Deep Reinforcement Learning

BLAKE MASON & MOAYAD ALNAMMI

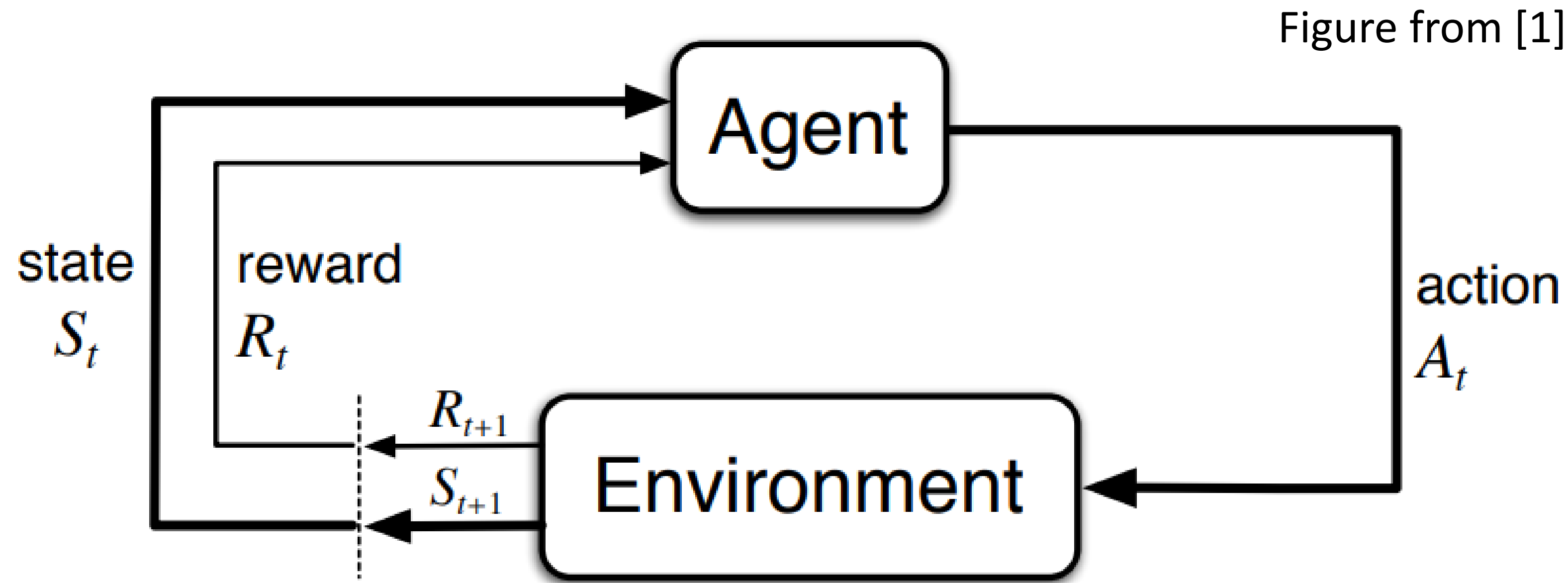
References

1. Reinforcement Learning An Introduction 2nd ed. 2018. Sutton and Barto.
2. An Introduction to Deep Reinforcement Learning. Lavet et. al. https://arxiv.org/pdf/1811.12560.pdf
3. Policy Gradient Methods Summaries: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#Proof-of-Policy-Gradient-Theorem
4. David Silver's slides on RL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Part 1: Problem Overview

Problem Overview (1)

Figure from [1]



- Assume the problem is a Markovian decision process (MDP):

$$P(S_{t+1}, R_{t+1} | S_t, A_t) = P(S_{t+1}, R_{t+1} | S_t, A_t, S_{t-1}, A_{t-1}, ..., S_0, A_0)$$

Problem Overview (2)

- The **dynamics of the system** fully specified by: p(s’, r | s, a).

1. State-transitions: $p(s'|s,a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s,a)$
2. Expected rewards:

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s,a)$$

$$r(s,a,s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s,a)}{p(s'|s,a)}$$

- **Goal** is to find a policy $\pi(a|s)$ that maximizes expected return (**state-value function**):

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Problem Overview (3)

- Action-value function. Action at current state already made:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

- Bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_\pi(s')\Big]$$

- Optimal policies:

$$v_*(s) \doteq \max_\pi v_\pi(s) \qquad q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

Reward of action made at current state; can be sub-optimal.

Reward of following optimal policy from next state onward.

Problem Overview (4)

- Bellman optimality equations:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$
$$= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\Big|\, S_t = s, A_t = a\Big]$$

- Optimal policy given optimal value functions:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} q_*(s, a)$$
$$= \operatorname*{argmax}_{a \in A} \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

Dynamic Programming Summary (1)

- The dynamics are **known:** p(s', r | s, a).
- Policy evaluation (Iterative): $$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$
- Policy Improvement: $$\begin{aligned} \pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned}$$
- Policy iteration: $\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$
- **Value iteration** combines policy iteration and improvement using Bellman optimality eq.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

Monte Carlo Methods Summary (1)

- No need for system dynamics. Applied to episodic tasks.
- Does not bootstrap: estimate of one state does not use estimates of other states.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Loop forever (for each episode): ← Ensures all (s, a) pairs are visited

  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ ← Generated from current policy

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$ ← Sum returns in reverse order

    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

      Append G to $Returns(S_t, A_t)$

      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$ ← Evaluate current policy

      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ ← Improve policy

Figure from [1]

Monte Carlo Methods Summary (2)

- Without exploring starts, need to use eps-soft policies to ensure all (s,a) are visited.
- Same idea of evaluation -> improvement.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$ ← Evaluate current policy
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily
      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

← Improve policy

Figure from [1]

Temporal-Difference (TD) Methods Summary (1)

- Update estimate towards a **target** at every time-step t: $V(S_t) \leftarrow V(S_t) + \alpha[Target - V(S_t)]$
- Recall: $$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned}$$
- Monte-Carlo update: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$ where G_t is estimate of $\mathbb{E}_\pi[G_t|S_t = s]$
- Dynamic-Prog. Update: $V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s]$ i.e. $\alpha = 1$
- MC target -> no bootstrapping. DP target -> bootstraps using $V(S_{t+1})$ estimate.
- TD(0) update combines these:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Bootstrap estimate

Sample

Temporal-Difference (TD) Methods Summary (2)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

On-policy because A' selected using Q and Q(S', A') used in update. i.e. following **current** policy.

Policy evaluation + improvement

Figure from [1]

Temporal-Difference (TD) Methods Summary (3)

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Figure from [1]

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q($

Loop for each episode:
- Initialize S
- Loop for each step of episode:
  - Choose A from S using policy derived from Q (e.g., ε-greedy)
  - Take action A, observe R, S'
  - $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
  - $S \leftarrow S'$
- until S is terminal

Approximates q_* directly to satisfy bellman opt. equation.

Off-policy because **next action** uses greedy policy $\max_a Q(S', a)$ in update. i.e. assuming **greedy** policy is followed.

Temporal-Difference (TD) Methods Summary (5)

- TD methods do not require system dynamics and can be run in the online setting.
- TD-target biased but lower variance than MC-target. In practice, TD methods converge faster. From Sutton & Barto pg. 124 no proof as of yet.
- Difference between SARSA and Q-Learning:

"As in all on-policy methods, we continually estimate q_π for the behavior policy π, and at the same time **change $\boldsymbol{\pi}$ toward greediness with respect to $\boldsymbol{q_\pi}$**." – SARSA Sutton & Barto pg.129

'' In this case, the learned action-value function, Q, **directly approximates q_*,** the optimal action-value function, independent of the policy being followed.'' – Q-Learning Sutton & Barto pg.131

- Other types of targets: n-step target and lambda-return targets/eligibility traces. We will show these later on. Requires more computation than TD(0), but converges faster.

Function Approximation of Value Functions (1)

- Previous methods use tables for storing value functions; one entry for each state or state-action pair.
- Some tasks have a large state-space and/or action-space. Memory issue, but also experience issue. Many states/actions may never be visited.
- Solution: approximate the value function using supervised learning.

$$\hat{v}(s; \boldsymbol{w}) \approx v_*(s) \qquad \hat{q}(s, a; \boldsymbol{w}) \approx q_*(s, a)$$

- With gradient descent based methods, we minimize the MSE:

$$MSE(\boldsymbol{w}) = \sum_{s \in S} (q_*(s, a) - \hat{q}(s, a; \boldsymbol{w}))^2$$

- In practice, we don't know q_* , so we approximate it via a target.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[U_t - \hat{q}(S_t, A_t; \boldsymbol{w}_t)]\nabla\hat{q}(S_t, A_t; \boldsymbol{w}_t)$$

Function Approximation of Value Functions (2)

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

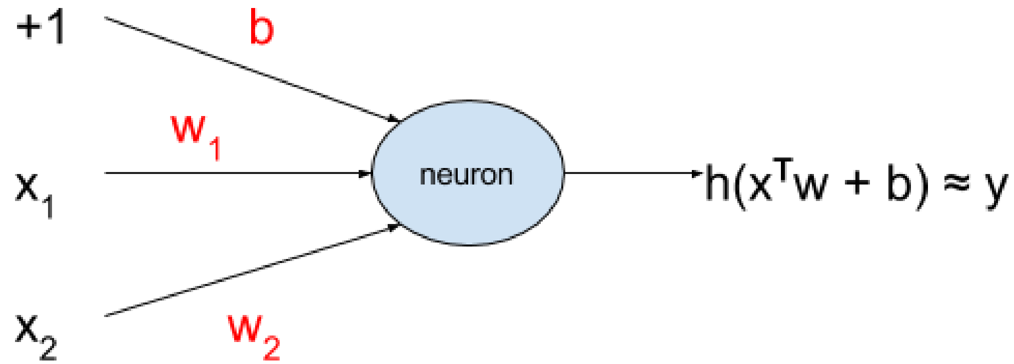
Figure from [1]

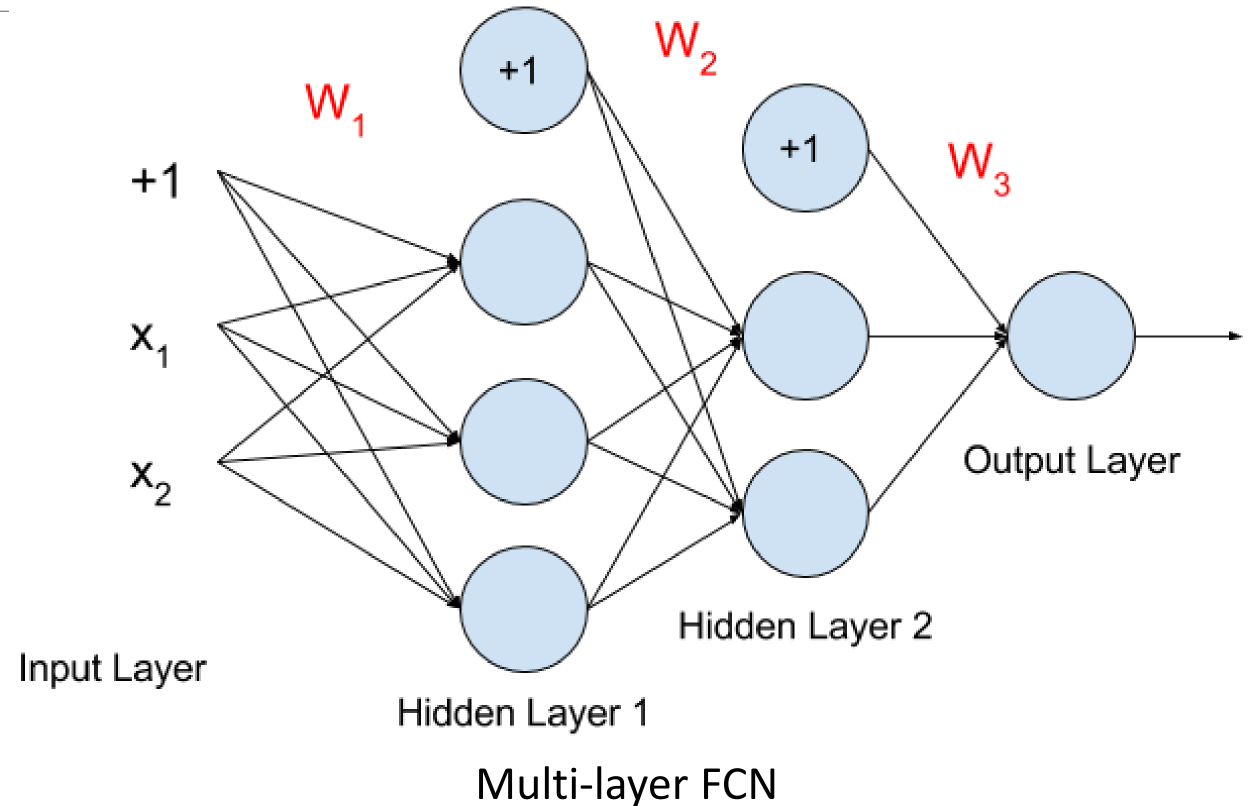
TD(0) target; **biased estimate of current policy: $q_{\pi,t}$*(S', A')***.
If MC target G_t is used, then it is unbiased; no dependence on w

Part 2:
Neural Networks
Quick Overview

Fully-Connected Network (FCN)

Single neuron unit:





Multi-layer FCN

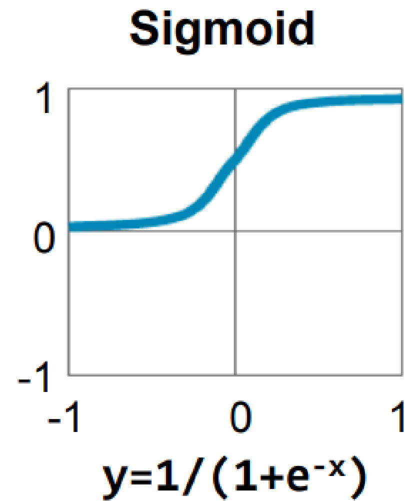
Introduce **Non-Linearity** via activations:

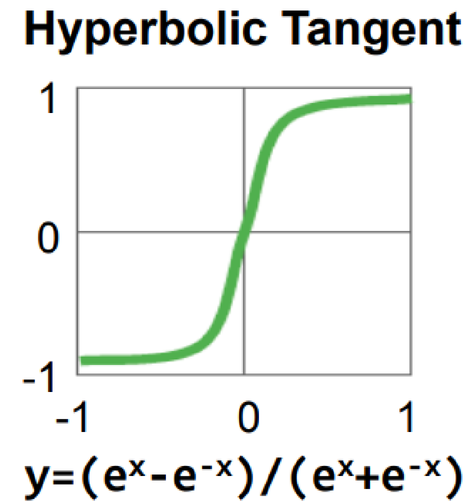
$$h(x^T w + b) = \frac{1}{1 + e^{-(x^T w + b)}}$$

$$h(x^T w + b) = \max(0, x^T w + b)$$

$$h(x^T w + b) = \max(0, \lambda(x^T w + b))$$

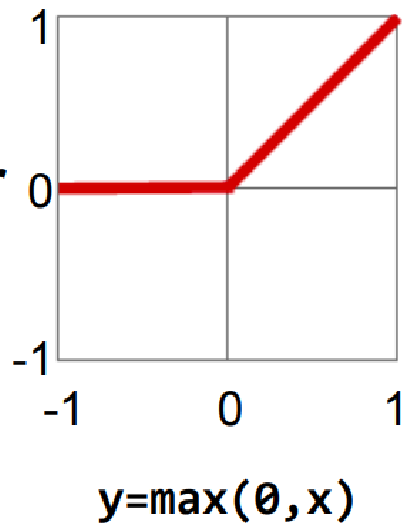
Traditional Non-Linear Activation Functions



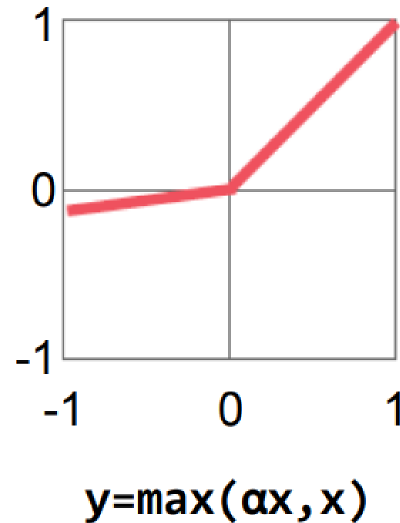


Modern Non-Linear Activation Functions

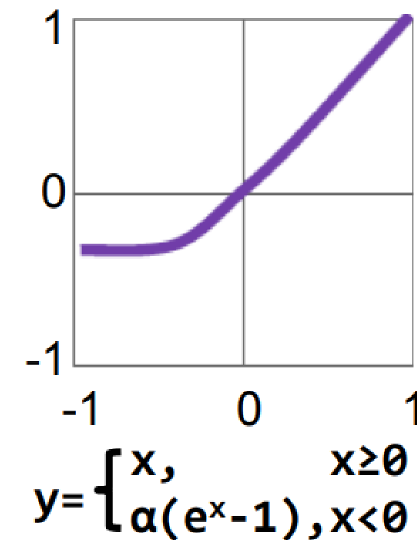
Rectified Linear Unit (ReLU)



Leaky ReLU



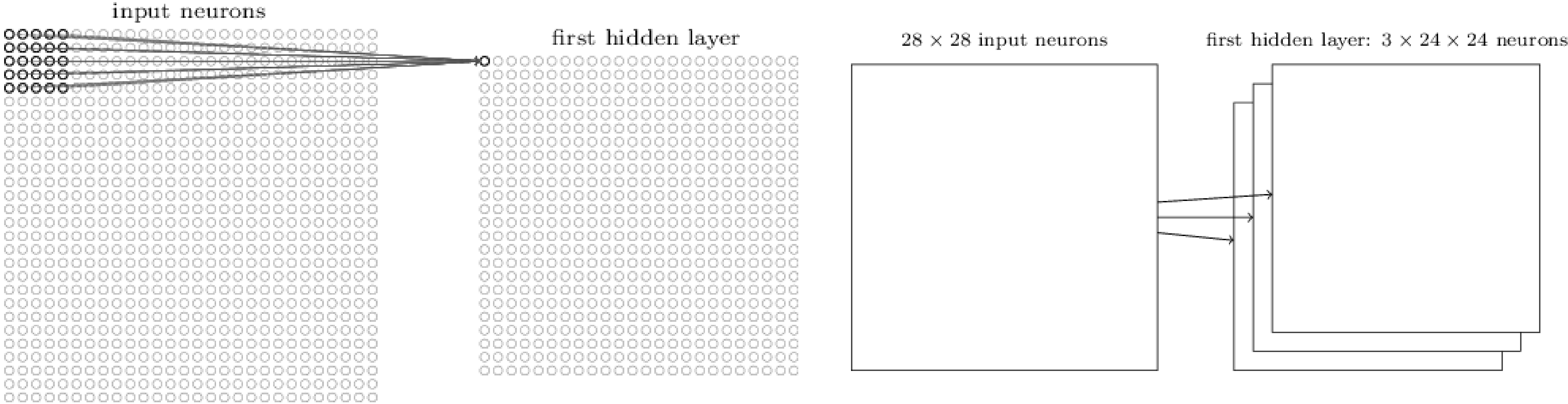
Exponential LU



From: https://arxiv.org/pdf/1703.09039.pdf

Convolutional-Neural-Network (CNN) (1)

Convolution-Layer

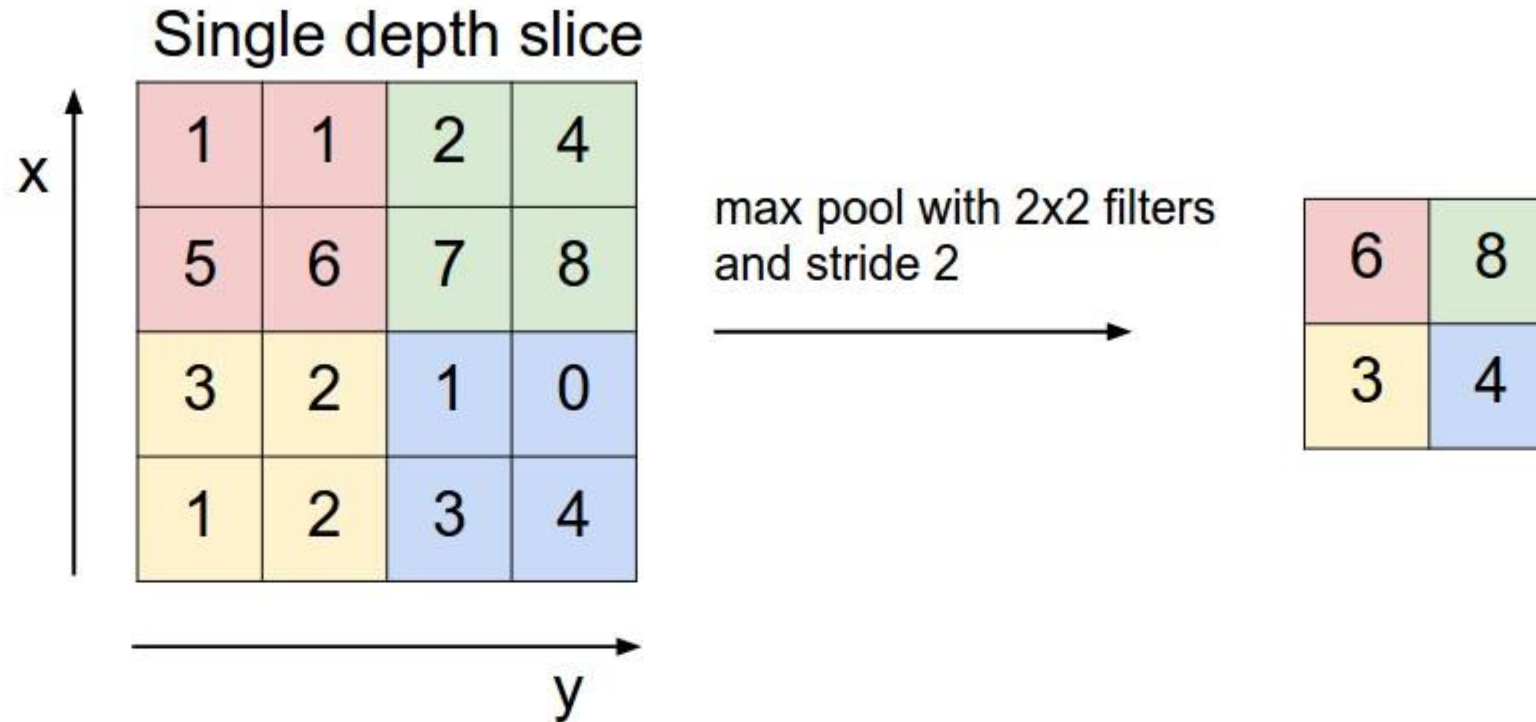


Simple summation. Introduces **spatial-locality** and **reduces** number of weights/parameters

Image from: http://neuralnetworksanddeeplearning.com/chap6.html

Convolutional-Neural-Network (CNN) (2)

Pooling-Layer



Summarizes spatial information -> extract **high-level** features

Image from: http://cs231n.github.io/neural-networks-1/

Convolutional-Neural-Network (CNN) (4)

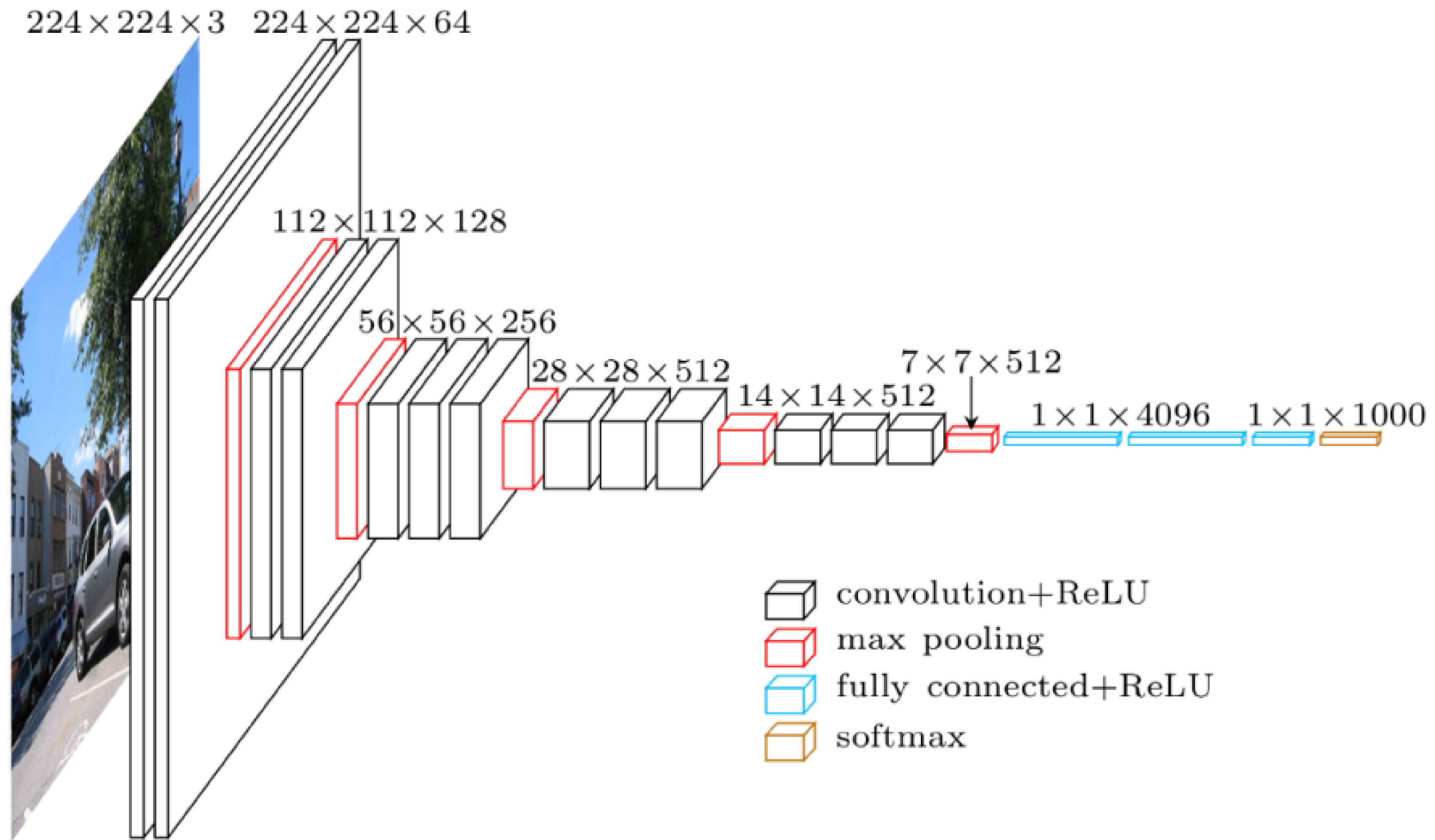
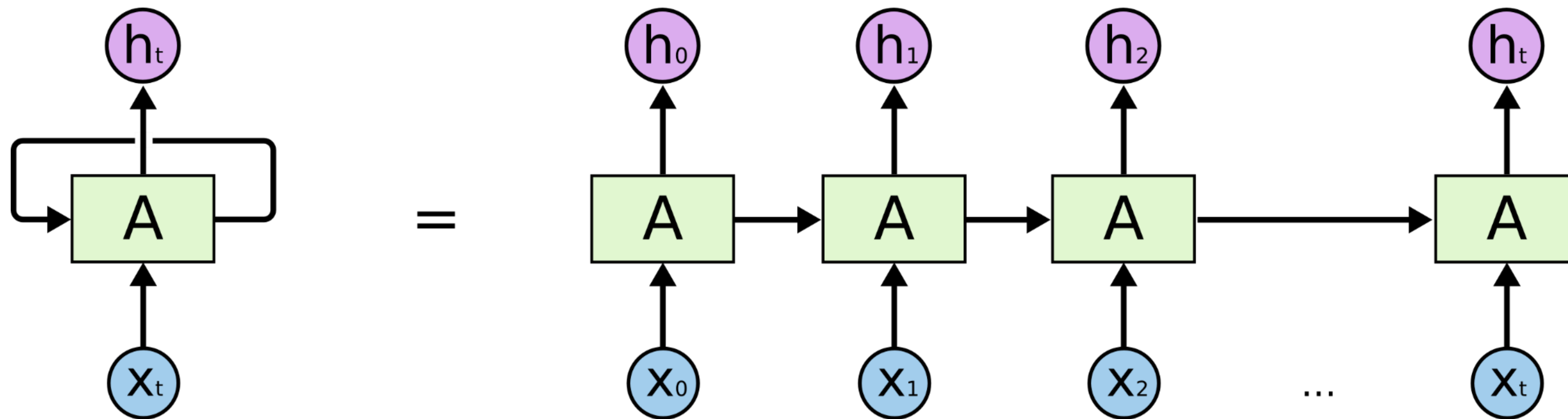


Image from: https://www.jeremyjordan.me/convnet-architectures/

Recurrent-Neural-Network (RNN) (1)



Introduce dependencies in **temporal/sequence** domain.

Image from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

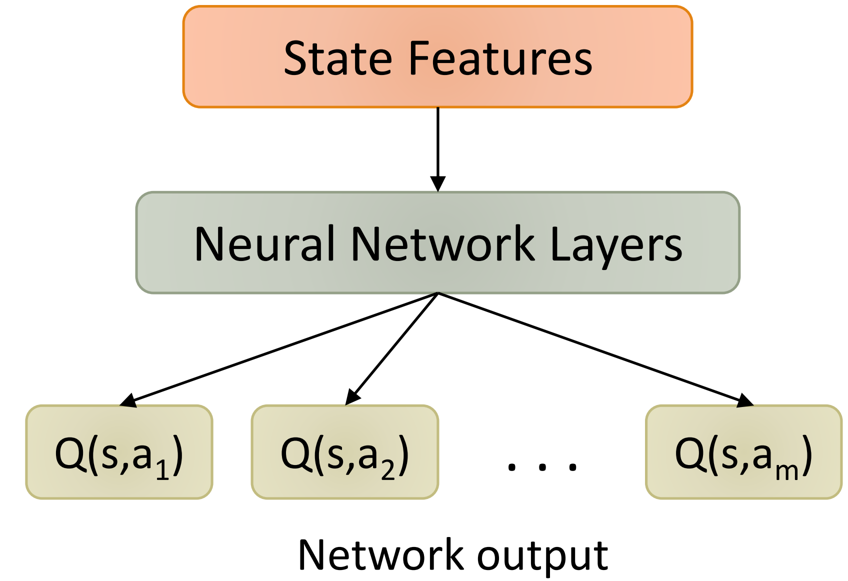
Part 3:
Value-based methods in Deep-RL

Fitted Q-Learning (Riedmiller 2005)

- The samples are of the form: < s, a, r, s' >.
- Approximate q_* by a function: $Q(s, a; \theta_k)$
- Target is: $Y_k = r + \gamma \max_{a' \in A} Q(s', a'; \theta_k)$
- Minimize the MSE using gradient descent:

$$L_{DQN} = (Q(s, a; \theta_k) - Y_k)^2$$

$$\theta_{k+1} = \theta_k + \alpha(Y_k - Q(s, a; \theta_k))\nabla_{\theta_k} Q(s, a; \theta_k)$$



- Update is applied after each sample. Can lead to slow convergence or **instability**.

"One reason for this is, that if weights are adjusted for one certain state action pair, then unpredictable changes also occur at other places in the state-action space." – Riedmiller 2005

Deep Q-Networks (Mnih et al. 2015)

- Successfully applied to ATARI games. To limit instabilities, it does:

1. Network weights θ_k are updated every iteration, but the target uses an "older" network via older weights θ_k^-. Every C iterations, we set $\theta_k^- = \theta_k$. $Y_k^{DQN} = r + \gamma \max_{a' \in A} Q(s', a'; \theta_k^-)$
2. In online setting, maintain a replay memory of N_R past samples. To update, it samples a mini-batch randomly from this memory. Helps reduce variance of updates compared to single sample.
3. Clips rewards between +1 and -1. Reasoning:

"As the scale of scores varies greatly from game to game, we clipped all positive rewards at 1 and all negative rewards at -1, leaving 0 rewards unchanged. Clipping the rewards in this manner limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games." – Mnih et. Al. 2015

- Deep learning techniques used: image preprocessing, normalization, CNN architectures, and optimizers.

Double DQN (Van Hasselt et al. 2016)

- Problem with this target: $Y_k^{DQN} = r + \gamma \max_{a' \in A} Q(s', a'; \theta_k^-)$
- A maximum over estimates is used as an estimate for the maximum -> positive bias.
- Example: true q(s, a) = 0 for all a, but $\hat{q}(s, a)$ noisy about 0 -> max is positive.
- Solution: **decouple** the **selection** of the maximum action and the **estimation** of the value of the maximum action.

Selection of maximum action using current weights

$$Y_k^{DDQN} = r + \gamma Q(s', \operatorname*{argmax}_{a \in A} Q(s', a; \theta_k); \theta_k^-)$$

Estimation of selected maximum using old weights

- "Overestimation combined with bootstrapping then has the pernicious effect of propagating the wrong relative information about which states are more valuable than others, directly affecting the quality of the learned policies." – Van Hasselt et. al. 2016
- "The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation." – Van Hasselt et. al. 2016

Dueling Network Architecture (1) (Wang et al. 2015)

- Makes use of Advantage function: $A(s,a) = Q(s,a) - V(s)$. How much better is it perform action *a* in state *s* versus following the policy?
- Network uses same target as DQN: $Y_k = r + \gamma \max_{a' \in A} Q(s', a'; \theta^{(1)}, \theta^{(2)}, \theta^{(3)})$
- Writes Q as:

Equal when a = argmax Q(s, a')

$$Q(s,a;\theta^{(1)},\theta^{(2)},\theta^{(3)}) = V\left(s;\theta^{(1)},\theta^{(3)}\right) + \left(A\left(s,a;\theta^{(1)},\theta^{(2)}\right) - \max_{a' \in \mathcal{A}} A\left(s,a';\theta^{(1)},\theta^{(2)}\right)\right)$$

Get closer to maximum advantage action

Dueling Network Architecture (2) (Wang et al. 2015)

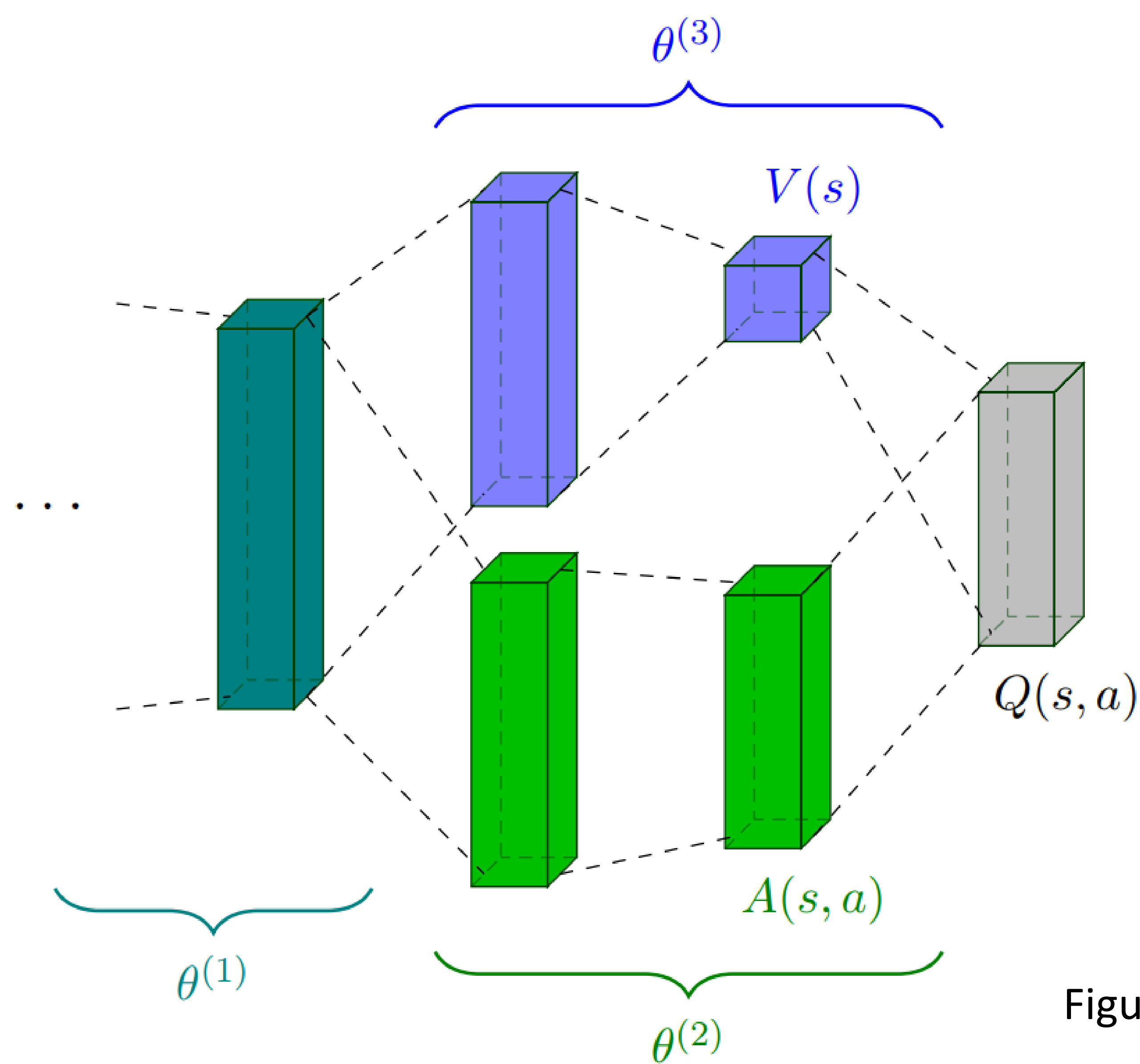


Figure from [2]

Dueling Network Architecture (3) (Wang et al. 2015)

- For stability uses this in practice:

$$Q(s,a;\theta^{(1)},\theta^{(2)},\theta^{(3)}) = V\left(s;\theta^{(1)},\theta^{(3)}\right) + \left(A\left(s,a;\theta^{(1)},\theta^{(2)}\right) - \frac{1}{|\mathcal{A}|}\sum_{a'\in\mathcal{A}} A\left(s,a';\theta^{(1)},\theta^{(2)}\right)\right)$$

Get closer to mean advantage action

Multi-step Targets

- Recall the DQN Target that uses bootstrapping: $Y_k^{DQN} = r + \gamma \max_{a' \in A} Q(s', a'; \theta_k^-)$
- n-step Target: $Y_k^{Q,n} = \sum_{t=0}^{n-1} \gamma^t r_t + \gamma^n \max_{a' \in A} Q(s_n, a'; \theta_k)$
- Weighted combination of multi-step targets called **truncated TD(λ)** target:

$$Y_k^{Q,n,\lambda} = (1-\lambda) \sum_{i=0}^{n-1} \lambda^i \left(\sum_{t=0}^{i} \gamma^t r_t + \gamma^{i+1} \max_{a' \in A} Q(s_{i+1}, a'; \theta_k) \right)$$

Normalizing so all weights sum to 1

Weight given to i^{th}-step target

i-step target

- For more details see Chapters 7 and 12 in Sutton & Barton book.

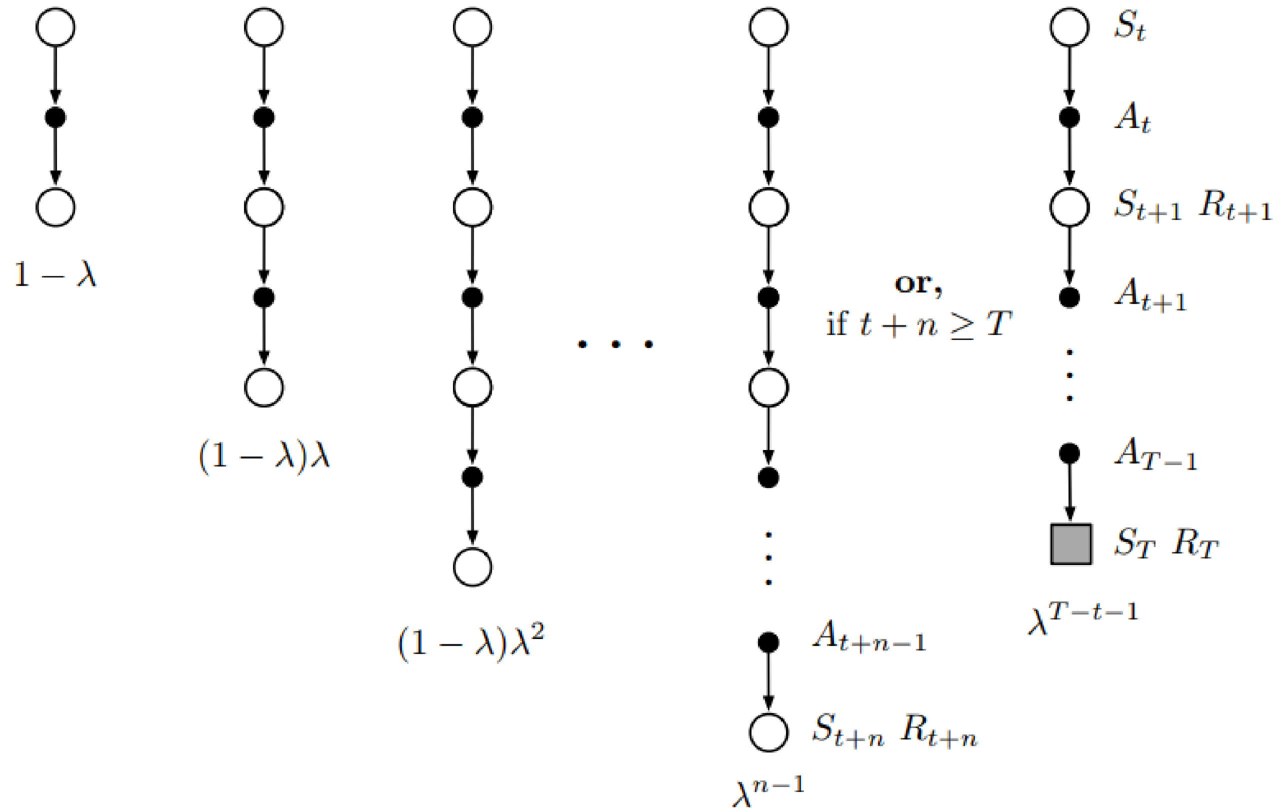
1-step TD and TD(0)

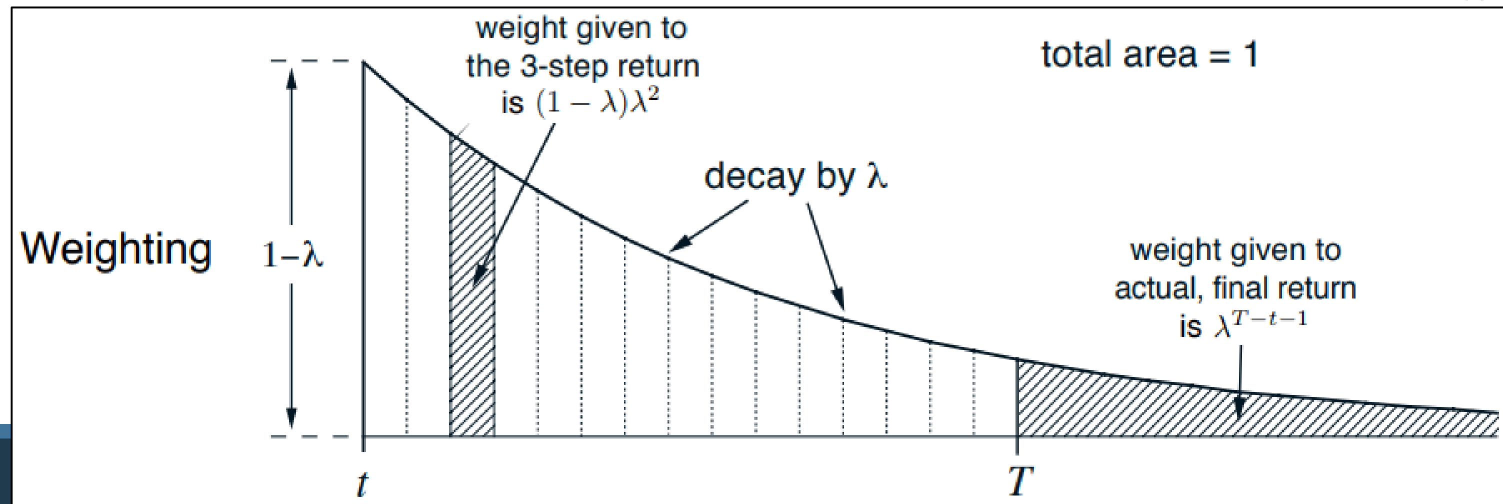
2-step TD

3-step TD

n-step TD

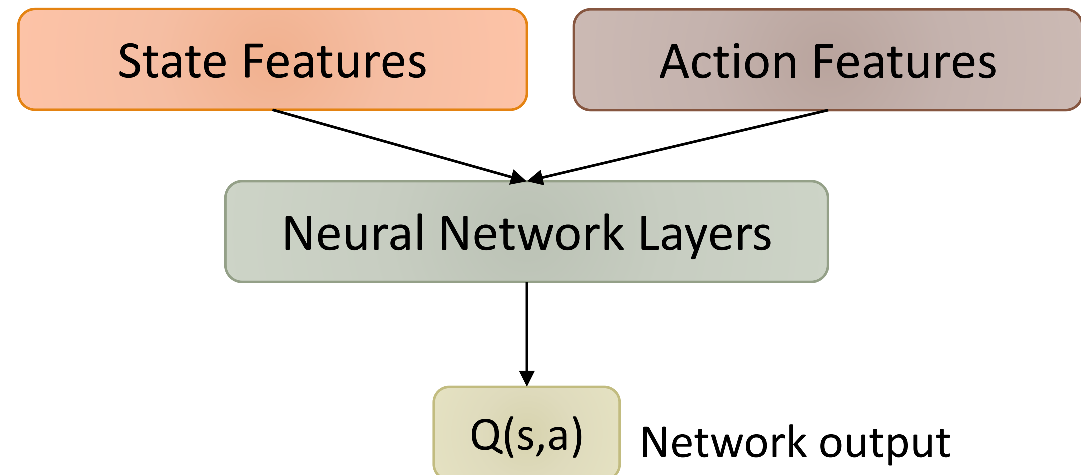
∞-step TD and Monte Carlo





Summary

- Hessel et al., 2017 achieved state-of-the-art performance with DQNs on ATARI benchmarks by combining the previous techniques: Double DQN, replay memory (prioritized), Dueling networks, multi-step targets, etc.
- Limitations of DQNs: large action spaces, continuous action spaces, explicit stochastic policies.
- Relatively simple to implement with open source frameworks: OpenAI Gym, Keras, Tensorflow. Demo at the end for DDQN with replay memory.
- For continuous action spaces, have seen networks that approximate Q(s, a) with states and actions as inputs into the network.



Part 4:
Policy-based methods in Deep-RL

Policy Gradient Methods Intro (1)

- Model the policy with a parametrized function: $\pi_\theta(a|s)$
- Maximize the objective function for episodic:

$$
\begin{aligned}
J(\theta) &= v_{\pi_\theta}(s_0) \\
&= \sum_a \pi_\theta(a|s_0) \sum_{s',r} p(s',r|s_0,a)\left[r + \gamma v_{\pi_\theta}(s')\right] \\
&= \sum_a \pi_\theta(a|s_0) q_{\pi_\theta}(s_0,a)
\end{aligned}
$$

- Policy Gradient Theorem:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta v_{\pi_\theta}(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} Pr(s_0 \to s, k, \pi_\theta) \right) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \\
&\propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s)
\end{aligned}
$$

n(s) is avg number of times state s is encountered in episode.

$\mu(s)$ is stationary distribution of policy:

$$\mu(s) = \lim_{\{t \to \infty\}} \Pr(s_t = s \mid s_0, \pi_\theta)$$

Policy Gradient Methods Intro (2)

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s)$$
$$= \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$$
$$= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}\left[q_{\pi_\theta}(s,a) \nabla_\theta \ln\left(\pi_\theta(a|s)\right)\right]$$

Note: Constant of proportionality is 1 in the continuous case, and average length of episode in episodic case. – see Ch. 13 Sutton and Barto

- This tells us that the gradient of the objective function can be estimated from samples. We just need to be able to compute $q_{\pi_\theta}(s,a)$ and $\nabla_\theta \ln \pi_\theta(a|s)$.

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} J(\theta_t)$$

- Detailed derivation of policy gradient theorem in Ch. 13 Sutton and Barto pg. 325. For continuous case, uses avg. rate of reward per step, but same conclusion on pg. 333.
- Even more detailed derivation at: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#Proof-of-Policy-Gradient-Theorem

REINFORCE Algorithm (1)

- Replace q by the return and use MC: $$\nabla_\theta J(\theta) = \mathbb{E}_{S_t \sim \mu, A_t \sim \pi_\theta}\left[q_{\pi_\theta}(S_t, A_t)\nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right]$$ $$= \mathbb{E}_{S_t \sim \mu, A_t \sim \pi_\theta}\left[G_t \nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right]$$ $$;\text{since } \mathbb{E}_{\pi_\theta}[G_t|S_t, A_t] = q_{\pi_\theta}(S_t, A_t)$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Figure from [1]

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$ ← Sampled quantity that equals gradient in expectation.

REINFORCE with Baseline (1)

- Problem: some states always have similar returns regardless of action. Thus gradient estimate will be similar for all actions. Couple this with variance problem of MC, makes it hard to differentiate between actions in states with similar rewards.
- Solution: subtract baseline performance. Gradient estimates still **unbiased** but **reduces** the variance.

$$\nabla_\theta J(\theta) = \mathbb{E}_{S_t \sim \mu, A_t \sim \pi_\theta}\left[(G_t - b(S_t))\nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right]$$

$$\text{Note: } \mathbb{E}\left[b(S_t)\nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right] = \sum_s \mu(s) b(s) \nabla_\theta \sum_a \pi_\theta(a|s)$$

$$= \sum_s \mu(s) b(s) \nabla_\theta 1 = 0$$

- A common baseline is $b(s) = V(s)$ the state-value function. Which we estimate by another parameter say: w.

REINFORCE with Baseline (2)

REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

Figure from [1]

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$ → v used as baseline.
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$ → Update state-value function approximation
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$ → Update policy function approximation

Online learning with Actor-Critic methods (1)

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{S_t\sim\mu, A_t\sim\pi_\theta}\left[\left(G_t - v(S_t)\right)\nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right] \\ &= \mathbb{E}_{S_t\sim\mu, A_t\sim\pi_\theta}\left[\left(q_{\pi_\theta}(S_t, A_t) - v_{\pi_\theta}(S_t)\right)\nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right] \\ &= \mathbb{E}_{S_t\sim\mu, A_t\sim\pi_\theta}\left[A_{\pi_\theta}(S_t, A_t)\nabla_\theta \ln\left(\pi_\theta(A_t|S_t)\right)\right]\end{aligned}$$

- So, advantage helps reduce the variance. Idea: estimate *q* and *v* functions with function approximations (parameters w, z) using TD(0). Then use those estimates to update the policy. $A_{\pi_\theta}(s,a) = q_w(s,a) - v_z(s)$. *Problem*: requires two set of parameters: w, z.
- Note that the TD(0) error with the **true** *v(s)* is an unbiased estimate of advantage:

$$\mathbb{E}_{\pi_\theta}\left[\underbrace{R_t + \gamma v_{\pi_\theta}(S_{t+1})}_{\text{TD(0)-target}} - \underbrace{v_{\pi_\theta}(s_t)}_{\text{baseline}}\middle| s_t, a_t\right] = \mathbb{E}_{\pi_\theta}\left[R_t + \gamma v_{\pi_\theta}(S_{t+1})\middle| s_t, a_t\right] - v_{\pi_\theta}(s_t)$$

$$= q_{\pi_\theta}(s_t, a_t) - v_{\pi_\theta}(s_t)$$

$$= A_{\pi_\theta}(s_t, a_t)$$

- This means we can use samples of the TD(0) target in the online setting to est. advantage.
- We only need to function appr. the state-value function *v*; i.e. **one** set of parameters instead of **two**. **Recall** that since we are using estimates of *v(s)*, the TD(0) target will be biased, but with reduced variance.

Online learning with Actor-Critic methods (2)

One-step Actor–Critic (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$ ← Sample action from stochastic policy; implicit exploration.
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$ ← Update v-function; critic
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$ ← Update policy; actor
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

Figure from [1]

Actor-Critic: Using Deep Learning

- We can approximate the policy with a neural network. A number of methods have been developed to limit instabilities, in addition to using replay memory and batch update. The gradient estimate given by:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

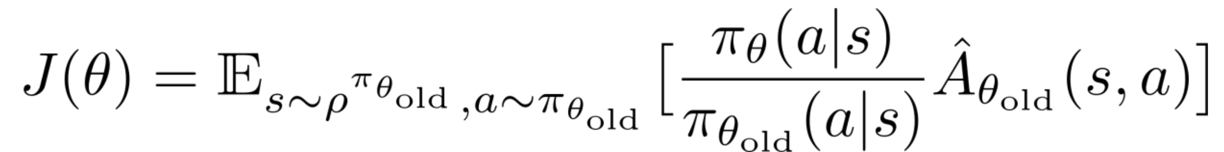
Empirical avg. over minibatch

Estimate of Advantage function (e.g. another network)

- From Schulman et. al. [2017]: "… empirically it often leads to destructively large policy updates (see Section 6.1; results are not shown but were similar or worse than the "no clipping or penalty" setting)."
- To limit instabilities, methods were proposed that limit/constrain parameter updates every step.

Actor-Critic: Trust Region Optimization (TRPO) [Schulman et. al. 2015]

Conservative Policy Iteration Objective [Kakade 2002]



On-policy version

Maximize $J(\theta)$ subject to:

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

- Maximization with no constraint can lead to large policy updates.
- Uses constraint on KL-divergence to limit the update of parameters so that they don't move away too much from the old policy.
- In paper proves monotonic improvement and speaks more on practical implementation.

Actor-Critic: Proximal Policy Optimization (PPO) [Schulman et. al. 2017]

- Simpler than TRPO, but achieves similar performance.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Conservative-Policy-Iteration objective $J^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_{\theta_{\text{old}}}(s_t, a_t)\right]$

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

- The clip function limits the amount of change away from 1. The min function takes a pessimistic lower bound for the objective. Ignores high improvements to the objective.

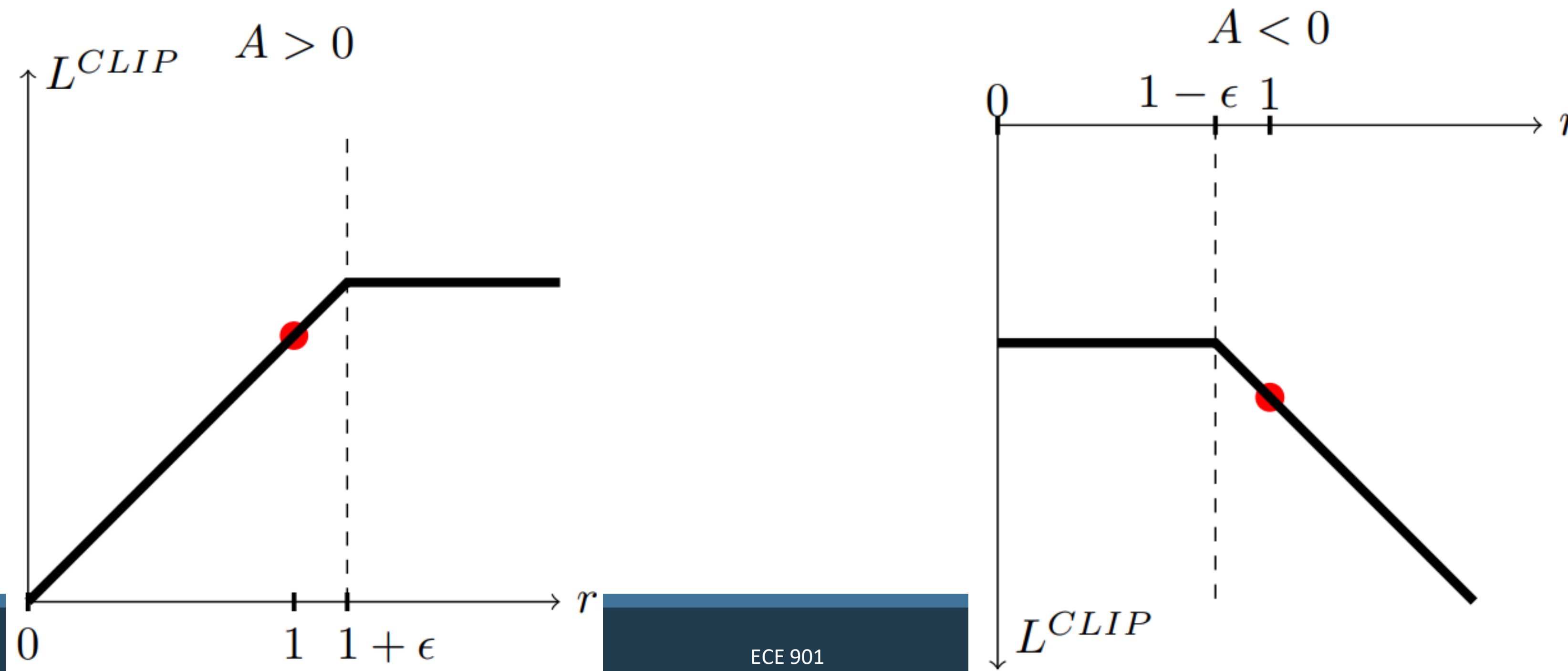


Figure from [Schulman et. al. 2017]

Actor-Critic: Proximal Policy Optimization (PPO) [Schulman et. al. 2017]

- Uses a single network with shared parameters to output: $V(s)$ and $\pi(a|s)$.

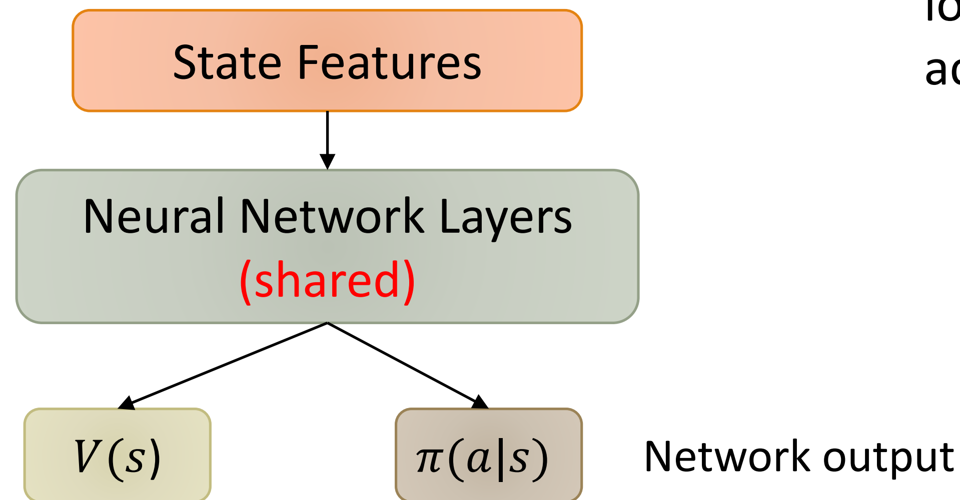
$$J^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_{\theta_{\text{old}}}(s_t, a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s_t, a_t))]$$

$$J^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t[J_t^{\text{CLIP}}(\theta) - c_1(V_\theta(s_t) - V_{\text{target}})^2 + c_2 S[\pi_\theta](s_t)]$$

Policy surrogate

Value function error

Entropy term to ensure sufficient exploration: $-\sum_a \pi(a|s)\log(\pi(a|s))$ Prevents policy from getting stuck in local minima and maximizing certain actions according to papers.



Network output

Actor-Critic: Proximal Policy Optimization (PPO) [Schulman et. al. 2017]

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

From [Schulman et. al. 2017]

Other policy gradient methods (1)

- **Deterministic policy methods** like Deterministic policy gradient (DPG) [Silver et. al. 2014] and Deep DPG [Lillicrap et. al. 2015]. Sutton and Barto book state that stochastic policies will approach deterministic policies (if they are optimal) asymptotically [pg. 322].
- **Parallel policy training** via simultaneous agents: Asynchronous Advantage Actor-Critic (A3C) [Mnih et al., 2016] and A2C (synchronous version). Speeds up training; agents can experience different parts of environment (on-policy); without replay memory.

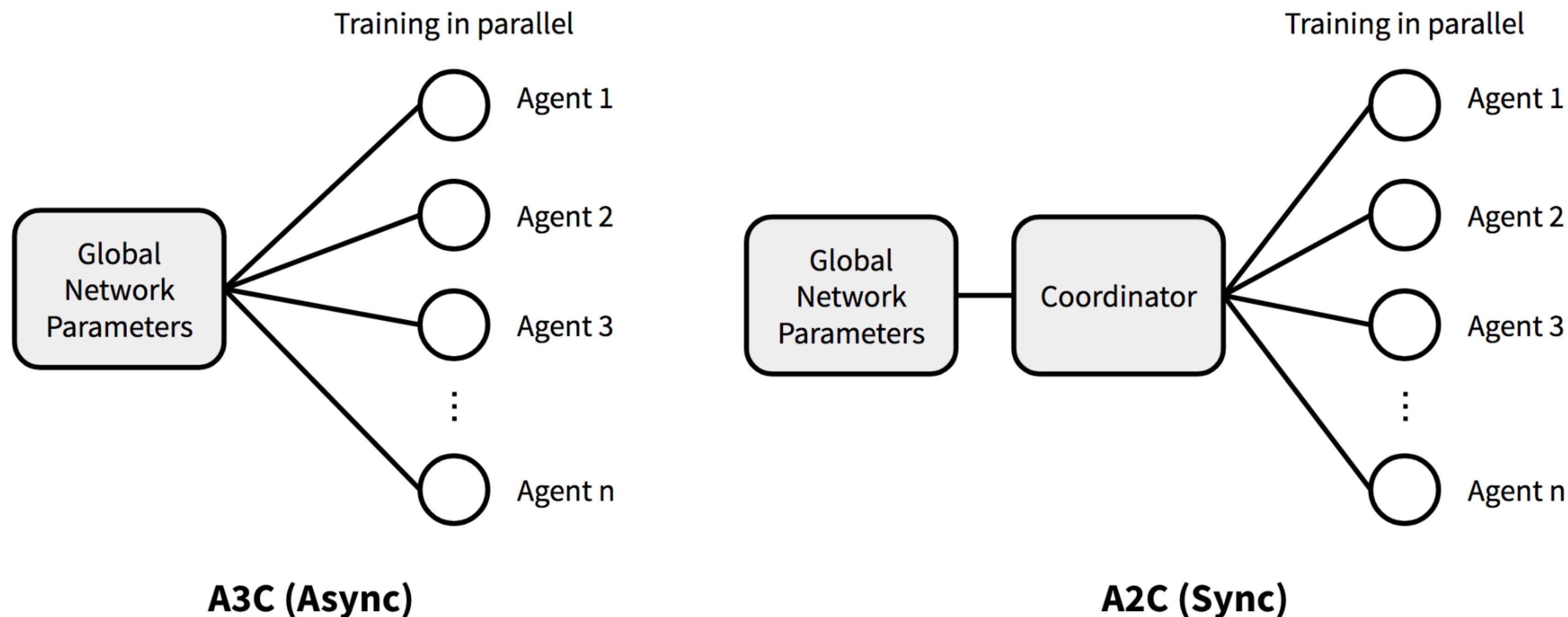


Figure from [3]

Part 5:
Frameworks and Success stories

Some success stories

- Some success stories with Deep RL:
  - Human level performance on ATARI games [Mnih et al 2015]
  - DeepMind’s AlphaZero for Chess, Shogi and Go [Silver et al 2017]
  - OpenAI’s Five neural networks for playing DOTA game. See details: https://openai.com/blog/openai-five/

OpenAI's Five

	OPENAI 1V1 BOT	OPENAI FIVE
CPUs	60,000 CPU cores on Azure	128,000 preemptible CPU cores on GCP
GPUs	256 K80 GPUs on Azure	256 P100 GPUs on GCP
Experience collected	~300 years per day	~180 years per day (~900 years per day counting each hero separately)
Size of observation	~3.3 kB	~36.8 kB
Observations per second of gameplay	10	7.5
Batch size	8,388,608 observations	1,048,576 observations
Batches per minute	~20	~60