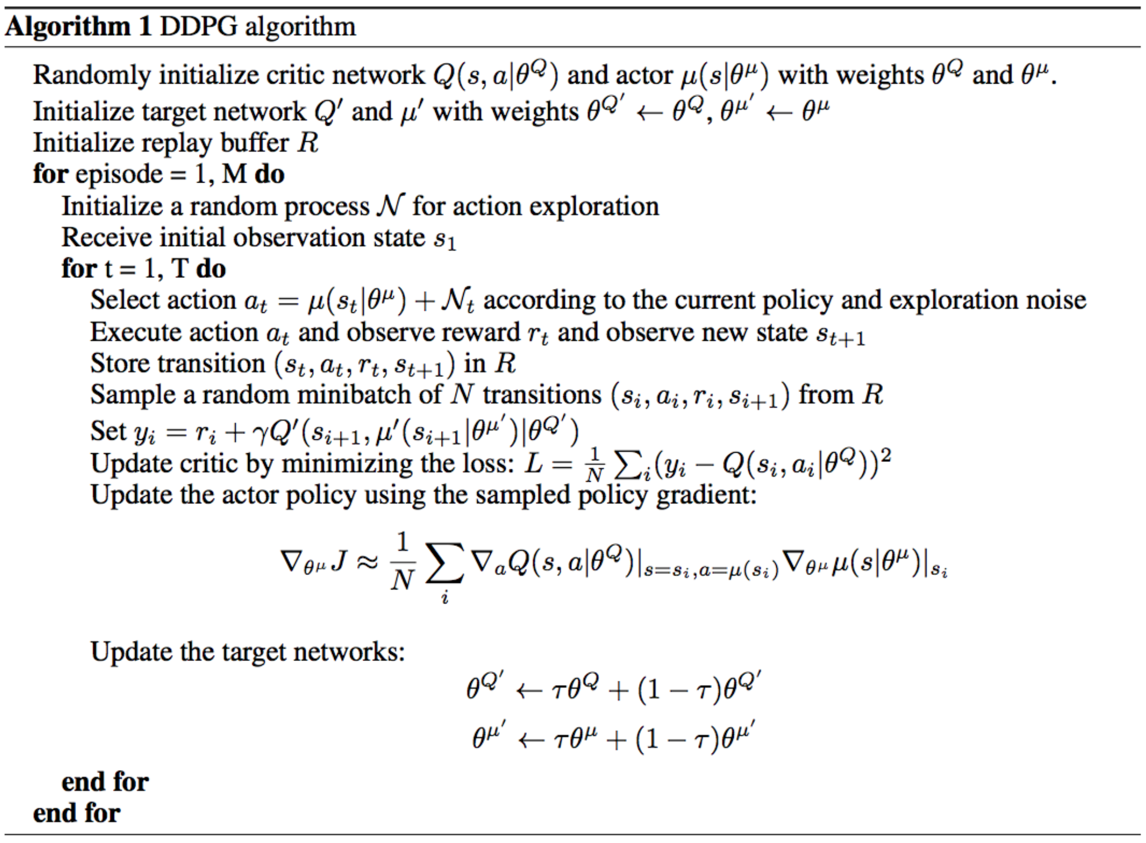
1 DDPG

https://arxiv.org/pdf/1509.02971.pdf



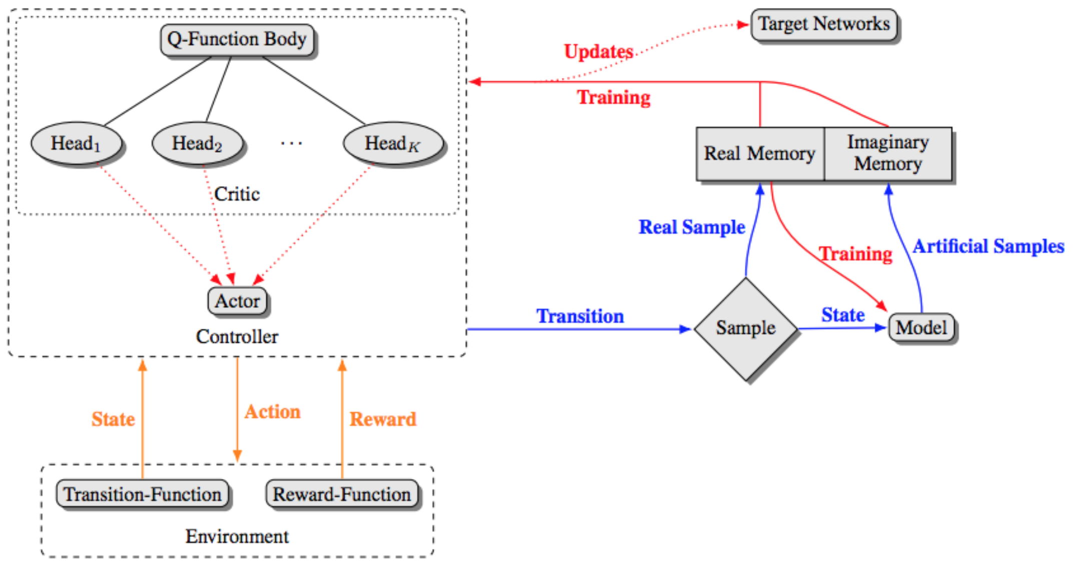
Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
end for
```

2 Uncertainty-driven Imagination for Continuous Deep Reinforcement Learning

http://proceedings.mlr.press/v78/kalweit17a/kalweit17a.pdf

この論文が言いたいこと: DDPG の通常の actor, critic の他に, モデルを推定するネットワークも学習させ, そのモデルを使って imaginary なデータを水増しするとうまくいくよ. モデルフリーの DDPG ではサンプル数がたくさん必要なのを解決し, かつモデルベースのような厳密なモデルを必要としないところが良さ.



2.1 Model assisted DDPG

環境のモデル (state, action → state) を学習させて, 実際のサンプル s_t から始まる Dstep の B 個のデータを imagenary memory に入れて普通の DDPG のように Q 関数を学習させる. 各データの action は今の actor ＋ random noise で決定する.(なので,Model, actor 共に deterministic だが,B 個のデータ達は別々のデータになる)

2.2 Including uncertainty

Deep Exploration via Bootstrapped DQN (`https://arxiv.org/pdf/1602.04621.pdf`) を DDPG でも使ってみた.

不明な点：network の構造. we did not use shared body と書いてあるということは別々のネットワークを使っている？ (input だけ同じであとは別々の MLP を持っている？)

1. 一つのサンプルにつき, どの head を使うかのマスクを作る (実験ではベルヌーイ分布).→ K 個の bootstrapped sample ができたことになる
2. それぞれのサンプルについて, それぞれの head が学習
   - $y_t^{Q_{1:K}} \leftarrow r_t + \gamma Q'_{1:K}(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'_{1:K}})$
   - $L(\theta^{Q_{1:K}}) = \sum_k \frac{1}{n_k} \sum_i m_i^k (y_i^k - Q^k(s_i, a_i|\theta^{Q_k}))^2$
3. actor が全ての head を使って学習
   - $\nabla_{\theta^\mu}\mathrm{R} \leftarrow \frac{1}{\mathrm{K}} \sum_\mathrm{k} \frac{1}{\mathrm{n}} \sum_\mathrm{i} \nabla_\mathrm{a} \mathrm{Q}^\mathrm{k}(\mathrm{s_i}, \mathrm{a}|\theta^{\mathrm{Q_k}}) \ |_{\mathrm{a}=\mu(\mathrm{s_i})} \nabla_{\theta^\mu}\mu(\mathrm{s_i}|\theta^\mu)$
4. ターゲットを更新する

Q function の分布の分散で,uncertainty を推定する

2.3 Limiting Model Usage

Uncertainty が高い時にだけ Imagination を使いたい. 分散を使いたいが, 分散は task specific で分散の前提知識は使いたくないので, 今までの分散の最大を使う. $P[b \in \mathcal{I}] = \frac{\sigma_e^2}{\max_E \sigma_{E<e}^2}$ の確率で Imagination を使う. σ_e^2 は episode e の mean variance. 学習が進むにつれ,uncertainty が下がるので, 下がってきたら Imaginary batch はあまり使わないようにする.

```
Algorithm 1: MA-BDDPG
initialize critic $Q$ and actor $\mu$, targets $Q'$ and $\mu'$ and model $\mathcal{M}^\Delta$
initialize replay buffer $\mathcal{R}$ and imaginary replay buffer $\mathcal{I}$
$\hat{\sigma}_1^2 = 0$
for $e = 1..E$ do
    get initial state $s_1$
    for $t = 1..T$ do
        apply action $a_t = \mu(s_t|\theta^\mu) + \xi$ and observe variance $\sigma_t^2(Q)$
        observe $s_{t+1}$ and $r_t$
        generate random mask $\mathbf{m}_t$
        save transition $(s_t, a_t, s_{t+1}, r_t, \mathbf{m}_t)$ in $\mathcal{R}$
        generate $B$ random imaginary transitions of length $D$ starting from $s_t$ using $\mathcal{M}^\Delta$
        store the imaginary transitions in $\mathcal{I}$
        for $u_\mathcal{M} = 1..U_\mathcal{M}$ do
            train $\mathcal{M}^\Delta$ on minibatch from $\mathcal{R}$
        for $u_\mathcal{R} = 1..U_\mathcal{R}$ do
            train $Q$ and $\mu$ on minibatch from $\mathcal{R}$
            adjust parameters of $Q'$ and $\mu'$
        for $u_\mathcal{I} = 1..U_\mathcal{I}$ do
            generate random number $rand \in [0, 1]$
            if $rand < \hat{\sigma}_e^2$ then
                train $Q$ and $\mu$ on minibatch from $\mathcal{I}$
                adjust parameters of $Q'$ and $\mu'$
    add variance values in $e$ and set maximum variance seen so far
    set current variance ratio $\hat{\sigma}_{e+1}^2 = \frac{\bar{\sigma}_e^2}{\max_E \bar{\sigma}_{E<e}^2}$, where $\bar{\sigma}_e^2$ denotes the mean variance in $e$
```

2.4 実験結果

Imaginary replay buffer の大きさは小さくして, 過去数 episode のものだけを使った方が安定する. Pusher があまり良くないのは,pusher は物体に手を当てる → 物体を目的地まで動かす, という二つの動作が必要だが, 始めは物体に手を当てる部分を頑張るので, モデルの学習の時に物体を目的地まで動かす部分については何も学習しないから. もう一つの理由が Pusher の可動範囲が狭すぎて学習が十分に進んでいなくても σ が十分小さくなってしまうから.

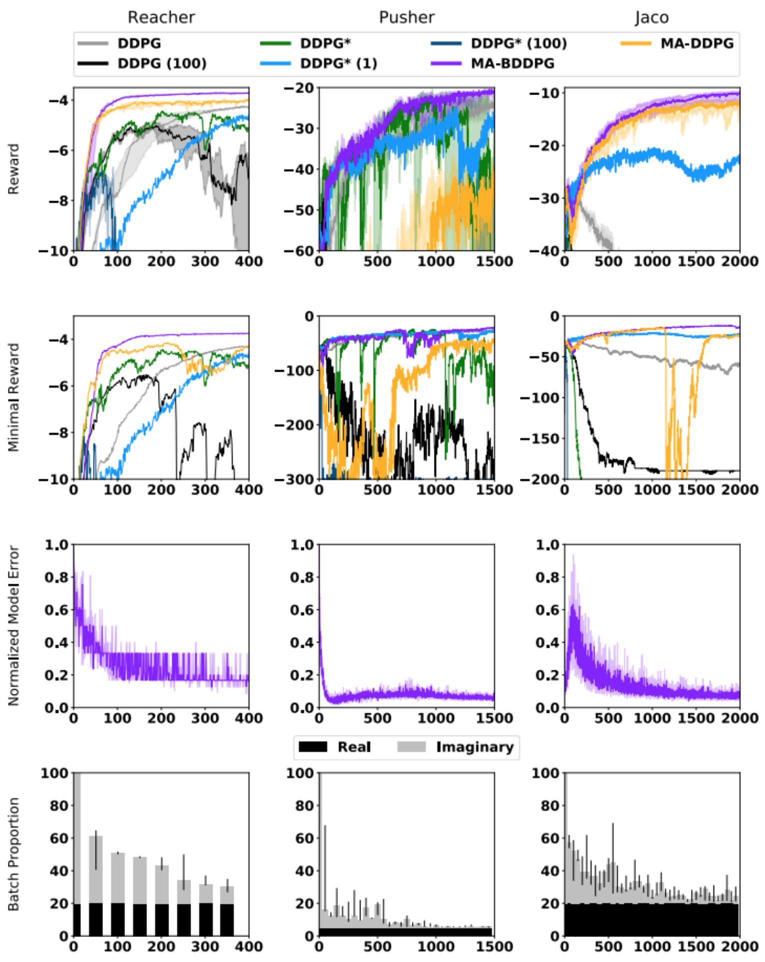


Figure 3: Results of environments. Top row shows median performance and interquartile ranges. The second row shows the worst occuring reward values across all runs. The third row shows the error in prediction over time, normalized to [0, 1]. The last row shows the ratio of real and imaginary batches used by MA-BDDPG. Numbers in parentheses denote differing update values.

	MA-BDDPG	DDPG (100)	DDPG* (100)	DDPG*
Reacher	**243 (-3.74)**	188 (-5.02)	64 (-6.866)	203 (-4.432)
Pusher	**1072 (-20.893)**	33 (-47.118)	0 (-58.378)	957 (-22.014)
Jaco	**1999 (-9.53)**	18 (-32.262)	0 (-28.21)	17 (-32.248)

Table 1: Best results returned by DDPG and DDPG* with 100 updates per step, as well as best results from DDPG* and equal real updates as MA-BDDPG.

	MA-BDDPG	DDPG	speed up ×	DDPG* (1)	speed up ×
Reacher	**243 (-3.74)**	3000 (-3.848)	12.35	3000 (-3.885)	12.35
Pusher	**1072 (-20.893)**	1902 (-20.579)	1.77	3455 (-20.941)	3.22
Jaco	**1999 (-9.53)**	30000 (-11.816)	15.00	22245 (-9.986)	11.13

Table 2: Median episodes until success of all runs. Next to episodes are rewards. In case there was no success within the considered time frame, the amount was set to the maximal episode, together with the best reward found after that time. Reacher was solved if there was a reward > -3.75, as defined in OpenAI Gym. For Pusher we defined success to get a reward > -21 and for Jaco > -10.

他の DDPG をモデルベースの力を借りてよくするもの
`https://kaigi.org/jsai/webprogram/2017/pdf/1152.pdf`

3 The Intentional Unintentional Agent: Learning to Solve Many Continuous Control Tasks Simultaneously

`https://arxiv.org/pdf/1707.03300.pdf`
この論文の言いたいこと: 一つのタスクに対して学習させるときに, 色々なタスクについても評価して学習させたら, 一つのタスクを学習させるよりもうまくいくよ.

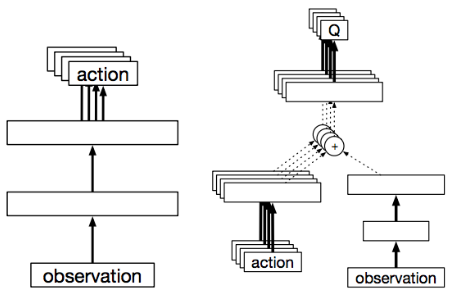


Figure 1: The IU architecture. The actor network on the left consists of two shared MLP-tanh layers followed by non-shared MLP-tanh layers to produce the multivariate actions for each policy (4 policies in this diagram). The right hand side shows the critic network. The action vectors provided by the policies are fed into a non-shared MLP-tanh layer, which is then point-wise added to the ouput of a two layer MLP-tanh network applied to the observation. The resulting activations are processed by non-shared linear layers to produce the Q values.

3.1 問題設定

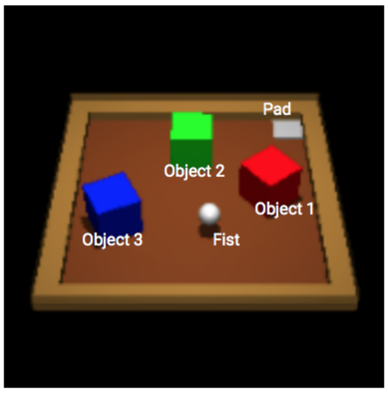


Figure 2: Example of the playroom environment showing three objects and the velocity controlled actuator (fist). Also shown is a goal pad location in the upper right corner.

- $p_{\mathrm{blue}}(o)$ は,0, 1 の o が blue かどうかの値.
- $b_{\mathrm{near}(o_1,o_2,s)}$ が,0, 1 の o_1 か o_2 が近いかどうかの値 (近さはパラメーターで定義)
- $r_{\mathrm{red\_near\_blue}}{=}\sum_{o_1,o_2} \mathrm{p_{red}}(o_1)\mathrm{p_{blue}}(o_2)\mathrm{b_{near}}(o_1,o_2,s)$
- $r_{\mathrm{red\_near\_blue}} = b_{\mathrm{near}}(\mathrm{red}, \mathrm{blue}, \mathrm{s})$ は red と blue が近いかどうかの reward になる.

タスクは, 近い, 遠い, 東西南北の 6 種類の b と,object(+fist, pad) のペアの組み合わせがある.
不明な点：observation は何なのか？
不明な点 2: reward は 0,1 なのに average reward が-1 から 1 の範囲なのはなぜ？

3.2　結論

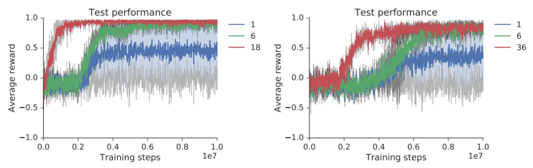


Figure 3: **Left**: Test performance for the task of gathering two blocks together in the playroom, using a varying number of additional tasks. **Right**: What happens when an extra block is added to the environment, causing significant physical interference. In both cases, the more tasks the IU agent solves simultaneously, the faster it learns the intentional task.

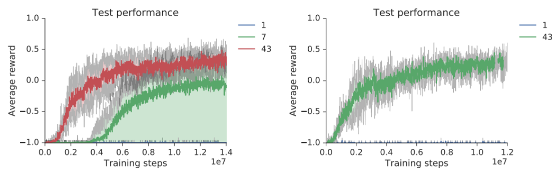


Figure 5: **Left**: Test performance for the task of moving three blocks to the corner pad of the playroom. **Right**: Same task after increasing the length of the playroom sides by 50%. When the length is doubled the task can not be learned at all. For these very hard exploration task, the IU agent performs reasonably, while DDPG completely fails.

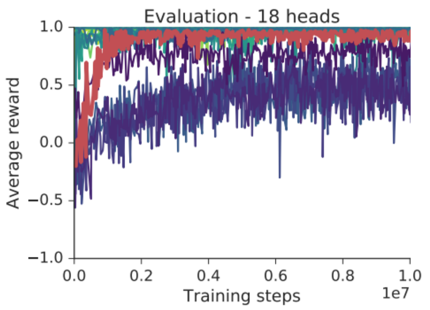


Figure 4: Test performance of the 18 policies 17 of which are inattentional and hence trained off-policy, with the intentional one shown in red, learned simultaneously in the experiment corresponding to Figure 3 (left). All 18 tasks are solved simultaneously.

難しいタスクの actor に従ってデータを sampling すれば, 全てのタスクを学習することができる．普通の DDPG では学習できないようなタスク (報酬が sparse だから？) も, この仕組みを使えば学習できる．ランダムなタスクに従ったり,reward がもらえたら従うタスクを変更したりしたが, 難しいタスクに従うのが一番よかった．理由は簡単なタスクを解くための行動で replay memory がいっぱいになると, 複雑なタスクが学習できないから．