TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Deep Reinforcement Learning

Review of Attention

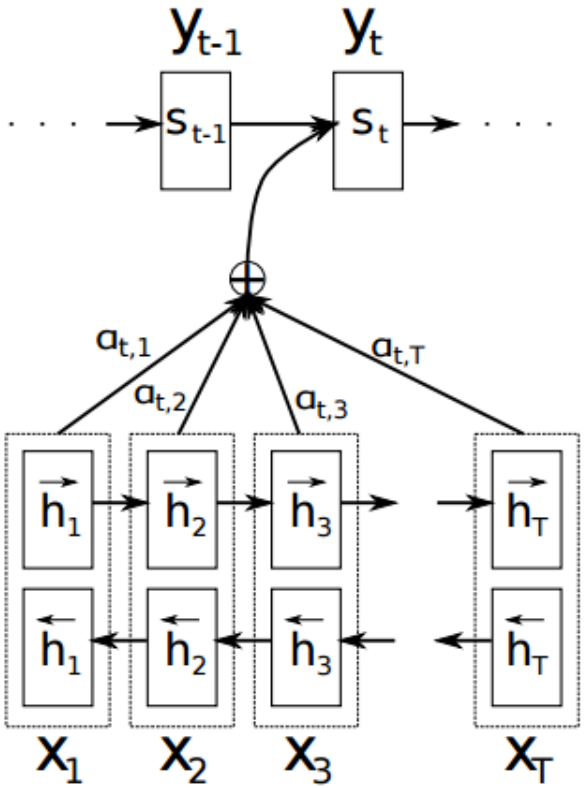
$$P(\cdot \,|\mathbf{x},\ y_1, \ldots, y_i) = \text{softmax}\ W_y\ [s_i,\ e(y_i),\ c_i]$$

$$s_{i+1} = \text{GRU}(s_i,\ [e(y_i),\ c_i],\ \Theta_s)$$

$$c_{i+1} = \sum_t \alpha_{i,t}\ \overleftrightarrow{h}_t$$

$$\alpha_{i+1,t} = \underset{t}{\text{softmax}}\ s_i \cdot \overleftrightarrow{h}_t$$

Attention



[Bahdanau, Cho, Bengio (2014)]

Definition of Reinforcement Learning

RL is defined by the following properties:

- An environment with **state**.
- State changes are influenced by **sequential decisions**.
- Reward (or loss) depends on **making decisions** that lead to **desirable states**.

Reinforcement Learning Examples

- Board games (chess or go)
- Atari Games (pong)
- Robot control (driving)
- Dialog

Immitation Learning

Immitation Learning

Construct a dataset of state-action pairs (s, a) from experts.

Define stochastic policy $\pi_\Theta(s)$.

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \, \mathrm{E}_{(s,a)} \left[- \ln \pi_\Theta(a \mid s) \right]$$

This is just log loss for labeled data.

Dangers of Imperfect Immitation Learning

Perfect imitation learning would reproduce expert behavior. Imitation learning is **off-policy** — the state distribution in the training data is different from that defined by the policy being learned.

Imperfect imitation learning can generate state distributions very different from that in the training data.

Also, imitating experts can never exceed expert performance (consider go).

Markov Decision Processes

Markov Decision Processes (MDPs)

For an RL problem we work with an action policy π

s_t is the state at time t

r_t is the reward at time t

a_t is the action taken at time t.

$r_t = R(s_t, a_t)$ reward at time t

$\pi(a_t|s_t)$ probability of action a_t in state s_t

$T(s_{t+1}|s_t, a_t)$ state transition probability

The state space, action space, R and T define a **Markov Decision Process** or **MDP**.

Optimizing Reward

In RL we maximize reward rather than minimize loss.

$$\pi^* = \underset{\pi}{\text{argmax}}\ R(\pi)$$

$$R(\pi) = \mathrm{E}\left[\sum_{t=0}^{T} r_t\right] \qquad \text{episodic reward (go)}$$

$$\text{or } \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \qquad \text{discounted reward (Atari games)}$$

$$\text{or } \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} r_t \qquad \text{asymptotic average reward (driving)}$$

The Value Function

For discounted reward:

$$V^{\pi}(s) = \mathrm{E}\left[\sum_{t} \gamma^{t} r_t \mid \pi,\ s_0 = s\right]$$

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

$$\pi^{*}(a|s) = \operatorname*{argmax}_{a} \mathrm{E}_{s' \sim T(s'|s,a)}\left[V^{*}(s')\right]$$

$$V^{*}(s) = \max_{a} R(s,a) + \gamma \mathrm{E}_{s' \sim T(\cdot|s,a)}\left[V^{*}(s')\right]$$

The Q Function

For discounted reward:

$$Q^{\pi}(s,a) = \mathrm{E}\left[\sum_t \gamma^t r_t \mid \pi,\ s_0 = s,\ a_0 = a\right]$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$$\pi^{*}(a|s) = \operatorname*{argmax}_{a} Q^{*}(s,a)$$

$$Q^{*}(s,a) = R(s,t) + \gamma \mathrm{E}_{s' \sim T(\cdot|s,a)}\left[\max_{a'} Q^{*}(s',a')\right]$$

Q-Learning

Q-Learning

We will assume a parameterized Q function $Q_\Theta(s, a)$.

Define the **Bellman error**

$$\left(Q_\Theta(s,a) - \left(R(s,a) + \gamma \, \mathrm{E}_{s' \sim S(\cdot|s,a)} \left[\max_{a'} \; Q_\Theta(s', a')\right]\right)\right)^2$$

Theorem: If this error is zero for all s, a then $Q_\Theta = Q^*$.

Algorithm: run the policy $\mathrm{argmax}_a \; Q_\Theta(s_{t+1}, a)$ and repeat

$$\Theta \; \texttt{-=} \; \eta \nabla_\Theta \; \left(Q_\Theta(s_t, a_t) - (r_t + \gamma \; \max_a \; Q_\Theta(s_{t+1}, a))\right)^2$$

Issues with Q-Learning

Problem 1: Nearby states in the same run are highly correlated.

Problem 2: SGD on Bellman error tends to be unstable.

To address these problems we can use a **replay buffer**.

Using a Replay Buffer

We use a replay buffer of tuples (s_t, a_t, r_t, s_{t+1}).

Repeat:

1. Run the policy $\text{argmax}_a\, Q_\Theta(s, a)$ to add tuples to the replay buffer. Remove oldest tuples to maintain a maximum buffer size.
2. $\Psi = \Theta$
3. for N times select a random element of the replay buffer and do

$$\Theta \mathrel{-}= \eta \nabla_\Theta \left(Q_\Theta(s_t, a_t) - (r_t + \gamma \operatorname*{argmax}_a Q_\Psi(s_{t+1}, a)\right)^2$$

Multi-Step Q-learning

$$\Theta \mathrel{-}= \sum_t \nabla_\Theta \left(Q_\Theta(s_t, a_t) - \sum_{\delta=0}^{K} \gamma^\delta r_{(t+\delta)} \right)^2$$

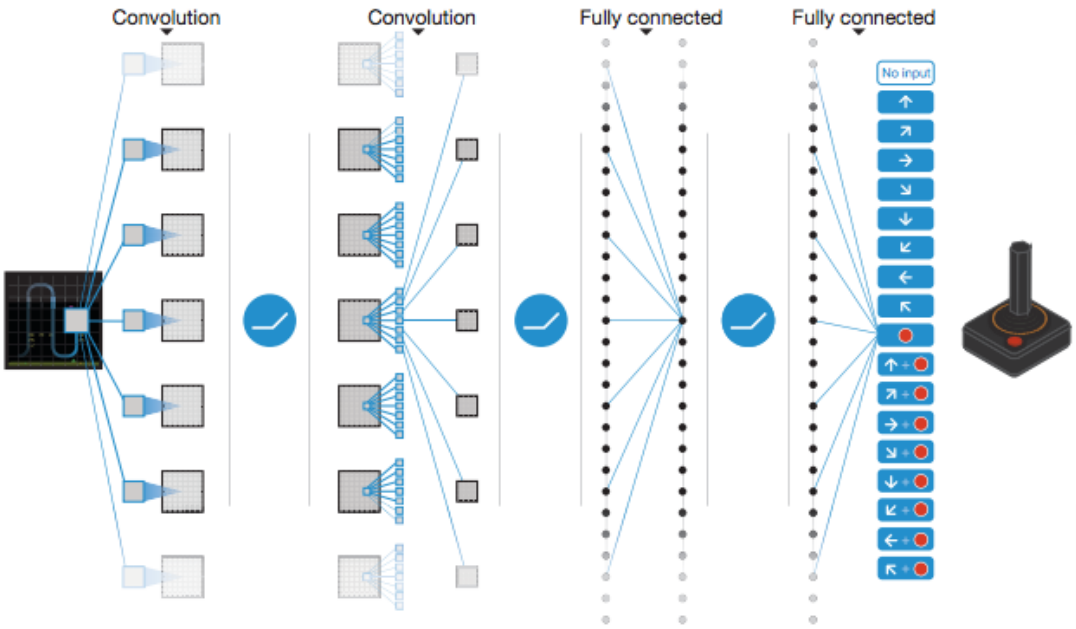
Deep Q-Learning (DQN)

Human-level control through deep reinforcement learning, Mnih et al., Nature, 2015. (Deep Mind)

Asynchronous Methods for Deep Reinforcement Learning, Mnih et al., Arxiv, 2016 (Deep Mind)

Deep Q-Networks

We consider a **Deep Q-network** — a deep network with parameters Θ and computing a Q-value $Q_\Theta(s, a)$.



Asynchronous Q-Learning (Simplified)

No replay buffer. Many aynchronous threads each repeating:

$\tilde{\Theta} = \Theta$ (retrieve global Θ)

using policy $\text{argmax}_a Q_{\tilde{\Theta}}(s, a)$ compute

$$s_t, a_t, r_t, \ldots, s_{t+K}, a_{t+K}, r_{t+K}$$

$$R_i = \sum_{\delta=0}^{t+K-i} \gamma^{i+\delta} r_{(i+\delta)}$$

Update global Θ:

$$\Theta \text{ -= } \eta \sum_{i=t}^{t+K} \nabla_{\tilde{\Theta}} \left(Q_{\tilde{\Theta}}(s_i, a_i) - R_i\right)^2$$

Policy Gradient

We assume a parameterized policy $\pi_\Phi(a|s)$.

$\pi_\Phi(a|s)$ is normalized while $Q_\Theta(s,a)$ is not.

$$\Phi \mathrel{+}= \eta \nabla_\Phi \, R(\Phi)$$

Policy Gradient Theorem (Episodic Case)

$$\mathrm{E}\left[R \mid \Phi\right] = \sum_{s_0,a_0,s_1,a_1,\ldots,s_T,a_T} P(s_0, a_0, s_1, a_1, \ldots, s_T, a_T)\ R$$

$$\begin{aligned}
\nabla_\Phi\ P(\ldots)R &= P(S_0)\nabla_\Phi\ \pi(a_0)P(s_1)\pi(a_1)\cdots P(s_T)\pi(a_T)\ R \\
&\quad + P(S_0)\pi(a_0)P(s_1)\nabla_\Phi\ \pi(a_1)\cdots P(s_T)\pi(a_T)\ R \\
&\quad \vdots \\
&\quad + P(S_0)\pi(a_0)P(s_1)\pi(a_1)\cdots P(s_T)\nabla_\Phi\ \pi(a_T)\ R \\
&= P(\ldots)\left(\sum_i \frac{\nabla_\Phi\ \pi_\Phi(a_i)}{\pi_\Phi(a_i)}\right) R \\
\nabla_\Phi\ \mathrm{E}\left[R \mid \Phi\right] &= \mathrm{E}\left[R \sum_t\ \nabla_\Phi\ \ln \pi_\Phi(a_t|s_t)\right]
\end{aligned}$$

Policy Gradient Theorem (Episodic Case)

$$\nabla_\Phi R(\Phi) = \sum_s \rho(s) \sum_a \left(\nabla_\Phi \, \pi_\Phi(a|s)\right) Q^{\pi_\Phi}(s,a)$$

$\rho(s)$ is the expected number of occurances of s

$Q^\pi(s,a) = \mathrm{E}_\pi \left[\sum_t \; r_t \mid s_0 = s, \; a_0 = a\right]$

This corresponds to an **Actor-Critic Algorithm**

Policy Gradient Theorem (Episodic Case)

$$\nabla_\Phi R(\Phi) = \sum_s \rho(s) \sum_a \left(\nabla_\Phi \, \pi_\Phi(a|s)\right) \left(Q^{\pi_\Phi}(s,a) - V^{\pi_\Phi}(s)\right)$$

$$V^\pi(s) = \mathrm{E}_{a \sim \pi(\cdot|s)} \left[Q(s,a)\right]$$

This corresponds to an **Advantage** Actor-Critic Algorithm.

Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Methods for Deep Reinforcement Learning, Mnih et al., Arxiv, 2016 (Deep Mind)

Asynchronous Advantage Actor-Critic (A3C)

$\tilde{\Phi} = \Phi; \tilde{\Psi} = \Psi$ (retrieve global Φ and Ψ)

using policy $\pi_{\tilde{\Phi}}$ compute $s_t, a_t, r_t, \ldots, s_{t+K}, a_{t+K}, r_{t+K}$

$$R_i = \sum_{\delta=0}^{t+K-i} \gamma^{i+\delta} r_{(i+\delta)}$$

Update global Φ and Ψ

$$\Phi \texttt{ += } \eta \sum_{i=t}^{t+K} \left(\nabla_{\tilde{\Phi}} \ln \pi_{\tilde{\Phi}}(a_i|s_i)\right) \left(R_i - V_{\tilde{\Psi}}(s_i)\right)$$

$$\Psi \texttt{ -= } \eta \sum_{i=t}^{t+K} \nabla_{\tilde{\Psi}} \left(V_{\tilde{\Psi}}(s_i) - R_i\right)^2$$

END