NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

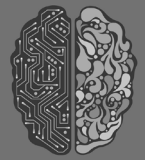
AI6101 Introduction to AI and AI Ethics

Deep Reinforcement Learning

Assoc Prof Bo AN

www.ntu.edu.sg/home/boan
Email: boan@ntu.edu.sg
Office: N4-02b-55

Lesson Outline



- DRL
- Deep Q-Network Models
- Advanced materials on DRL

Reinforcement Learning in a nutshell

- RL is a general-purpose framework for decision-making
  - RL is for an agent with the capacity to act
  - Each action influences the agent's future state
  - Success is measured by a scalar reward signal
  - Goal: select actions to maximize future reward

Deep Learning in a nutshell

- DL is a general-purpose framework for representation learning
  - Given an objective
  - Learn representation that is required to achieve objective
  - Directly from raw inputs
  - Using minimal domain knowledge

Deep RL: AI = RL + DL

- We seek a single agent which can solve any human-level task
  - RL defines the objective
  - DL gives the mechanism
  - RL + DL = general intelligence

Major Components of a Deep RL Agent

- A Deep RL agent may include one or more of these components:
  - Policy: agent's behavior function (**stochastic** or **deterministic**)
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

Deep Q-Network

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

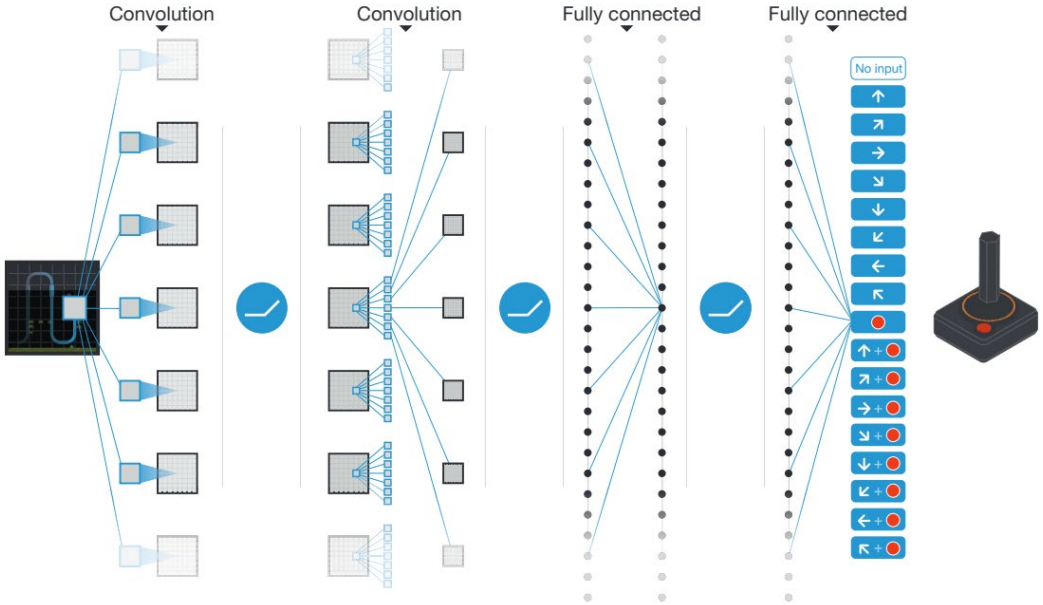
Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted

agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

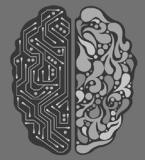
$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\, a_t = a,\, \pi\right],$$

DQN Architecture



The input to the neural network consists of an 84 x 84 x 4 image produced by the preprocessing map w, followed by three convolutional layers and two fully connected layers with a single output for each valid action

Main Component of DQN (1)



- Deep network with Q-learning (1)
  - Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

  - The Q network works like the Q table in Q-learning when selecting actions
    - States in Q-learning are countable and finite
    - States in DQN can be either finite or infinite/ continuous or discrete
  - The Q network updates via updating the weights in the Q networks

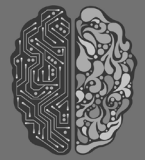
Main Component of DQN (2)

- Deep network with Q-learning (2)
  - Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

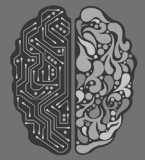
- This is a loss function for minimizing the TD error to update the weights in Q network

Main Component of DQN (3)



- Deep network with Q-learning (3)
  - Optimize objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$
    - There are many optimization algorithms available to use, for example *Adam* in TensorFlow or PyTorch

Main Component of DQN (4)



- Experience replay
  - Naive Q-learning oscillates or diverges with neural nets
    - Data is sequential
      - Successive samples are correlated, non-iid
    - Policy changes rapidly with slight changes to Q-values
      - Policy may oscillate
      - Distribution of data can swing from one extreme to another
    - Scale of rewards and Q-values is unknown
      - Naive Q-learning gradients can be large unstable when backpropagated
  - Store transition in a replay buffer and sample

DQN Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

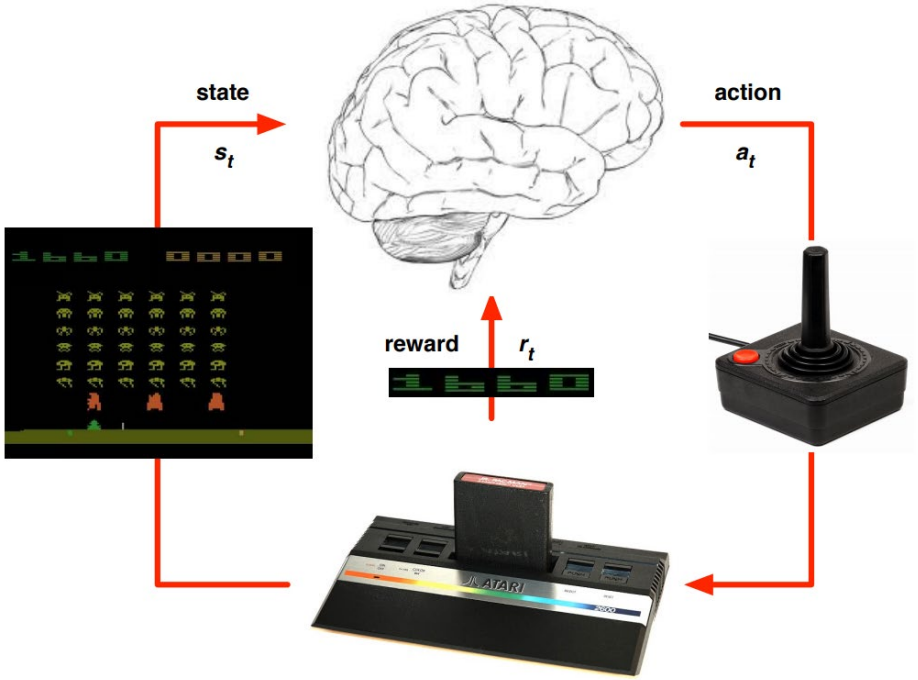
    Every C steps reset $\hat{Q} = Q$

  End For

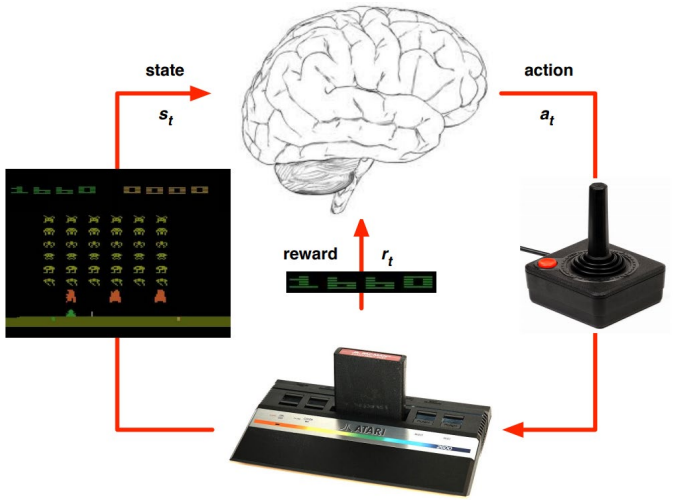
End For

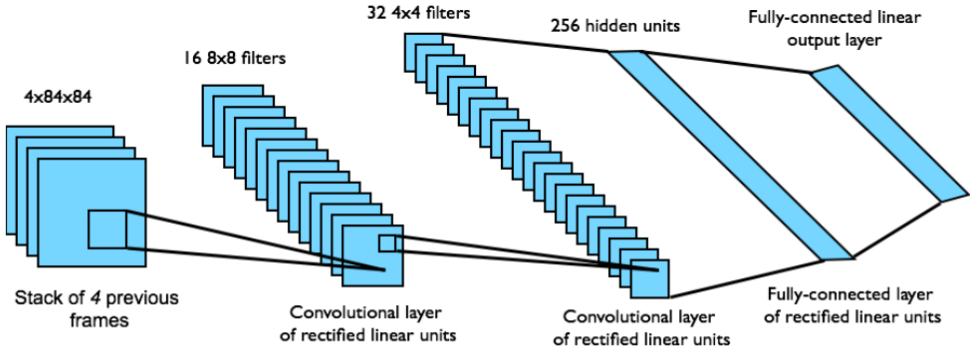
From DQN nature paper 2015

DQN in Atari



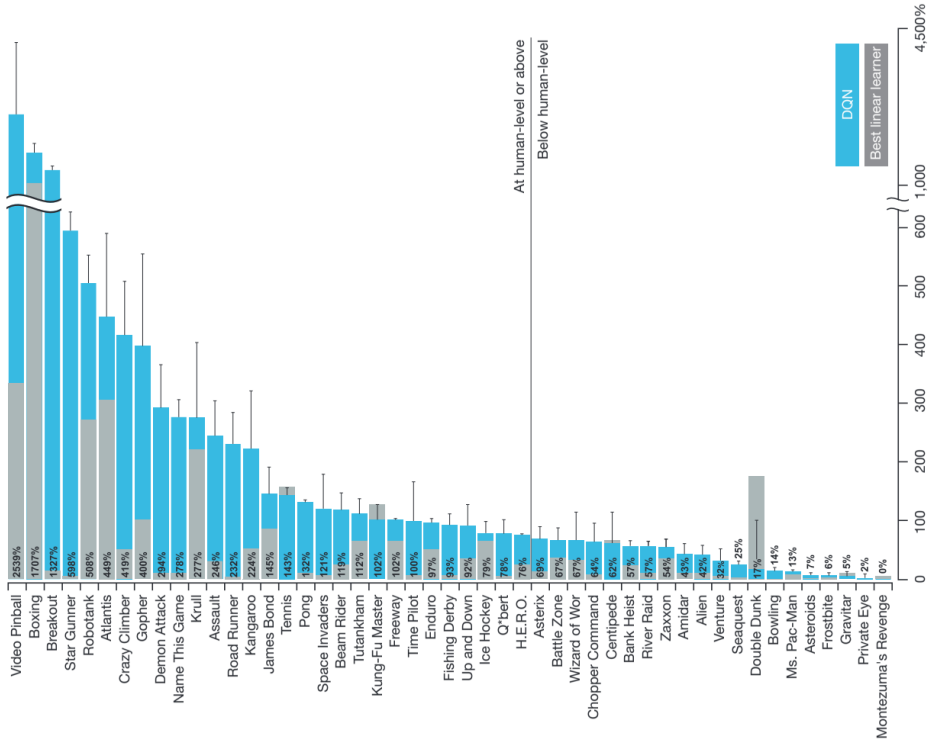
DQN in Atari



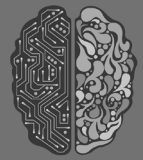


Network architecture and hyperparameters fixed across all games *[Mnih et al.]*

DQN Results in Atari



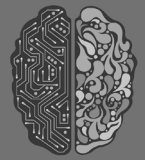
DQN in Atari



```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
    select an action a
        with probability ε select a random action
        otherwise select a = argmaxa'Q(s,a')
    carry out action a
    observe reward r and new state s'
    store experience <s, a, r, s'> in replay memory D

    sample random transitions <ss, aa, rr, ss'> from replay memory D
    calculate target for each minibatch transition
        if ss' is terminal state then tt = rr
        otherwise tt = rr + γmaxa'Q(ss', aa')
    train the Q network using (tt - Q(ss, aa))^2 as loss

    s = s'
until terminated
```

Refer to https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/

DQN in Atari with Code

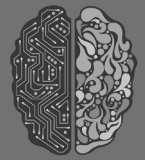


```
def nature_dqn_network(num_actions, network_type, state):
  """The convolutional network used to compute the agent's Q-values.

  Args:
    num_actions: int, number of actions.
    network_type: namedtuple, collection of expected values to return.
    state: `tf.Tensor`, contains the agent's current state.

  Returns:
    net: _network_type object containing the tensors output by the network.
  """
  net = tf.cast(state, tf.float32)
  net = tf.div(net, 255.)
  net = contrib_slim.conv2d(net, 32, [8, 8], stride=4)
  net = contrib_slim.conv2d(net, 64, [4, 4], stride=2)
  net = contrib_slim.conv2d(net, 64, [3, 3], stride=1)
  net = contrib_slim.flatten(net)
  net = contrib_slim.fully_connected(net, 512)
  q_values = contrib_slim.fully_connected(net, num_actions, activation_fn=None)
  return network_type(q_values)
```

Build Q networks

Refer to https://github.com/google/dopamine

DQN in Atari with Code



```
target = tf.stop_gradient(self._build_target_q_op())
loss = tf.losses.huber_loss(
    target, replay_chosen_q, reduction=tf.losses.Reduction.NONE)
if self.summary_writer is not None:
  with tf.variable_scope('Losses'):
    tf.summary.scalar('HuberLoss', tf.reduce_mean(loss))
return self.optimizer.minimize(tf.reduce_mean(loss))
```

The loss

For more code, please refer to Google's Dopamine

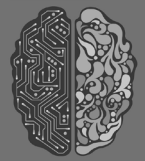
```
if random.random() <= epsilon:
  # Choose a random action with probability epsilon.
  return random.randint(0, self.num_actions - 1)
else:
  # Choose the action with highest Q-value at the current state.
  return self._sess.run(self._q_argmax, {self.state_ph: self.state})
```

Select actions

Refer to https://github.com/google/dopamine

Advanced Materials

DQN's Variants

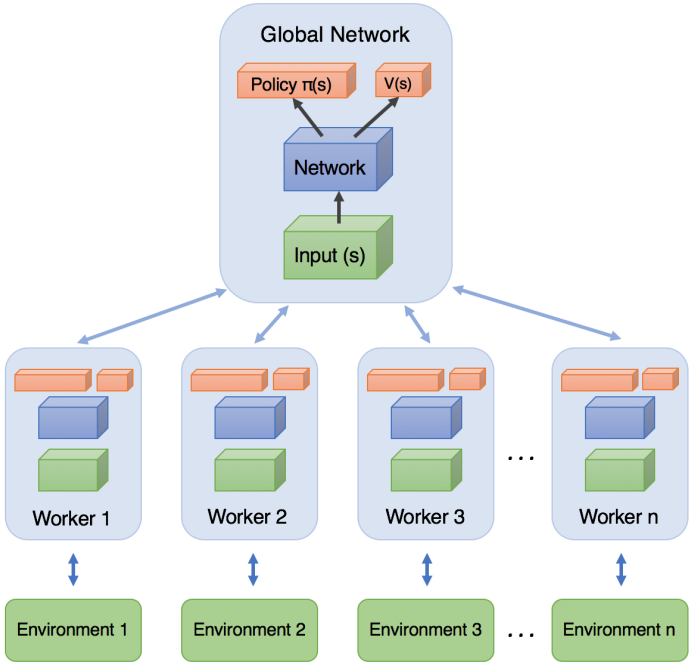


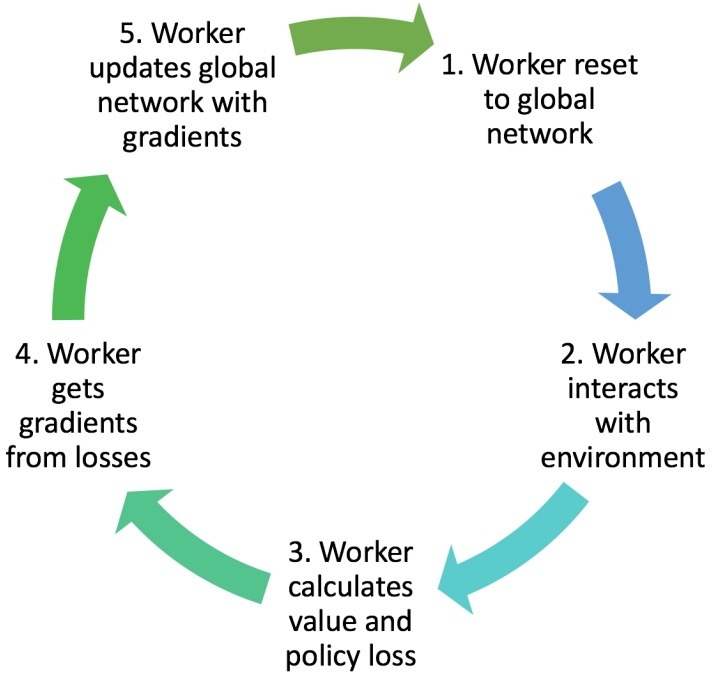
- Double DQN
- C51: distributional DQN with Categorical distribution
- Quantile Q Network
- Implicit Quantile Network
- Rainbow

Deep Policy Gradient Methods

- Deep Stochastic Policy Gradient
  - The action is subject to a distribution and chosen with certain probability
  - Algorithms:
    - A3C
    - TRPO
    - PPO
    - SAC
- Deep Deterministic Policy Gradient
  - The probability of selecting a specific action given a state is 1
  - Algorithms:
    - Deterministic Policy Gradient (DPG)
    - Deep Deterministic Policy Gradient (DDPG)

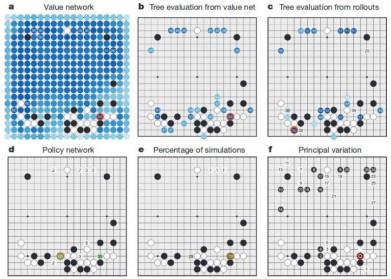
Asynchronous Advantage Actor-Critic (A3C)

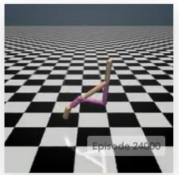




Trust Region Policy Optimization

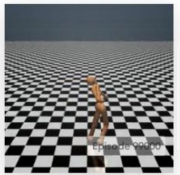
- Problem Domain: Locomotion
  - The two action domains in reinforcement learning
    - Discrete action space (Atari, Game of Go)
    - Continuous action space (Mujuco, Robotic control)





Walker2d-v1
Make a 2D robot walk.

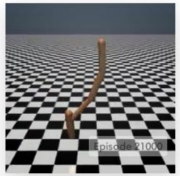
Ant-v1
Make a 3D four-legged robot walk.



Humanoid-v1
Make a 3D two-legged robot walk.

HalfCheetah-v1
Make a 2D cheetah robot run.

Swimmer-v1
Make a 2D robot swim.



Hopper-v1
Make a 2D robot hop.

Trust Region Policy Optimization

- To improve training stability, avoid parameter updates that change the policy too much at one step
- Use a KL divergence constraint on the size of policy update at each iteration

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right]$$

$$D_{\mathrm{KL}}(P \parallel Q) = -\sum_{x\in\mathcal{X}} P(x)\log\left(\frac{Q(x)}{P(x)}\right)$$

Trust Region Policy Optimization

- TRPO aims to maximize the objective function J(θ) subject to, trust region constraint which enforces the distance between old and new policies measured by KL-divergence to be small enough, within a parameter δ

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(.\,|s)\|\pi_{\theta}(.\,|s)\right]\leq\delta\right.$$

Trust Region Policy Optimization

Line search
(like gradient ascent)

Trust region

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(.|s)\|\pi_{\theta}(.|s)\right]\leq\delta\right.$$

Proximity Policy Optimization

- TRPO is complicated.
- Proximal Policy Optimization (PPO) simplifies it by using a clipped surrogate objective while retaining similar performance

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

$$J^{\text{TRPO}}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a)]$$

instability with extremely large parameter updates and big policy ratios

Proximity Policy Optimization

- PPO imposes the constraint by forcing r(θ) to stay within a small interval around 1, precisely [1-ε, 1+ε]

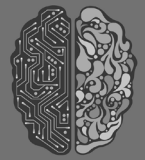
$$J^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a))]$$

Adding More constrains on shared policy and value networks

$$J^{\text{CLIP'}}(\theta) = \mathbb{E}[J^{\text{CLIP}}(\theta) - c_1(V_\theta(s) - V_{\text{target}})^2 + c_2 H(s, \pi_\theta(.))]$$

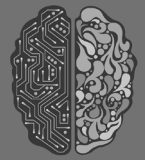
value error entropy

Soft Actor-Critic



- Uses the entropy of the policy into the reward to encourage exploration
  - learn a policy that acts as randomly as possible
  - while it is still able to succeed at the task
  - follow maximum entropy RL framework

Soft Actor-Critic



- Three key components in SAC (continuous control)
  - Actor-Critic architecture
  - Off-policy formulation
  - Entropy maximization to enable stability and exploration

Entropy

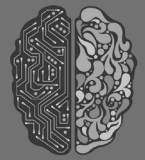
$$H(P) = \mathop{\mathrm{E}}_{x\sim P}[-\log P(x)]$$

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(.|s_t))]$$

Entropy

Entropy-Regularized Reinforcement Learning

Soft Actor-Critic



- Three key components in SAC (continuous control)
  - Actor-Critic architecture
  - Off-policy formulation
  - Entropy maximization to enable stability and exploration

Q

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho_\pi(s)}[V(s_{t+1})]$$

V

$$\text{where } V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \alpha \log \pi(a_t | s_t)]$$

Bellman Equation

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}, a_{t+1}) \sim \rho_\pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} | s_{t+1})]$$

J

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}[Q_w(s_t, a_t) - \log \pi_\theta(a_t | s_t)])^2]$$

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_w(s_t, a_t) + \log \pi_\theta(a_t | s_t))$$

Entropy

$$H(P) = \mathop{\mathrm{E}}_{x \sim P}[-\log P(x)]$$

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi_\theta}}[r(s_t, a_t) + \alpha \mathcal{H}(\pi_\theta(. | s_t))]$$

Entropy

Entropy-Regularized Reinforcement Learning

Soft Actor-Critic

Soft Q function

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho_\pi(s)}[V(s_{t+1})]$$
$$\text{where } V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \alpha \log \pi(a_t | s_t)]$$

Soft Value function

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}[\frac{1}{2}\big(V_\psi(s_t) - \mathbb{E}[Q_w(s_t, a_t) - \log \pi_\theta(a_t | s_t)]\big)^2]$$
$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)\big(V_\psi(s_t) - Q_w(s_t, a_t) + \log \pi_\theta(a_t | s_t)\big)$$

SAC updates the policy to minimize the KL-divergence

$$\begin{aligned}
\pi_{\text{new}} &= \arg\min_{\pi' \in \Pi} D_{\text{KL}}\left(\pi'(.|s_t) \| \frac{\exp(Q^{\pi_{\text{old}}}(s_t, .))}{Z^{\pi_{\text{old}}}(s_t)}\right) \\
&= \arg\min_{\pi' \in \Pi} D_{\text{KL}}\left(\pi'(.|s_t) \| \exp(Q^{\pi_{\text{old}}}(s_t, .) - \log Z^{\pi_{\text{old}}}(s_t))\right) \\
J_\pi(\theta) &= \nabla_\theta D_{\text{KL}}\left(\pi_\theta(.|s_t) \| \exp(Q_w(s_t, .) - \log Z_w(s_t))\right) \\
&= \mathbb{E}_{a_t \sim \pi}\Big[-\log\big(\frac{\exp(Q_w(s_t, a_t) - \log Z_w(s_t))}{\pi_\theta(a_t | s_t)}\big)\Big] \\
&= \mathbb{E}_{a_t \sim \pi}[\log \pi_\theta(a_t | s_t) - Q_w(s_t, a_t) + \log Z_w(s_t)]
\end{aligned}$$

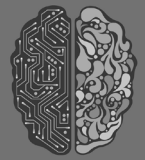
Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t | \mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ Soft state value function
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ Soft Q function
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ Updates the policy
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Deterministic Policy Gradient (DPG)

- Why deterministic
  - Can we make the stochastic policies deterministic and follow the updating approach as stochastic PG methods do: adjusting the policy parameters in the direction of the policy gradient?
- Main property
  - Action space: **continuous**
  - Policy (action function): **deterministic**

DPG Overview



- Represent deterministic policy by deep network **a = π(s, u)** with weights **u**
- Define objective function as total discounted reward

$$J(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + ...\right]$$

policy weights

- Optimize objective end-to-end by SGD

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s\left[\frac{\partial Q^\pi(s,a)}{\partial a}\frac{\partial \pi(s,u)}{\partial u}\right]$$

  - Update policy in the direction that most improves Q
  - i.e. Backpropagate critic through actor
- Not a DRL algorithm

Main Components of DPG

- Use two networks: an actor and a critic
  - Critic estimates value of current policy by Q-learning
  - Actor updates policy in direction that improves Q

Objective

the initial distribution over states

$$J(\mu_\theta) = \int_{\mathcal{S}} \rho^{\mu}(s) r(s, \mu_\theta(s)) \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^{\mu}} \left[r(s, \mu_\theta(s)) \right]$$

Gradient

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_\theta \mu_\theta(s) \left. \nabla_a Q^{\mu}(s,a) \right|_{a=\mu_\theta(s)} \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^{\mu}} \left[\nabla_\theta \mu_\theta(s) \left. \nabla_a Q^{\mu}(s,a) \right|_{a=\mu_\theta(s)} \right] \quad ($$

Parameter update rules

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t) \quad \text{; TD error in SARSA}$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t)$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_a Q_w(s_t, a_t) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)} \quad \text{; Deterministic policy gradient theorem}$$

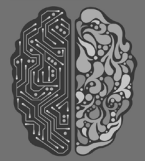
Refer to http://proceedings.mlr.press/v32/silver14.pdf

Deep DPG for Continuous Control

- Motivation
  - DQN cannot handle continuous action spaces, can we derive a continuous-action space DQN with deterministic policy gradients?
- **Deep DPG = DPG + DQN**
  - Target network
  - Experience replay
  - Deterministic policy
  - Soft updates on the parameters of both actor and critic

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

DDPG: Algorithm



Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' 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

Deterministic policy

Exploration

Replay Buffer

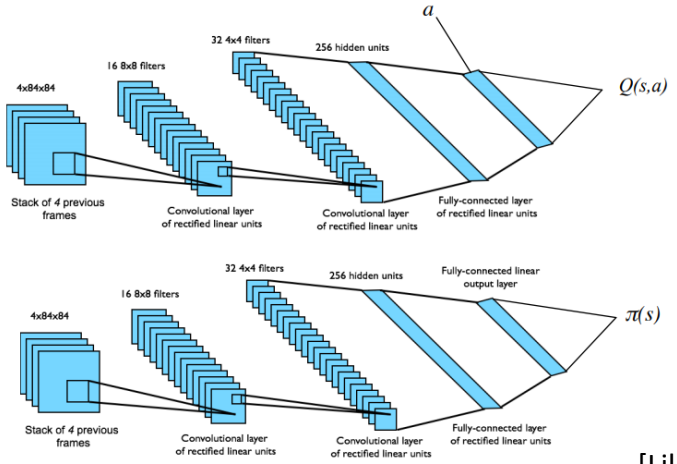
Critic loss

Update gradient of the actor

Soft update

DDPG: Network Structure

- Network Structure
  - End-to-end learning of control policy from raw pixels s
  - Input state s is stack of raw pixels from last 4 frames
  - Two separate convnets are used for Q and π



[Lillicrap et al.]