их утга хэрхэн гарч байгааг харууллаа.

Table 2.3: LR Uncorrelated

Critic LR \ Actor LR	0.001	0.0001
	0.001	0.0001
0.001	123	68
0.0001	-	-

## 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. Buffer-аа үүсгэснийхээ дараа hot start хийхийн тулд санамсаргүй утгуудаар дүүргэж өгсөн.

```

1 ram = memory.ReplayBuffer(1000000)

```

Код 2.3: Replay buffer үүсгэх

Ornstein-Uhlenbeck Process-г үүсгэх

```

1 noise = utilities.OrnsteinUhlenbeckActionNoise(action_dimension)

```

Код 2.4: Шуугиан үүсгэх

Дээр үүсгэсэн шуугианаа ашиглан үйлдэл дээр correleled шуугианыг нэмэх

```

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

```

Код 2.5: Үйлдэл дээр шуугиан нэмэх

Үйлдэл дээр uncorreleled 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

Параметр шуугиан үүсгэх

```

1 #Parameter noise-d zoriulsan actor
2 actor_copy = models.Actor(state_dimension, action_dimension, action_max
    )
3
4 #Parameter noise
5 parameter_noise = utilities.AdaptiveParamNoiseSpec(initial_stddev=0.05,
    desired_action_stddev=0.3, adaptation_coefficient=1.05)

```

Код 2.7: Үйлдэл дээр шуугиан нэмэх

Параметр шуугианд зориулан үүсгэсэн шинэ actor сүлжээний неорал сүлжээн дээр параметр шуугианыг нэмэх

```

1 parameters = actor_copy.state_dict()
2 for name in parameters:
3     parameter = parameters[name]
4     rand_number = torch.randn(parameter.shape)
5     parameter = parameter + rand_number * parameter_noise.
        current_stddev

```

Код 2.8: Үйлдэл хийх

Үйлдлийг хийж шинэ төлөв, reward-г авах

```

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

```

Код 2.9: Үйлдэл хийх

Параметр шуугианы зай тооцох, сигмаг шинэчлэх

```
1 #Distance tootsoh
2 diff_actions = actor_copy_actions - actor_actions
3 mean_diff_actions = np.mean(np.square(diff_actions),axis=0)
4 distance = math.sqrt(np.mean(mean_diff_actions))
5
6 #Sigma-g update hiih
7 parameter_noise.adapt(distance)
```

Код 2.10: 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,
5     predicted_action).detach())
6 y_expected = rewards + 0.99*next_val
7 y_predicted = torch.squeeze(critic.forward(states, actions))
8
9 # Critic network-g shinechleh, critic loss-g tootsooloh
10
11 critic_loss = F.smooth_l1_loss(y_predicted, y_expected)
12 critic_optimizer.zero_grad()
13 critic_loss.backward()
14 critic_optimizer.step()
```

Код 2.11: 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 2.12: Actor сүлжээг сургах



## 3. ҮР ДҮНГИЙН БОЛОВСРУУЛАЛТ

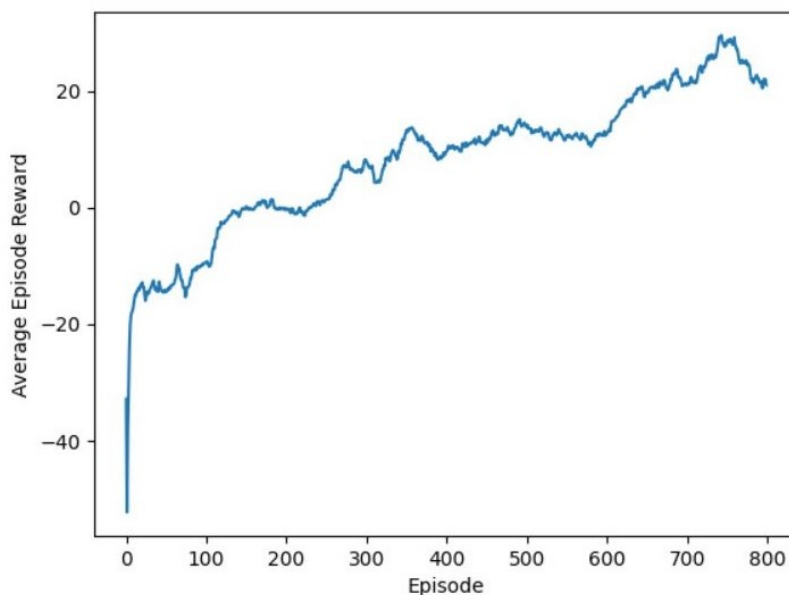
### 3.1 Туршилтын үр дүн

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

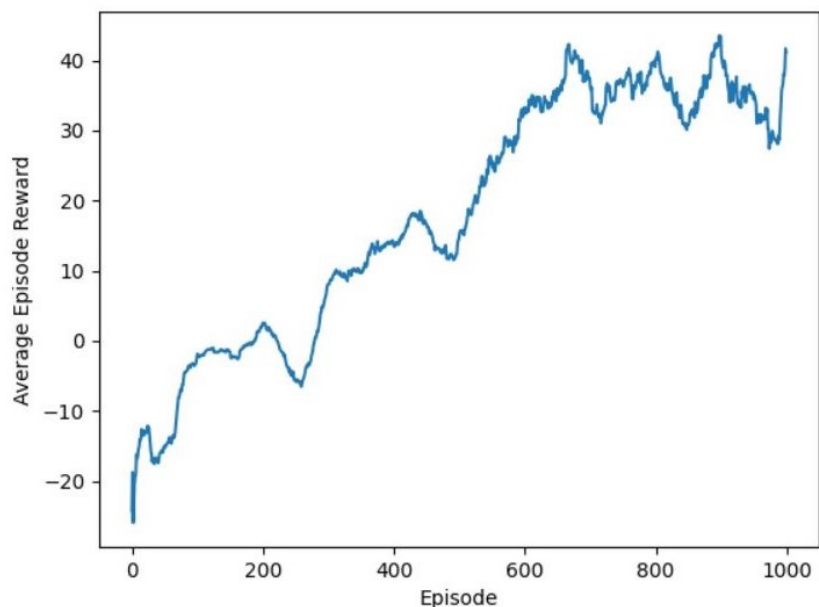
##### Correlated шуугиан

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. Доорх бүх графикийн хувьд Actor LEARNING RATE: 0.001 Critic LEARNING RATE: 0.001 бусад хипер параметрууд ижил.



Зураг 3.1: Correlated noise

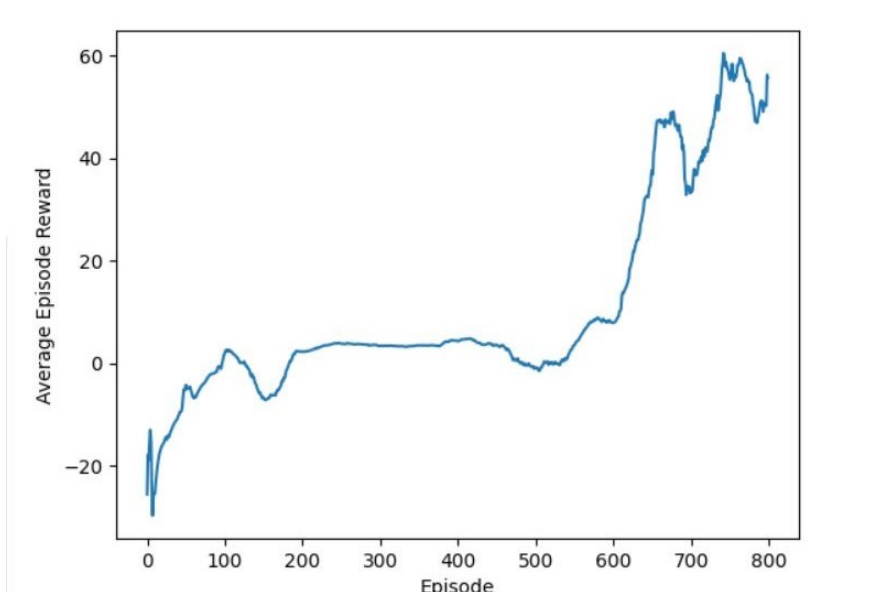
1000 episode ажиллуулсны дараа:



Зураг 3.2: Correlated noise

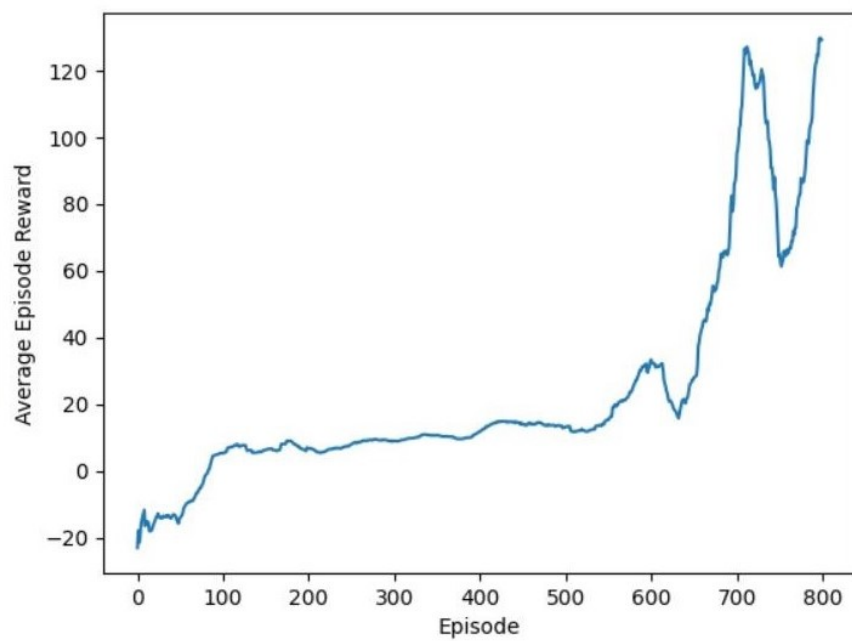
### Uncorrelated шуугиан

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. Доорх бүх грапикийн хувьд Actor LEARNING RATE: 0.001 Critic LEARNING RATE: 0.001 бусад хипер параметрууд ижил.



Зураг 3.3: Unrelated noise

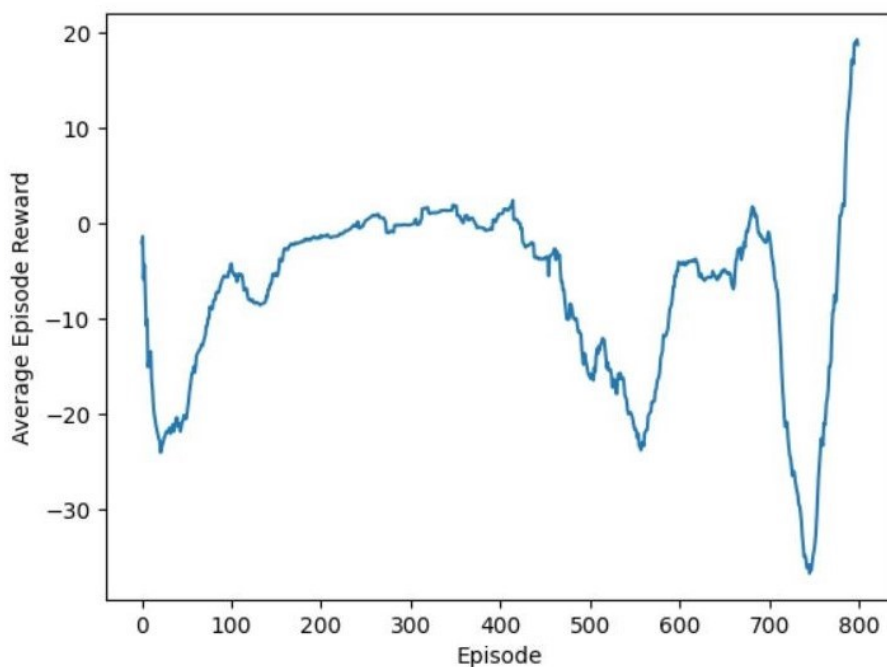
Unrelated шуугианы өөр нэг график:



Зураг 3.4: Unrelated noise 2

Доорх графикийн хувьд 800 episode ажилласан бөгөөд орчны шуугианыг -0.4-оос 0.4-

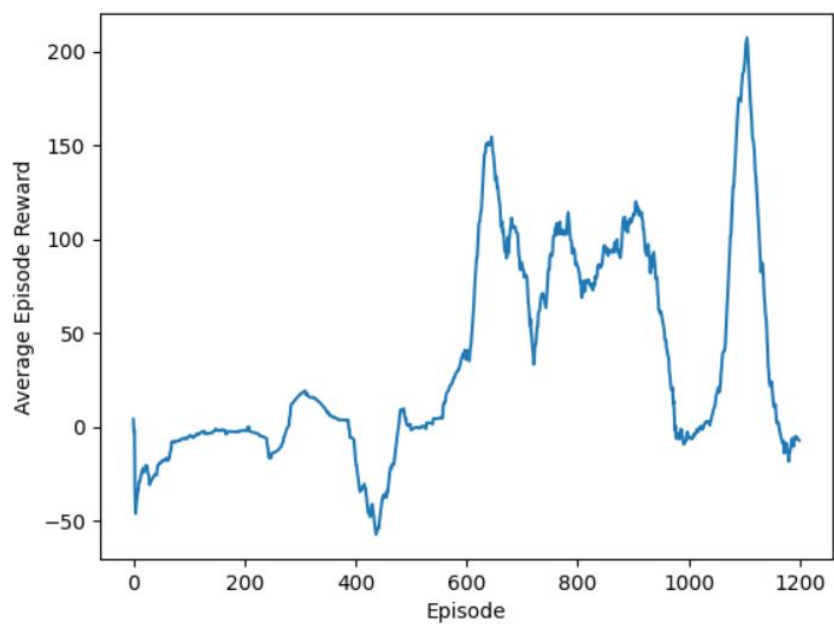
ын хооронд санамсаргүй байдлаар сонгон авч үйлдэл дээрээ нэмж байгаа. Энэ нь мөн адил өмнөх шуугиантай уялдаа хамааралгүй буюу uncorrelated шуугиан юм. Графикаас харахад 0.4 өөр сонгон авсан үед нь үр дүн муу гарч байна.



Зураг 3.5: Uncorrelated noise 0.4

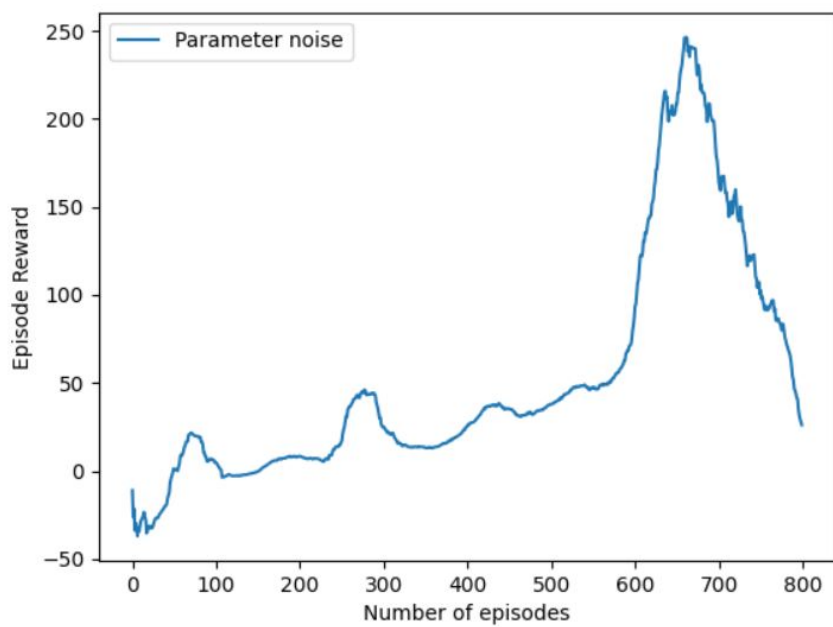
### 3.1.2 *Parameter Space Noise*

Доорх параметр шуугианы үр дүнг нь гаргахын тулд 1200 episode ажиллуулсан. ACTOR LEARNING RATE: 0.0001, CRITIC LEARNING RATE: 0.001:



Зураг 3.6: Параметр орчны шуугиан

Доорх графикийн хувьд 800 episode ажилласан. Савалгаа их ч хамгийн их дундаж шагнал нь бусад үр дүнгээс их байна

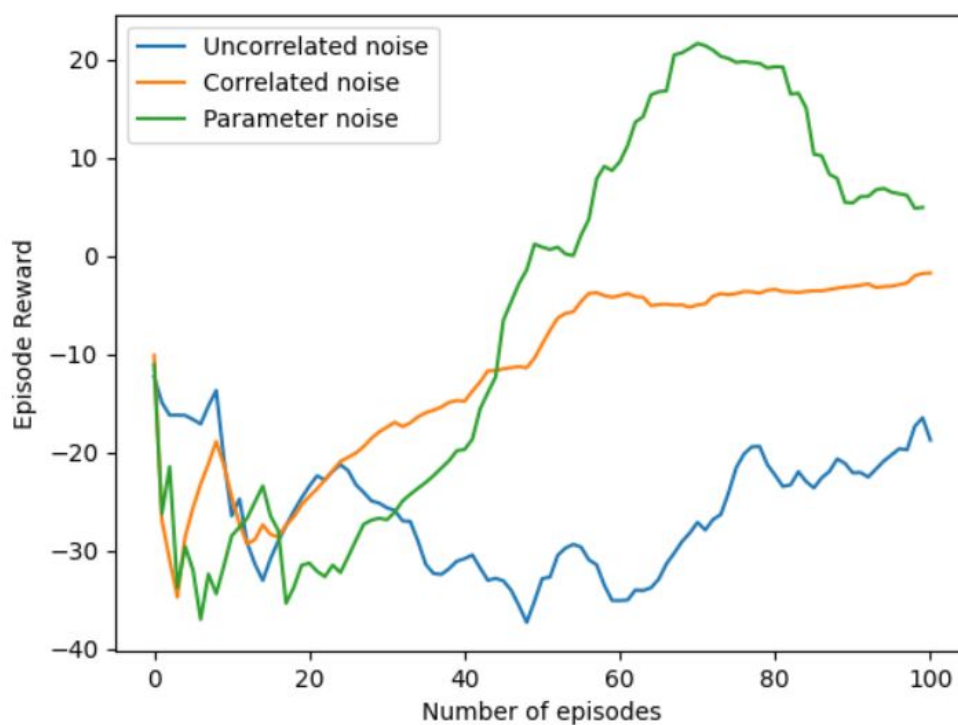


Зураг 3.7: Параметр орчны шуугиан 800 episode

## 3.2 Үр дүнгийн харьцуулалт

Харьцуулалт хийхийн тулд бүх шуугиан дээр ижил хипер параметртэйгээр ажиллуулсан. Uncorrelated noise буюу бие биенээсээ хамааралгүй шуугиан ( $\theta = 0, \sigma = 0.2$ ), Correlated noise буюу бие биетэйгээ хамааралтай шуугиан (Ornstein–Uhlenbeck process), Parameter space noise буюу параметр шуугиан гэсэн гурван шуугианы харьцуулалтыг хийлээ. Мөн үр дүнг хоорондоо харьцуулагдах боломжтой болгохын тулд параметр шуугианыг дангаар нь ашиглахгүйгээр Ornstein-Uhlenbeck-ын процесстэй цуг ашигласан.

Доорх графикийн хувьд үр дүнг гаргахдаа 101 episode ажиллуулсан бөгөөд нэг episode-д хийх алхам буюу MAX-STEP нь 10000 байна. Гүйцэтгэлийг үнэлэхийн тулд эхний 20 episode-ыг ямар ч шуугиангүйгээр ажиллуулсан.



Зураг 3.8: Харьцуулалтын график

Графикаас харахад бие биенээсээ хамааралгүй шуугианы хувьд дундаж шагнал нь харьцангуй бага байна. Бие биетэйгээ хамааралтай шуугианы хувьд савалгаа бага, дундаж шагналын

хувьд бие биенээсээ хамааралгүй шуугианаас их байна. Параметр шуугианы хувьд дундаж шагнал бусад шуугианаас харьцангуй их байна. Үүнээс дүгнэхэд гүйцэтгэлийн хувьд параметр шуугиан нь бага зэрэг илүү байна. Мөн үүнийг батлах өөр нэг үр дүн бол Зураг 3.6, 3.7 бөгөөд уг зурагт дундаж шагналын хамгийн дээд хэмжээ нь 200+ байна. Бусад шуугианы хувьд дундаж шагнал ийм хэмжээнд хүрээгүй. BipedalWalker-V3 орчин дээр роботын алхах чадварын хувьд параметр шуугианы хувьд илүү байсан. Гэхдээ параметр шуугианы хувьд илт давуу сайн биш байгаа бөгөөд хүлээлтэд хүрээгүй үр дүн гарсан нь хэрэгжүүлэлтийн үед алдаа гарсан байж болзошгүй эсвэл хипер параметрууд сайн таараагүй гэж харж байна.

# Дүгнэлт

Энэхүү судалгааны ажлаар 3 төрлийн шуугианыг DDPG алгоритмыг ашиглан BipedalWalker-V3 орчин дээр амжилттай туршиж гүйцэтгэлээ. Бие биенээсээ хамааралгүй болон бие биетэйгээ хамааралтай шуугианы хувьд үр дүнг хангалттай сайн гарсан гэж харж байгаа бол параметр шуугианы хувьд үр дүн хангалттай сайн сайн биш ч бүр муу үр дүн гараагүй гэж харж байна. Үр дүнгээс үзэхэд шуугианыг нэмэх хамгийн сайн арга бол үйлдэлд шууд шуугиан нэмэх бус неорал сүлжээнд параметр байдлаар шуугианыг нэмэх нь илүү гэдгийг харж болохоор байна. Роботын алхах чадвараас харахад параметр шуугиан нь орчны асуудлыг илүү сайн шийдвэрлэж чадаж байгааг харж болохоор байсан. Яагаад параметр шуугиан нь илүү байна вэ гэхээр шуугианыг нэмэхдээ тухайн нөхцөл байдлаас хамааруулан шинэчлэн нэмж байгаа учраас гэж бодож байна.

Хэрэгжүүлэлтийн явцад гаргасан нэг алдаа бол сургасан моделиудаа хадгалдаг, хадгалсан моделио уншиж аваад үргэлжлүүлэн сургадаг үйлдлийг параметр шуугианыг хэрэгжүүлснийхээ дараа хийх бус анх DDPG алгоритмаа хэрэгжүүлэхдээ цуг хийх байсан юм. Үүнийг эрт хэрэгжүүлээгүйгээс болж бие биенээсээ хамааралгүй шуугиан болон бие биетэйгээ хамааралтай шуугианы моделийг хадгалан аваагүй учир харьцуулалт хийхэд энэ хоёр шуугианыг дахин сургах асуудал гарсан.



# 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/](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/>
- [7] Parameter space noise for exploration, Arxiv, <https://arxiv.org/abs/1706.01905>
- [8] Benchmarking Deep Reinforcement Learning for Continuous Control, Arxiv, <https://arxiv.org/abs/1604.06778>
- [9] Policy Gradient Algorithms. 2018., LilianWeng, <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>

# A. КОДЫН ХЭРЭГЖҮҮЛЭЛТ

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import numpy as np
5
6 def fanin_init(size, fanin=None):
7     fanin = fanin or size[0]
8     v = 1. / np.sqrt(fanin)
9     return torch.Tensor(size).uniform_(-v, v)
10
11 class Actor(nn.Module):
12     def __init__(self, state_dimension, action_dimension, action_max):
13         super(Actor, self).__init__()
14
15         self.state_dimension = state_dimension
16         self.action_dimension = action_dimension
17         self.action_max = action_max
18
19         # 3 layer uusgene
20
21         self.fc1 = nn.Linear(state_dimension, 256)
22         self.fc2 = nn.Linear(256, 128)
23         self.fc3 = nn.Linear(128, 64)
24         self.fc4 = nn.Linear(64, action_dimension)
25
26         self.fc1.weight.data = fanin_init(self.fc1.weight.data.size())
27         self.fc2.weight.data = fanin_init(self.fc2.weight.data.size())
28         self.fc3.weight.data = fanin_init(self.fc3.weight.data.size())
29         self.fc4.weight.data.uniform_(-0.003, 0.003)
30
31     def forward(self, state):
32         output = self.fc1(state)
33         output = F.relu(output)
34
35         output = self.fc2(output)
36         output = F.relu(output)
37
38         output = self.fc3(output)
39         output = F.relu(output)
40
41         action = self.fc4(output)
42         action = F.tanh(action)
43
44         action = action * self.action_max
45
46         # action butsaana
47
48     return action
```

```

49 class Critic(nn.Module):
50     def __init__(self, state_dimension, action_dimension):
51         super(Critic, self).__init__()
52
53         self.state_dimension = state_dimension
54         self.action_dimension = action_dimension
55
56         # 4 layer uusgene
57
58         self.fcs1 = nn.Linear(state_dimension, 256)
59         self.fcs2 = nn.Linear(256, 128)
60         self.fca1 = nn.Linear(action_dimension, 128)
61         self.fc2 = nn.Linear(256, 128)
62         self.fc3 = nn.Linear(128, 1)
63
64         self.fcs1.weight.data = fanin_init(self.fcs1.weight.data.size()
65         )
66         self.fcs2.weight.data = fanin_init(self.fcs2.weight.data.size()
67         )
68         self.fca1.weight.data = fanin_init(self.fca1.weight.data.size()
69         )
70         self.fc2.weight.data = fanin_init(self.fc2.weight.data.size())
71         self.fc3.weight.data.uniform_(-0.003, 0.003)
72
73     def forward(self, state, action):
74         s1 = self.fcs1(state)
75         s1 = F.relu(s1)
76         s2 = self.fcs2(s1)
77         s2 = F.relu(s2)
78         a1 = self.fca1(action)
79         a1 = F.relu(a1)
80
81         output = torch.cat((s2, a1), dim=1)
82         output = self.fc2(output)
83         output = F.relu(output)
84         q_value = self.fc3(output)
85
86         # Q value-g butsaana
87
88         return q_value

```

Код A.1: models.py

```

1 import numpy as np
2 from collections import deque
3 import random
4
5 # experience-uudiig hadgalah buffer
6
7 class ReplayBuffer:
8     def __init__(self, size):

```

```

9         self.size = size
10        self.buffer = deque(maxlen=self.size)
11
12        # experience-g buffer-t nemeh
13
14        def add(self, state, action, reward, next_state):
15            exp = (state, action, reward, next_state)
16            self.buffer.append(exp)
17
18        # Randor-oor size-toonii experience-g awah
19
20        def sample_exp(self, size):
21            batch = []
22            size = min(size, len(self.buffer))
23            batch = random.sample(self.buffer, size)
24
25            states = np.float32([arr[0] for arr in batch])
26            actions = np.float32([arr[1] for arr in batch])
27            rewards = np.float32([arr[2] for arr in batch])
28            next_states = np.float32([arr[3] for arr in batch])
29
30            return states, actions, rewards, next_states

```

Код A.2: memory.py

```

1 import numpy as np
2
3 #Ornstein-uhlenbeck process-g correlated noise uusgehed ashiglana
4
5 #http://math.stackexchange.com/questions/1287634/implementing-ornstein-
6   uhlenbeck-in-matlab
7
8 class OrnsteinUhlenbeckActionNoise:
9
10     def __init__(self, action_dim, mu = 0, theta = 0.15, sigma = 0.2):
11         self.action_dim = action_dim
12         self.mu = mu
13         self.theta = theta
14         self.sigma = sigma
15         self.X = np.ones(self.action_dim) * self.mu
16
17     def reset(self):
18         self.X = np.ones(self.action_dim) * self.mu
19
20     def sample(self):
21         dx = self.theta * (self.mu - self.X)
22         dx = dx + self.sigma * np.random.randn(len(self.X))
23         self.X = self.X + dx
24         return self.X
25
26 class ActionNoise(object):
27     def reset(self):

```

```

27         pass
28
29 #Based on https://github.com/openai/baselines/tree/master/baselines/
    ddp
30 class OrnsteinUhlenbeckActionNoiseBaseline(ActionNoise):
31     def __init__(self, mu, sigma, theta=.15, dt=1e-2, x0=None):
32         self.theta = theta
33         self.mu = mu
34         self.sigma = sigma
35         self.dt = dt
36         self.x0 = x0
37         self.reset()
38
39     def __call__(self):
40         x = self.x_prev + self.theta * (self.mu - self.x_prev) * self.
            dt + self.sigma * np.sqrt(self.dt) * np.random.normal(size=
            self.mu.shape)
41         self.x_prev = x
42         return x
43
44     def reset(self):
45         self.x_prev = self.x0 if self.x0 is not None else np.zeros_like
            (self.mu)
46
47     def __repr__(self):
48         return 'OrnsteinUhlenbeckActionNoise(mu={}, sigma={})'.format(
            self.mu, self.sigma)
49
50 class AdaptiveParamNoiseSpec(object):
51     def __init__(self, initial_stddev=0.1, desired_action_stddev=0.2,
        adaptation_coefficient=1.01):
52         self.initial_stddev = initial_stddev
53         self.desired_action_stddev = desired_action_stddev
54         self.adaptation_coefficient = adaptation_coefficient
55
56         self.current_stddev = initial_stddev
57
58     def adapt(self, distance):
59         if distance > self.desired_action_stddev:
60             # Decrease stddev.
61             self.current_stddev /= self.adaptation_coefficient
62         else:
63             # Increase stddev.
64             self.current_stddev *= self.adaptation_coefficient
65
66     def get_stats(self):
67         stats = {
68             'param_noise_stddev': self.current_stddev,
69         }
70         return stats
71
72     def __repr__(self):

```

```

73         fmt = 'AdaptiveParamNoiseSpec(initial_stddev={},
74             desired_action_stddev={}, adaptation_coefficient={})'
75         return fmt.format(self.initial_stddev, self.
76             desired_action_stddev, self.adaptation_coefficient)
77
78 if __name__ == '__main__':
79     ou = OrnsteinUhlenbeckActionNoise(1)
80     ou_baseline = OrnsteinUhlenbeckActionNoiseBaseline(mu=np.zeros(1),
81         sigma=float(0.2) * np.ones(1))
82
83     states = []
84     states2 = []
85
86     for i in range(1000):
87         states.append(ou.sample())
88         states2.append(ou_baseline())
89
90     import matplotlib.pyplot as plt
91
92     plt.plot(states, label = "OU")
93     plt.plot(states2, label = "OU Baseline")
94     plt.legend()
95     plt.show()

```

Код A.3: utilities.py

```

1  import gc
2  import random
3  import gym
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from torch.autograd import Variable
7  import torch
8  import torch.nn.functional as F
9  import pickle
10 import os
11 import math
12 import time
13
14 import memory
15 import models
16 import utilities
17
18 if __name__ == "__main__":
19
20     ACTOR_LEARNING_RATE = 0.0001
21     CRITIC_LEARNING_RATE = 0.001
22     MAX_EPISODE = 1201
23     MAX_STEPS = 1600
24     NOISE_TYPE = "ParameterWithOU" #OU, OUBaseline, Parameter,
25         Uncorrelated, NoNoise, ParameterWithOU
26     # Orchin beldeh

```

```

26 env = gym.make('BipedalWalker-v3')
27
28 state_dimension = env.observation_space.shape[0]
29 action_dimension = env.action_space.shape[0]
30 action_max = env.action_space.high[0]
31
32 print("State dimension: {}".format(state_dimension))
33 print("Action dimension: {}".format(action_dimension))
34 print("Action max: {}".format(action_max))
35
36 load_models = False
37
38 # Actor network, critic network uusgeh
39
40 actor = models.Actor(state_dimension, action_dimension, action_max)
41 target_actor = models.Actor(state_dimension, action_dimension,
42                             action_max)
43 actor_optimizer = torch.optim.Adam(actor.parameters(), lr=
44                                     ACTOR_LEARNING_RATE)
45
46 critic = models.Critic(state_dimension, action_dimension)
47 target_critic = models.Critic(state_dimension, action_dimension)
48 critic_optimizer = torch.optim.Adam(critic.parameters(), lr=
49                                     CRITIC_LEARNING_RATE)
50
51 # Target network-g huulah
52
53 for target_param, param in zip(target_actor.parameters(), actor.
54                                 parameters()):
55     target_param.data.copy_(param.data)
56
57 for target_param, param in zip(target_critic.parameters(), critic.
58                                 parameters()):
59     target_param.data.copy_(param.data)
60
61 # Hadgalsan modeliig ashiglah
62
63 if load_models:
64     actor.load_state_dict(torch.load('./Models/' + str(0) + '_actor
65                                     .pt'))
66     critic.load_state_dict(torch.load('./Models/' + str(0) + '
67                                     _critic.pt'))
68
69     for target_param, param in zip(target_actor.parameters(), actor
70                                     .parameters()):
71         target_param.data.copy_(param.data)
72
73     for target_param, param in zip(target_critic.parameters(),
74                                     critic.parameters()):
75         target_param.data.copy_(param.data)
76
77 print("Models loaded!")

```

```

69
70 # Replay buffer uusgeh
71
72 ram = memory.ReplayBuffer(1000000)
73
74 #Buffer-g utgaar duurgeh (hot start)
75
76 st = np.float32(env.reset())
77 print(type(st))
78 for step in range(128):
79     action = env.action_space.sample()
80     new_observation, reward, done, info = env.step(action)
81
82     # Replay buffer-d state, action, reward, new_state -g hadgalah
83
84     ram.add(st, action, reward, new_observation)
85
86     if done:
87         st = env.reset()
88     else:
89         st = new_observation
90
91 print("Initial ram size: ", len(ram.buffer))
92
93 # Reward-iig hadgalah list
94
95 reward_list = []
96 average_reward_list = []
97 steps_reward_list = []
98
99 # Noise uusgeh
100
101 if NOISE_TYPE == "OU":
102     noise = utilities.OrnsteinUhlenbeckActionNoise(action_dimension
103 )
104 elif NOISE_TYPE == "OUBaseline":
105     noise = utilities.OrnsteinUhlenbeckActionNoiseBaseline(mu=np.
106         zeros(action_dimension), sigma=float(0.2))
107 elif NOISE_TYPE == "Parameter" or NOISE_TYPE == "ParameterWithOU":
108     #Parameter noise-d zoriulsan actor
109     actor_copy = models.Actor(state_dimension, action_dimension,
110         action_max)
111
112     #Parameter noise
113     parameter_noise = utilities.AdaptiveParamNoiseSpec(
114         initial_stddev=0.05,desired_action_stddev=0.3,
115         adaptation_coefficient=1.05)
116     noise = utilities.OrnsteinUhlenbeckActionNoise(action_dimension
117 )
118     #noise = utilities.OrnsteinUhlenbeckActionNoiseBaseline(mu=np.
119         zeros(action_dimension), sigma=float(0.2))
120

```



```

114     print("Noise type: ", NOISE_TYPE)
115
116     start_time = time.time()
117
118     tmp_noise_type = NOISE_TYPE
119
120     for ep in range(MAX_EPISODE):
121
122         # Anhnii state-g awah
123
124         observation = env.reset()
125
126         ep_reward = 0
127         step_cntr = 0
128
129         #Ehnii 20 episode uncorrelated noise-toigoor ywna
130         if ep < 20:
131             NOISE_TYPE = "NoNoise"
132         else:
133             NOISE_TYPE = tmp_noise_type
134
135         if NOISE_TYPE == "Parameter":
136             #Actor-g actor_copy-d huulah
137
138             for target_param, param in zip(actor_copy.parameters(),
139                                           actor.parameters()):
140                 target_param.data.copy_(param.data)
141
142             # Parameter noise-iig neural suljeen deer nemeh
143
144             parameters = actor_copy.state_dict()
145             for name in parameters:
146                 parameter = parameters[name]
147                 rand_number = torch.randn(parameter.shape)
148                 parameter = parameter + rand_number * parameter_noise.
149                     current_stddev
150
151         for step in range(MAX_STEPS):
152             env.render()
153             state = np.float32(observation)
154
155             # Action-g songoh
156
157             tmp_state = Variable(torch.from_numpy(state))
158             action_without_noise = actor.forward(tmp_state).detach()
159
160             if NOISE_TYPE == "NoNoise":
161                 action = np.clip(action_without_noise.data.numpy(),
162                                 -1., 1.)
163             elif NOISE_TYPE == "OU":
164                 #OU processiin noisetoi action

```

```

163         action = np.clip(action_without_noise.data.numpy() + (
164             noise.sample() * action_max), -1., 1.)
165     elif NOISE_TYPE == "OUBaseline":
166         #OU Baseline noisetoi action
167         action = np.clip(action_without_noise.data.numpy() +
168             noise(), -1., 1.)
169     elif NOISE_TYPE == "Parameter":
170         action = actor_copy.forward(tmp_state).detach().numpy()
171     elif NOISE_TYPE == "ParameterWithOU":
172         noise.reset()
173         action_with_parameter_noise = actor_copy.forward(
174             tmp_state).detach()
175         #Parameter noisetoi action
176         action = action_with_parameter_noise.numpy() + (noise.
177             sample() * action_max)
178         #action = np.clip(action_with_parameter_noise.numpy() +
179             noise(), -1., 1.)
180     elif NOISE_TYPE == "Uncorrelated":
181         #[-0.2, 0.2] random noisetoi action
182         action = np.clip(action_without_noise.data.numpy() + (
183             np.random.uniform(-0.2,0.2) * action_max), -1., 1.)
184     else:
185         raise RuntimeError('Buruu turliin noise: "{}".format(
186             NOISE_TYPE))
187
188     # Action-g hiiij shine state, reward awah
189
190     new_observation, reward, done, info = env.step(action)
191
192     if done:
193         new_state = None
194     else:
195         new_state = np.float32(new_observation)
196
197         # Replay buffer-d state, action, reward, new_state -g
198         # hadgalah
199
200         ram.add(state, action, reward, new_state)
201         ep_reward += reward
202         steps_reward_list.append(reward)
203
204     observation = new_observation
205
206     # Replay buffer-aas 128 bagts turshalagiig random-oor awna
207
208     states, actions, rewards, next_states = ram.sample_exp(128)
209
210     states = Variable(torch.from_numpy(states))
211     actions = Variable(torch.from_numpy(actions))
212     rewards = Variable(torch.from_numpy(rewards))
213     next_states = Variable(torch.from_numpy(next_states))

```

```

207     # Critic network-g surgah
208
209     predicted_action = target_actor.forward(next_states).detach
210     ()
211     next_val = torch.squeeze(target_critic.forward(next_states,
212     predicted_action).detach())
213     y_expected = rewards + 0.99*next_val
214     y_predicted = torch.squeeze(critic.forward(states, actions)
215     )
216
217     # Critic network-g shinechleh, critic loss-g tootsooloh
218
219     critic_loss = F.smooth_l1_loss(y_predicted, y_expected)
220     critic_optimizer.zero_grad()
221     critic_loss.backward()
222     critic_optimizer.step()
223
224     # Actor network-g surgah
225
226     predicted_action = actor.forward(states)
227     actor_loss = -1*torch.sum(critic.forward(states,
228     predicted_action))
229     actor_optimizer.zero_grad()
230     actor_loss.backward()
231     actor_optimizer.step()
232
233     # Target network-g shinechleh
234
235     for target_param, param in zip(target_actor.parameters(),
236     actor.parameters()):
237         target_param.data.copy_(target_param.data * (1.0 -
238         0.001) + param.data * 0.001)
239
240     for target_param, param in zip(target_critic.parameters(),
241     critic.parameters()):
242         target_param.data.copy_(target_param.data * (1.0 -
243         0.001) + param.data * 0.001)
244
245     if done:
246         break
247
248     step_cntr += 1
249
250 if NOISE_TYPE == "Parameter":
251     #Noisetoi actor deer hiigdsen data-g list-d hadgalj awaad
252     suuliin episode-d hiigdsen stepiin toogoor datagaa awna
253
254     noise_data_list = list(ram.buffer)
255     noise_data_list = np.array(noise_data_list[-step_cntr:])
256
257     actor_copy_state, actor_copy_action, _, _ = zip(*
258     noise_data_list)

```

```

249
250     #Noisetoi actoriin action
251     actor_copy_actions = np.array(actor_copy_action)
252
253     #Engiin actoriin action
254
255     actor_actions = []
256     for state in np.array(actor_copy_state):
257         state = Variable(torch.from_numpy(state))
258         action = actor.forward(state).detach().numpy()
259         actor_actions.append(action)
260
261     #Distance tootsoh
262     diff_actions = actor_copy_actions - actor_actions
263     mean_diff_actions = np.mean(np.square(diff_actions),axis=0)
264     distance = math.sqrt(np.mean(mean_diff_actions))
265
266     #Sigma-g update hiih
267     parameter_noise.adapt(distance)
268
269     # reward-g hadgalj awna
270
271     reward_list.append(ep_reward)
272     average_reward = np.mean(reward_list[-40:])
273     print("Episode: {} Average Reward: {}".format(ep,
274         average_reward))
275     average_reward_list.append(average_reward)
276
277     #Model-iig hadgalah
278     if ep % 100 == 0 and ep != 0:
279         folder_path = './Models_ep1200_' + str(NOISE_TYPE) + '_' +
280             str(ACTOR_LEARNING_RATE) + '_' + str(
281                 CRITIC_LEARNING_RATE)
282         if not os.path.exists(folder_path):
283             os.makedirs(folder_path)
284
285         torch.save(target_actor.state_dict(), folder_path + '/' +
286             str(ep) + '_actor.pt')
287         torch.save(target_critic.state_dict(), folder_path + '/' +
288             str(ep) + '_critic.pt')
289
290     #Ram hadgalah
291     file_name = folder_path + '/' + str(ep) + '_ram.deque'
292     open_file = open(file_name, "wb")
293     pickle.dump(ram.buffer, open_file)
294     open_file.close()
295
296     #Average reward list hadgalah
297     file_name = folder_path + '/' + str(ep) + '_average_rewards
298         .list'
299     open_file = open(file_name, "wb")
300     pickle.dump(average_reward_list, open_file)

```

```

295         open_file.close()
296
297         #Step bolgonii rewardii hadgalah
298         file_name = folder_path + '/' + str(ep) + '_step_rewards.
                list'
299         open_file = open(file_name, "wb")
300         pickle.dump(steps_reward_list, open_file)
301         open_file.close()
302
303         print("Target actor, critic models saved")
304
305         gc.collect()
306
307         execution_time = time.time() - start_time
308         file_name = './Models_ep1200_' + str(NOISE_TYPE) + '_' + str(
                ACTOR_LEARNING_RATE) + '_' + str(CRITIC_LEARNING_RATE) + '/' +
                str(ep) + '_execution_time.sec'
309         open_file = open(file_name, "wb")
310         pickle.dump(execution_time, open_file)
311         open_file.close()
312         # Reward-g durslen haruulah
313
314         print("Reward max: ", max(average_reward_list))
315
316         # plt.plot(average_reward_list, label = NOISE_TYPE)
317         # plt.legend()
318         # plt.xlabel("Episode")
319         # plt.ylabel("Average Episode Reward")
320         # plt.show()
321
322         #os.system("shutdown /s /t 1")

```

Код A.4: main.py