Plan for today

• Finish DQN
• Briefly talk about Rainbow DQN
• DDPG
• BCQ
• Breakout room discussion
DDPG, BCQ

Saurabh Gupta
DDPG

$$\rho^\pi(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi)$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} \left[ r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi} [Q^\pi(s_{t+1}, a_{t+1})] \right]$$

$$\nabla_{\theta \mu} J \approx \mathbb{E}_{s_t \sim \rho^\beta} \left[ \nabla_{\theta \mu} Q(s, a | \theta^Q) \bigg|_{s=s_t, a=\mu(s_t | \theta^\mu)} \right]$$

$$= \mathbb{E}_{s_t \sim \rho^\beta} \left[ \nabla_a Q(s, a | \theta^Q) \bigg|_{s=s_t, a=\mu(s_t)} \nabla_{\theta \mu} \mu(s | \theta^\mu) \bigg|_{s=s_t} \right]$$
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
DDPG

- Replay buffer
- Target network
- Co-related noise
DDPG

Published as a conference paper at ICLR 2016

Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.

Table 1: Performance after training across all environments for at most 2.5 million steps. We report both the average and best observed (across 5 runs). All scores, except Torcs, are normalized so that a random agent receives 0 and a planning algorithm 1; for Torcs we present the raw reward score. We include results from the DDPG algorithm in the low-dimensional (lowd) version of the environment and high-dimensional (pix). For comparison we also include results from the original DPG algorithm with a replay buffer and batch normalization (cntrl).

<table>
<thead>
<tr>
<th>environment</th>
<th>$R_{av,lowd}$</th>
<th>$R_{best,lowd}$</th>
<th>$R_{av,pix}$</th>
<th>$R_{best,pix}$</th>
<th>$R_{av,cntrl}$</th>
<th>$R_{best,cntrl}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>blockworld1</td>
<td>1.156</td>
<td>1.511</td>
<td>0.466</td>
<td>1.299</td>
<td>-0.080</td>
<td>1.260</td>
</tr>
<tr>
<td>blockworld3da</td>
<td>0.340</td>
<td>0.705</td>
<td>0.889</td>
<td>2.225</td>
<td>-0.139</td>
<td>0.658</td>
</tr>
<tr>
<td>canada</td>
<td>0.303</td>
<td>1.735</td>
<td>0.176</td>
<td>0.688</td>
<td>0.125</td>
<td>1.157</td>
</tr>
<tr>
<td>canada2d</td>
<td>0.400</td>
<td>0.978</td>
<td>-0.285</td>
<td>0.119</td>
<td>-0.045</td>
<td>0.701</td>
</tr>
<tr>
<td>cart</td>
<td>0.938</td>
<td>1.336</td>
<td>1.096</td>
<td>1.258</td>
<td>0.343</td>
<td>1.216</td>
</tr>
<tr>
<td>cartpole</td>
<td>0.844</td>
<td>1.115</td>
<td>0.482</td>
<td>1.138</td>
<td>0.244</td>
<td>0.755</td>
</tr>
<tr>
<td>cartpoleBalance</td>
<td>0.951</td>
<td>1.000</td>
<td>0.335</td>
<td>0.996</td>
<td>-0.468</td>
<td>0.528</td>
</tr>
<tr>
<td>cartpoleParalleldouble</td>
<td>0.549</td>
<td>0.900</td>
<td>0.188</td>
<td>0.323</td>
<td>0.197</td>
<td>0.572</td>
</tr>
<tr>
<td>cartpoleSerialDouble</td>
<td>0.272</td>
<td>0.719</td>
<td>0.195</td>
<td>0.642</td>
<td>0.143</td>
<td>0.701</td>
</tr>
<tr>
<td>cartpoleSerialTriple</td>
<td>0.736</td>
<td>0.946</td>
<td>0.412</td>
<td>0.427</td>
<td>0.583</td>
<td>0.942</td>
</tr>
<tr>
<td>cheetah</td>
<td>0.903</td>
<td>1.206</td>
<td>0.457</td>
<td>0.792</td>
<td>-0.008</td>
<td>0.425</td>
</tr>
<tr>
<td>fixedReacher</td>
<td>0.849</td>
<td>1.021</td>
<td>0.693</td>
<td>0.981</td>
<td>0.259</td>
<td>0.927</td>
</tr>
<tr>
<td>fixedReacherDouble</td>
<td>0.924</td>
<td>0.996</td>
<td>0.872</td>
<td>0.943</td>
<td>0.290</td>
<td>0.995</td>
</tr>
<tr>
<td>fixedReacherSingle</td>
<td>0.954</td>
<td>1.000</td>
<td>0.827</td>
<td>0.995</td>
<td>0.620</td>
<td>0.999</td>
</tr>
<tr>
<td>gripper</td>
<td>0.655</td>
<td>0.972</td>
<td>0.406</td>
<td>0.790</td>
<td>0.461</td>
<td>0.816</td>
</tr>
<tr>
<td>gripperRandom</td>
<td>0.618</td>
<td>0.937</td>
<td>0.082</td>
<td>0.791</td>
<td>0.557</td>
<td>0.808</td>
</tr>
<tr>
<td>hardCheetah</td>
<td>1.311</td>
<td>1.990</td>
<td>1.204</td>
<td>1.431</td>
<td>-0.031</td>
<td>1.411</td>
</tr>
<tr>
<td>hopper</td>
<td>0.676</td>
<td>0.936</td>
<td>0.112</td>
<td>0.924</td>
<td>0.078</td>
<td>0.917</td>
</tr>
<tr>
<td>hyq</td>
<td>0.416</td>
<td>0.722</td>
<td>0.234</td>
<td>0.672</td>
<td>0.198</td>
<td>0.618</td>
</tr>
<tr>
<td>movingGripper</td>
<td>0.474</td>
<td>0.936</td>
<td>0.480</td>
<td>0.644</td>
<td>0.416</td>
<td>0.805</td>
</tr>
<tr>
<td>pendulum</td>
<td>0.946</td>
<td>1.021</td>
<td>0.663</td>
<td>1.055</td>
<td>0.099</td>
<td>0.951</td>
</tr>
<tr>
<td>reacher</td>
<td>0.720</td>
<td>0.987</td>
<td>0.194</td>
<td>0.878</td>
<td>0.231</td>
<td>0.953</td>
</tr>
<tr>
<td>reacher3daFixedTarget</td>
<td>0.585</td>
<td>0.943</td>
<td>0.453</td>
<td>0.922</td>
<td>0.204</td>
<td>0.631</td>
</tr>
<tr>
<td>reacher3daRandomTarget</td>
<td>0.467</td>
<td>0.739</td>
<td>0.374</td>
<td>0.735</td>
<td>-0.046</td>
<td>0.158</td>
</tr>
<tr>
<td>reacherSingle</td>
<td>0.981</td>
<td>1.102</td>
<td>1.000</td>
<td>1.083</td>
<td>1.010</td>
<td>1.083</td>
</tr>
<tr>
<td>walker2d</td>
<td>0.705</td>
<td>1.573</td>
<td>0.944</td>
<td>1.476</td>
<td>0.393</td>
<td>1.397</td>
</tr>
<tr>
<td>torcs</td>
<td>-393.385</td>
<td>1840.036</td>
<td>-401.911</td>
<td>1876.284</td>
<td>-911.034</td>
<td>1961.600</td>
</tr>
</tbody>
</table>
Underlying physical model and its derivatives. We normalize scores so that the naive policy has a mean score of 0 and iLQG has a mean score of 1. DDPG is able to learn good policies on many of the tasks, and in many cases some of the replicas learn policies which are superior to those found by iLQG, even when learning directly from pixels.

It can be challenging to learn accurate value estimates. Q-learning, for example, is prone to over-estimating values (Hasselt, 2010). We examined DDPG's estimates empirically by comparing the values estimated by Q after training with the true returns seen on test episodes. Figure 3 shows that in simple tasks DDPG estimates returns accurately without systematic biases. For harder tasks the Q estimates are worse, but DDPG is still able to learn good policies.

To demonstrate the generality of our approach we also include Torcs, a racing game where the actions are acceleration, braking and steering. Torcs has previously been used as a testbed in other policy learning approaches (Koutník et al., 2014b). We used an identical network architecture and learning algorithm hyper-parameters to the physics tasks but altered the noise process for exploration because of the very different time scales involved. On both low-dimensional and high-dimensional inputs, some replicas were able to learn reasonable policies that are able to complete a circuit around the track though other replicas failed to learn a sensible policy.

Figure 1: Example screenshots of a sample of environments we attempt to solve with DDPG. In order from the left: the cartpole swing-up task, a reaching task, a gasp and move task, a puck-hitting task, a monoped balancing task, two locomotion tasks and Torcs (driving simulator). We tackle all tasks using both low-dimensional feature vector and high-dimensional pixel inputs. Detailed descriptions of the environments are provided in the supplementary. Movies of some of the learned policies are available at https://goo.gl/J4PIAz.

Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.
Over estimation of Q-values

Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.
Related Work

• TRPO
• PILCO
• iLQG
Batch RL / Offline RL

Instead of actively interacting with the environment

Online Reinforcement Learning

Agent

Environment

Offline Reinforcement Learning

Agent

Logged data
Batch RL / Offline RL

Why batch RL?

• Re-use experience: gathering experience is the most expensive part of RL
• Gathering experience may be unsafe
• Learn from other’s experience
Problems with Off-line Learning

\[ Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') \quad \forall (s, a, s', r) \in \mathcal{D} \]

- Extrapolation error
  - we do not know where our estimate of \( Q(s', a') \) is good
  - even if we assume \( Q(s', a') \) is an unbiased estimate, the \textbf{max} will cause it to become biased

Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.
Experiment 1

(a) Final buffer performance
(b) Concurrent performance
(c) Imitation performance
(d) Final buffer value estimate
(e) Concurrent value estimate
(f) Imitation value estimate
But, existing methods work, don’t they?

- DQN, DDPG aren’t really off-policy, use $\epsilon$-greedy policies
- $\textbf{max}$ introduces a bias, but unsubstantiated optimism can be tested in subsequent iterations.
Batch-constrained Q-learning

- policy should induce a similar state-action distribution as dataset
  - minimize distance of selected action to data in batch
  - lead to states where familiar data is observed
  - maximize the value function
- train a pair of networks (use minimum of Q-value)
Batch-constrained Q-learning

Algorithm 1 BCQ

Input: Batch $B$, horizon $T$, target network update rate $\tau$, mini-batch size $N$, max perturbation $\Phi$, number of sampled actions $n$, minimum weighting $\lambda$.

Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network $\xi_{\phi}$, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters $\theta_1$, $\theta_2$, $\phi$, $\omega$, and target networks $Q_{\theta'_1}, Q_{\theta'_2}, \xi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$.

for $t = 1$ to $T$ do

Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $B$

$\mu, \sigma = E_{\omega_1}(s, a)$, $\tilde{a} = D_{\omega_2}(s, z)$, $z \sim \mathcal{N}(\mu, \sigma)$

$\omega \leftarrow \text{argmin}_{\omega} \sum (a - \tilde{a})^2 + D_{KL}(\mathcal{N}(\mu, \sigma)\|\mathcal{N}(0, 1))$

Sample $n$ actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$

Perturb each action: $\{a_i = a_i + \xi_{\phi}(s', a_i, \Phi)\}_{i=1}^n$

Set value target $y$ (Eqn. 13)

$\theta \leftarrow \text{argmin}_{\theta} \sum (y - Q_\theta(s, a))^2$

$\phi \leftarrow \text{argmax}_{\phi} \sum Q_{\theta_1}(s, a + \xi_{\phi}(s, a, \Phi)), a \sim G_\omega(s)$

Update target networks: $\theta'_i \leftarrow \tau \theta + (1 - \tau) \theta'_i$

$\phi' \leftarrow \tau \phi + (1 - \tau) \phi' + r + \gamma \max_{a_i} \left[ \lambda \min_{j=1,2} Q_{\theta'_j}(s', a_i) + (1 - \lambda) \max_{j=1,2} Q_{\theta_j}(s', a_i) \right]$

end for
Variational Auto Encoders
We evaluate each method following the three experiments: to evaluate the effectiveness of Batch-Constrained deep reinforcement learning algorithms, we consider the off-policy agents, as well as the imperfect demonstrations task. For reproducibility, we make the off-policy agents learn from a dataset collected by an expert policy, with two sources of expert transitions, even in the presence of noise. The behavioral policy selects actions randomly with a single set of fixed hyper-parameters. Furthermore, compared to current state-of-the-art baselines, BCQ exhibit a highly stable value function in millions of time steps, unlike the deep reinforcement learning algorithms, DDPG and DQN, which perform poorly. BCQ, however, is able to strongly outperform the noisy demonstrator, disentangling error has been successfully mitigated through the batch. This suggests our approach effectively leverages the presence of off-policy samples, suggesting extrapolation algorithms perform poorly. BCQ, however, is able to outperform all other agents, besides the off-policy agents, even in the presence of off-policy samples. Furthermore, compared to current state-of-the-art baselines, BCQ exhibit a highly stable value function, with a single set of fixed hyper-parameters. The experimental results are reported in Figure 2, which shows the performance of BCQ and several baselines on the experiments from Section 2017, with a single set of fixed hyper-parameters. Additionally, to study the robustness of BCQ to noisy and imperfect demonstrations task can be found in the Supplementary Material.
We evaluate each method following the three experiments defined in Section 3.1.

To evaluate the effectiveness of Batch-Constrained deep reinforcement learning (BC), and a variant with a VAE (VAE-BC), we use cloning method (BC), and a variant with a VAE (VAE-BC), independently discretized action space, a feed-forward behavioral policy, and Q-learning (BCQ) in a high-dimensional setting, we focus on MuJoCo environments in OpenAI gym (Brockman et al. 2015).

We compare our method with DDPG (Lillicrap et al. 2016) and DQN (Mnih et al. 2016), as well as the imperfect demonstrations from Section 3.1.

In the imitation learning task where behavioral cloning unceremoniously fails, disentangling poor and expert actions. Furthermore, compared to current state-of-the-art baselines, BCQ exhibits a highly stable value function in each task, in which the agents are trained with a batch of 100k transitions collected by an expert policy, with two sources of noise. The behavioral policy selects actions randomly with probability $\frac{1}{3}$ and with high exploratory noise. The experimental results for these tasks are reported in Figure 2.

Additionally, to study the robustness of BCQ to noisy and imperfect demonstrations, even in the presence of noise. The behavioral policy selects actions randomly with high exploratory noise. The experimental results for these tasks are reported in Figure 2.

In the imperfect demonstrations task, we find that both deep reinforcement learning and imitation learning and off-policy reinforcement learning, that our algorithm can be used as a single approach for both imitation learning and off-policy reinforcement learning, and DQN, BCQ exhibits a highly stable value function in each instance, and outperforming all other agents, besides DDPG and DQN on the experiments from Section 3.1.

We evaluate BCQ and several baselines on the experiments from Section 3.1. In Figure 3, we examine the value estimates of BCQ, along with the estimated values of BCQ, DDPG and DQN, and the true value of BCQ, evaluated for these tasks are reported in Figure 3.

The shaded area represents half a standard deviation. The bold black line measures the average return of episodes contained in the final buffer.

The shaded area represents half a standard deviation. The bold black line measures the average return of episodes contained in the final buffer.

In the imperfect demonstrations task, we find that our algorithm can be used as a single approach for both imitation learning and off-policy reinforcement learning, and DQN, BCQ exhibits a highly stable value function in each instance, and outperforming all other agents, besides DDPG and DQN on the experiments from Section 3.1.

We examine the value estimates of BCQ, along with the estimated values of BCQ, DDPG and DQN, and the true value of BCQ, evaluated for these tasks are reported in Figure 3.

The shaded area represents half a standard deviation. The bold black line measures the average return of episodes contained in the final buffer.
We evaluate each method following the three experiments defined in Section 3.1. We examine the value estimates of BCQ, along with DDPG and DQN, in the Hopper-v1 environment. Each individual trial is plotted, with the estimated values of BCQ, DDPG, and DQN, and the true estimated values of BCQ, evaluated for these tasks.}

Q-value Estimates

(a) Final Buffer  (b) Concurrent  (c) Imitation

We evaluate each method following the three experiments:

- (a) Final Buffer: We observe the performance of BCQ, DDPG, DQN, and the true value in the final buffer task, where agents are trained with a batch of 100k expert demonstrations.
- (b) Concurrent: In the concurrent task, agents learn concurrently with the same replay buffer, as the behavioral DDPG policy.
- (c) Imitation: The imitation task involves learning from the final replay buffer gathered by training a DDPG agent over a million time steps.

Graphs and charts are used to compare the performance of BCQ, DDPG, and DQN, with a focus on estimating the true value of BCQ in the context of off-policy learning. The graphs represent the estimated values over time steps, with different environments and tasks.
Related Work

- Modeling uncertainty in neural networks

### (a) Imitation performance

![Graph showing Imitation performance](image1)

### (b) Imitation value estimates

![Graph showing Imitation value estimates](image2)
We observe a clear drop in performance with the increase of but large enough such that learning can be performed when exploratory actions are included in the dataset. Given that the data is based on expert performance, this is consistent with our understanding of extrapolation error.

We perform an ablation study on the perturbation model of BCQ, on the imitation task from Section 3.1. Performance is graphed on the left, and value estimates are graphed on the right. The shaded area represents half a standard deviation. The bold black line should be small enough to stay close to the generated actions, along with an increase in instability in the value function.

The performance and value estimates of BCQ when varying the hyper-parameter learning algorithms, such as DQN and DDPG. The model in the range BCQ includes a perturbation model which outputs a small residual update to the actions sampled by the generative model. If the agent learns to take actions that are further away from the data in the batch after erroneously overestimating the value of suboptimal actions. This suggests the ideal value of

\[ \Phi = a_{max} - a_{min}, \]

The average return of episodes contained in the batch measures the average return of episodes contained in the batch. For the value estimates, each individual trial is plotted, with the mean in bold. The policy to select actions which may not have been sampled by the generative model in the range Lillicrap et al. 2015

Mnih et al. 2015

\[ \Phi; \hspace{1em} ✓ \]

We examine the perturbation model in Figure 7. in Figure 7, we examine the perturbation model which outputs a small residual update to the actions sampled by the generative model. If

\[ \Phi; \hspace{1em} ✓ \]

\( s, a, \) along with an increase in instability in the value function.