Boulder

Reinforcement Learning

YouTube Playlist

Maziar Raissi

Assistant Professor

Department of Applied Mathematics

University of Colorado Boulder

maziar.raissi@colorado.edu

Playing Atari with Deep Reinforcement Learning

YouTube Video

$\mathcal{E} \rightarrow$ environment
actions, observations, rewards
$a_t \rightarrow$ action
$\mathcal{A} = \{1, \ldots, K\} \rightarrow$ set of legal actions
$x_t \in \mathbb{R}^d \rightarrow$ image
$r_t \rightarrow$ reward
partially observed task
$s_t = x_1, a_1, x_2, a_2, \ldots, a_{t-1}, x_t \rightarrow$ state
MDP: Markov Decision Process

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \rightarrow \text{future discounted return at time } t$$

$Q^*(s,a) \rightarrow$ optimal action-value function

$\pi \rightarrow$ policy mapping sequences s to actions
(distribution over actions)

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

Bellman Equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q^*(s',a') | s, a]$$

Value Iteration Algorithm

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s',a') | s, a]$$

$Q(s,a;\theta) \approx Q^*(s,a) \rightarrow$ Q-Network

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s,a;\theta_i))^2]$$

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s, a]$$

$y_i \rightarrow$ target for iteration i

$\rho(s,a) \rightarrow$ prob. distribution over sequences & actions
(behavior distribution)

model free: no explicit construction of an estimator for $\mathcal{E}$

off-policy: greedy strategy $a^* = \max_a Q(s,a;\theta)$

Experience Replay

$e_t = (s_t, a_t, r_t, s_{t+1}) \rightarrow$ agent's experience
$\mathcal{D} = e_1, e_2, \ldots, e_N \rightarrow$ replay memory
Input to Q-Network: state representation
Output of DQN: predicted Q-values of individual actions

Arcade Learning Environment (ALE)

Algorithm 1 Deep Q-learning with Experience Replay

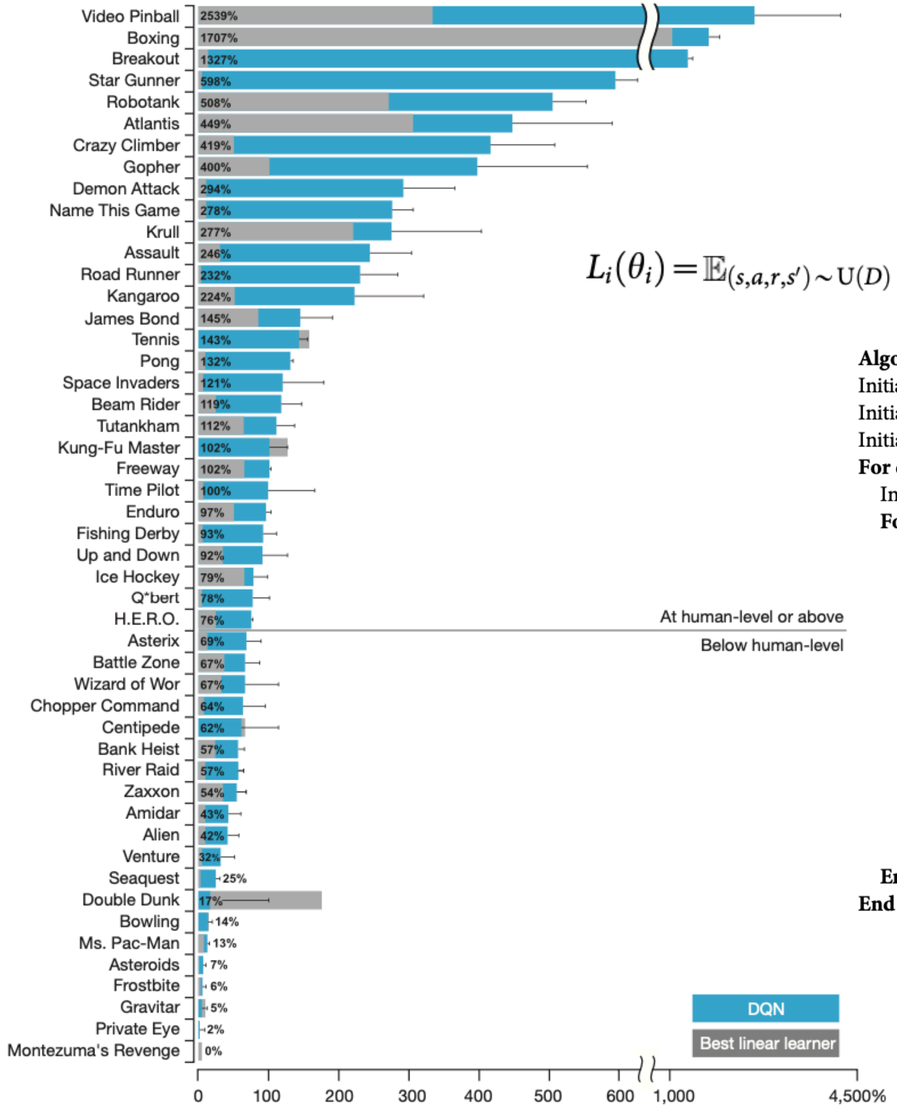
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    ϵ-greedy $\rightarrow$ { greedy action with prob. $1-\epsilon$; random action with prob. ϵ
    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 $\phi_{t+1} = \phi(s_{t+1})$ $\rightarrow$ last four frames of a history (stacked)
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Mnih, Volodymyr, et al. "Playing Atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

Human-level Control through Deep Reinforcement Learning

Boulder

YouTube Video



$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

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

    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

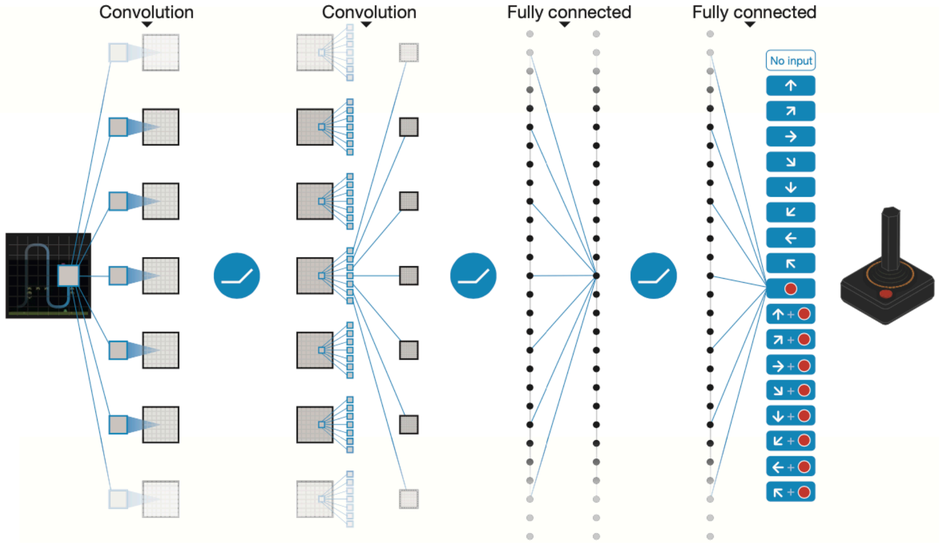
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

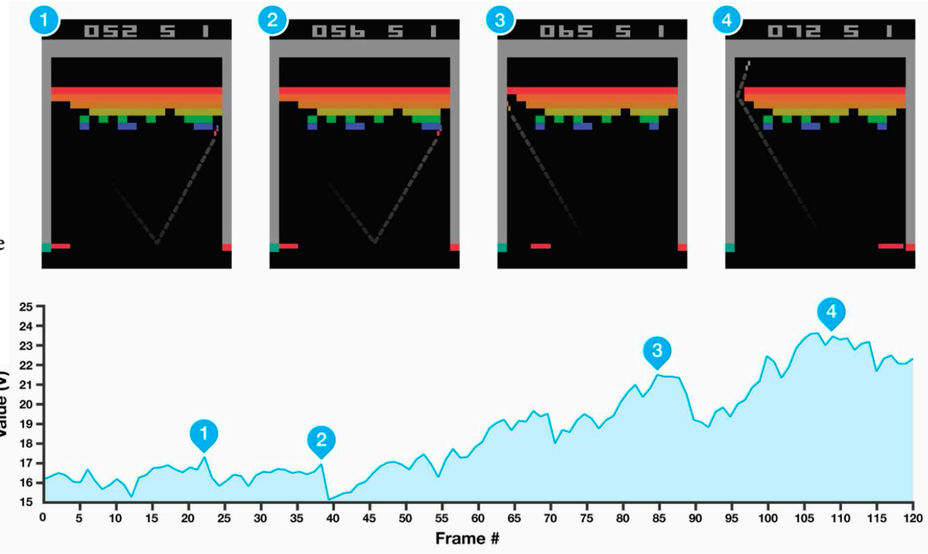
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

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

  End For

End For





Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *nature* 518.7540 (2015): 529-533.

Continuous Control with Deep Reinforcement Learning

YouTube Playlist

human arm $\rightarrow$ 7 degrees of freedom

$a_i \in \{-k, 0, k\}, i = 1, \ldots, 7$

$3^7 = 2187 \rightarrow$ dimensionality of action space

Background

$\mathcal{E} \rightarrow$ environment (maybe stochastic)

$t \rightarrow$ time step

$x_t \rightarrow$ observation

$a_t \rightarrow$ action $\quad a_t \in \mathbb{R}^N$

$r_t \rightarrow$ reward

$s_t = x_1, a_1, x_2, a_2, \ldots, a_{t-1}, x_t \rightarrow$ state

partially observed environment

$s_t = x_t \rightarrow$ fully observed environment

$\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$ — $\mathcal{S}$: states; $\mathcal{P}(\mathcal{A})$: prob. distributions; set of legal actions; π: policy

$p(s_1) \rightarrow$ initial state distribution

$p(s_{t+1}|s_t, a_t) \rightarrow$ transition dynamics

$r(s_t, a_t) \rightarrow$ reward function

MDP: Markov Decision Process

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \rightarrow \text{future discounted return}$$

$$J = \mathbb{E}_{r_i, s_i \sim \mathcal{E}, a_i \sim \pi}[R_1]$$

$\rho^\pi \rightarrow$ discounted state visitation distribution for a policy π

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim \mathcal{E}, a_{i>t} \sim \pi}[R_t | s_t, a_t]$$

action-value function: expected return after taking action a_t in state s_t and thereafter following policy π

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

Bellman Equation

$\mu : \mathcal{S} \rightarrow \mathcal{A}$

if deterministic target policy

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

It is therefore possible to learn Q^π off-policy

$\beta \rightarrow$ stochastic behavior policy

$$\mu(s) = \arg\max_a Q(s, a) \rightarrow \text{greedy policy}$$

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim \mathcal{E}}[(Q(s_t, a_t|\theta^Q) - y_t)^2]$$

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(S_{t+1})|\theta^Q)$$

⋆ replay buffer

⋆ separate target network

$\mu(s|\theta^\mu) \rightarrow$ actor function

$Q(s, a) \rightarrow$ critic (Q-Learning & Bellman Equation)

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

$$L(\theta^\mu) = -\mathbb{E}_{s_t \sim \rho^\beta}[Q(s_t, \underbrace{\mu(s_t|\theta^\mu)}_{a_t} |\theta^Q)]$$

Deterministic Policy Gradient (DPG) 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

https://www.youtube.com/watch?v=tJBIqkC1wWM&feature=youtu.be

Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

Trust Region Policy Optimization

YouTube Playlist

$(\mathcal{S}, \mathcal{A}, P, c, \rho_0, \gamma) \rightarrow$ infinite-horizon discounted Markov Decision Process (MDP)

$\mathcal{S} \rightarrow$ finite set of states

$\mathcal{A} \rightarrow$ finite set of actions

$P \rightarrow$ transition probability distribution

$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$

$c \rightarrow$ cost function

$c : \mathcal{S} \rightarrow \mathbb{R}$

$\rho_0 \rightarrow$ distribution of initial state s_0

$\rho_0 : \mathcal{S} \rightarrow \mathbb{R}$

$\gamma \in (0, 1) \rightarrow$ discount factor

$\pi \rightarrow$ stochastic policy

$\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

$\eta(\pi) \rightarrow$ expected discounted cost

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}[\sum_{t=0}^{\infty} \gamma^t c(s_t)]$$

$$s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

$Q_\pi(s_t, a_t) \rightarrow$ state-action value function

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}[\sum_{l=0}^{\infty} \gamma^l c(s_{t+l})]$$

$V_\pi(s_t) \rightarrow$ value function

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}[\sum_{l=0}^{\infty} \gamma^l c(s_{t+l})]$$

$A_\pi(s_t, a_t) \rightarrow$ advantage function

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

$\rho_\pi(s) \rightarrow$ (unnormalized) discounted visitation frequencies

$$\rho_\pi(s) = (P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots)$$

where $s_0 \sim \rho_0$ and the actions are chosen according to π

Algorithm 1 Approximate policy iteration algorithm guaranteeing non-increasing expected cost η

Initialize π_0. See the paper for proof!

for $i = 0, 1, 2, \ldots$ until convergence **do**

Compute all advantage values $A_{\pi_i}(s, a)$.

Solve the constrained optimization problem

$$\pi_{i+1} = \arg\min_{\pi} \left[L_{\pi_i}(\pi) + \left(\overbrace{\frac{2\epsilon\gamma}{(1-\gamma)^2}}^{C} \right) D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

where $\epsilon = \max_s \max_a |A_\pi(s, a)|$

and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

$D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathrm{KL}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s))$

In practice, if we used the penalty coefficient C above recommended by the theory, the step sizes would be very small.

$\eta(\theta) := \eta(\pi_\theta)$

$L_\theta(\tilde{\theta}) := L_{\pi_\theta}(\pi_{\tilde{\theta}})$, and $D_{\mathrm{KL}}(\theta \parallel \tilde{\theta}) := D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\tilde{\theta}})$

$\overline{D}_{\mathrm{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s \sim \rho}[D_{\mathrm{KL}}(\pi_{\theta_1}(\cdot|s) \parallel \pi_{\theta_2}(\cdot|s))]$

$$\underset{\theta}{\text{minimize}}\, L_{\theta_{\text{old}}}(\theta)$$ trust region constraint

$$\text{subject to } \overline{D}_{\mathrm{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

$$\underset{\theta}{\text{minimize}} \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s, a)$$

$$\text{subject to } \overline{D}_{\mathrm{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

$$\sum_s \rho_{\theta_{\text{old}}}(s)[\ldots] = \frac{1}{1-\gamma} \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}}[\ldots]$$

$$\sum_a \pi_\theta(a|s_n) A_{\theta_{\text{old}}}(s_n, a) =$$

importance sampling

$$\mathbb{E}_{a \sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{\text{old}}}(s_n, a) \right]$$

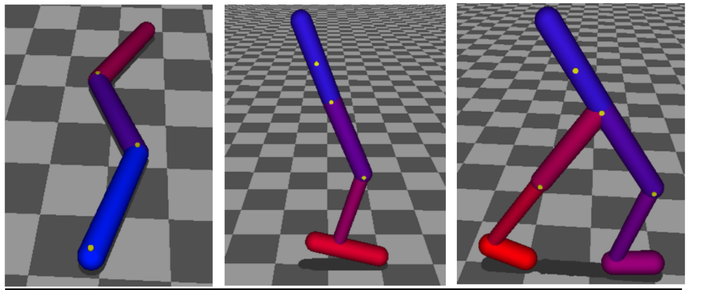
Replace advantage values by Q-values which only changes the objective by a constant.

$$\underset{\theta}{\text{minimize}}\, \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}, a \sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{\text{old}}}(s, a) \right]$$

$$\text{subject to } \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}}[D_{\mathrm{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s) \parallel \pi_\theta(\cdot|s))] \leq \delta.$$

$q(a|s) = \pi_{\theta_{\text{old}}}(a|s)$

TRPO



http://sites.google.com/site/trpopaper/

Schulman, John, et al. "Trust region policy optimization." *International conference on machine learning*. 2015.

Conjugate Gradient Method

YouTube Playlist

TRPO objective function

$$\pi_{k+1} = \arg\max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \text{ s.t. } \bar{D}_{KL}(\pi' || \pi_k) \leq \delta$$

Linear Approximation

$$\mathcal{L}_{\theta_k}(\theta) \approx \mathcal{L}_{\theta_k}(\theta_k) + g^T(\theta - \theta_k)$$

Quadratic Approximation

$$\bar{D}_{KL}(\theta||\theta_k) \approx \bar{D}_{KL}(\theta_k||\theta_k) + \nabla_\theta \bar{D}_{KL}(\theta||\theta_k)|_{\theta_k}(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)$$

$$\mathcal{L}_{\theta_k}(\theta) \approx g^T(\theta - \theta_k) \qquad g \doteq \nabla_\theta \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$$

$$\bar{D}_{KL}(\theta||\theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \qquad H \doteq \nabla^2_\theta \bar{D}_{KL}(\theta||\theta_k)|_{\theta_k}$$

$$\theta_{k+1} = \arg\max_\theta g^T(\theta - \theta_k) \text{ s.t. } \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta$$

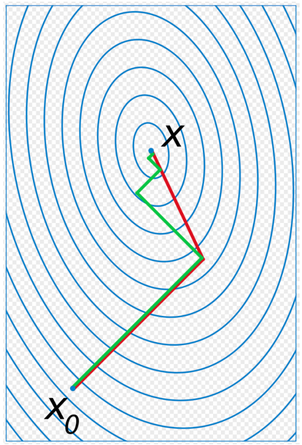
$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

natural policy gradient

$$H^{-1}g = x \implies Hx = g$$

$$Hx = \nabla_\theta \left(\left(\nabla_\theta \bar{D}_{KL}(\theta||\theta_k) \right)^T x \right)$$

$$\arg\min_x f(x) = \frac{1}{2} x^T H x - x^T g \qquad \nabla f(x) = Hx - g$$



Conjugate gradient method

$r_0 = g - Hx_0 = -\nabla f(x_0)$

$p_0 = r_0$

$x_1 = x_0 + \alpha_0 p_0$

$f(x_1) = f(x_0 + \alpha_0 p_0) =: g(\alpha_0)$

$$g'(\alpha_0) = 0 \implies \alpha_0 = \frac{p_0^T(g - Hx_0)}{p_0^T H p_0}$$

$r_1 = g - Hx_1 = -\nabla f(x_1)$

$$p_1 = r_1 - \frac{p_0^T H r_1}{p_0^T H p_0} p_0 \rightarrow \text{Gram-Schmidt orthonormalization}$$

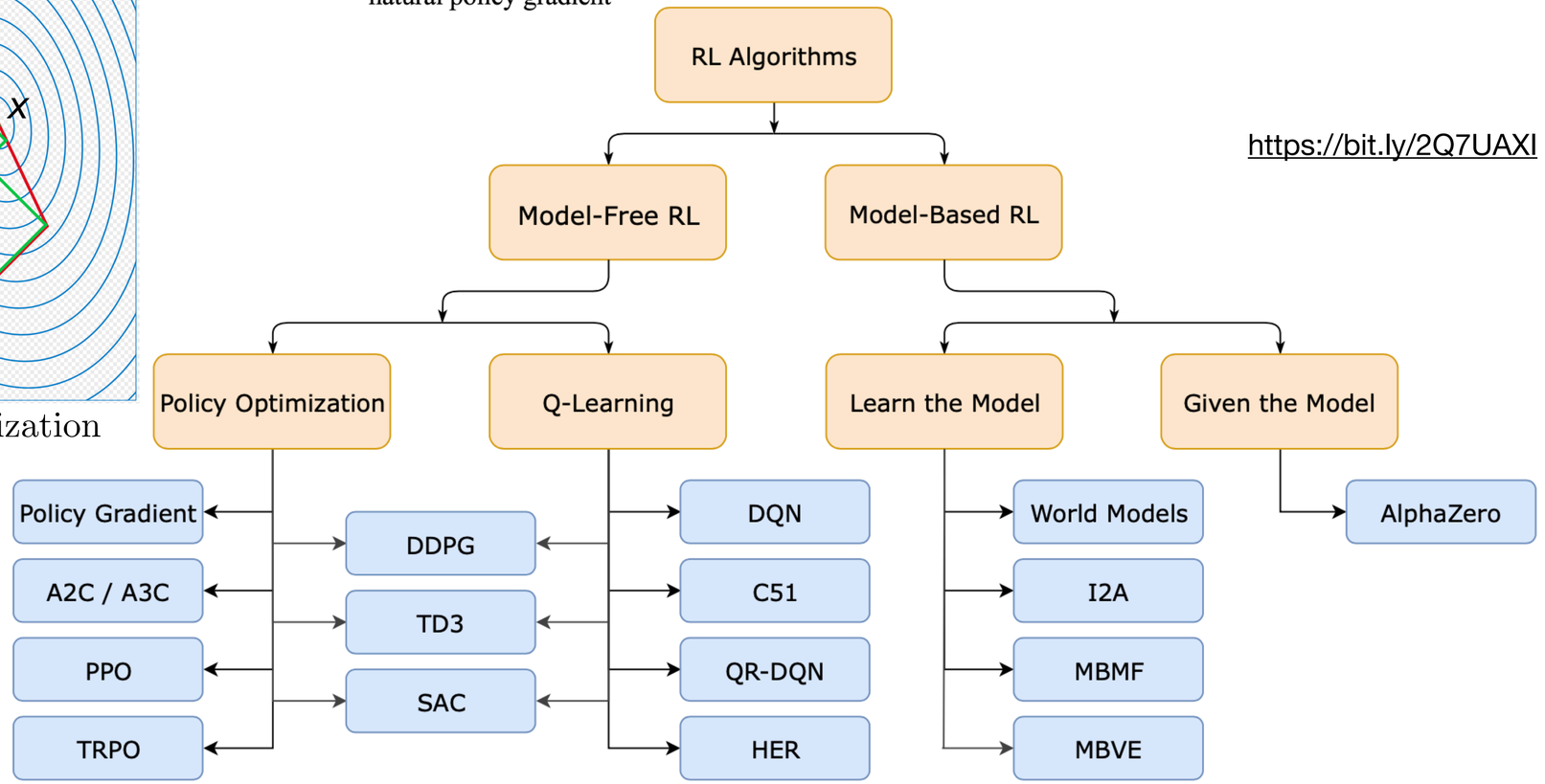
$$p_0^T H p_1 = p_0^T H r_1 - \frac{p_0^T H r_1}{p_0^T H p_0} p_0^T H p_0 = 0$$

$\implies p_0$ and p_1 are conjugate w.r.t. H.

$x_2 = x_1 + \alpha_1 p_1$

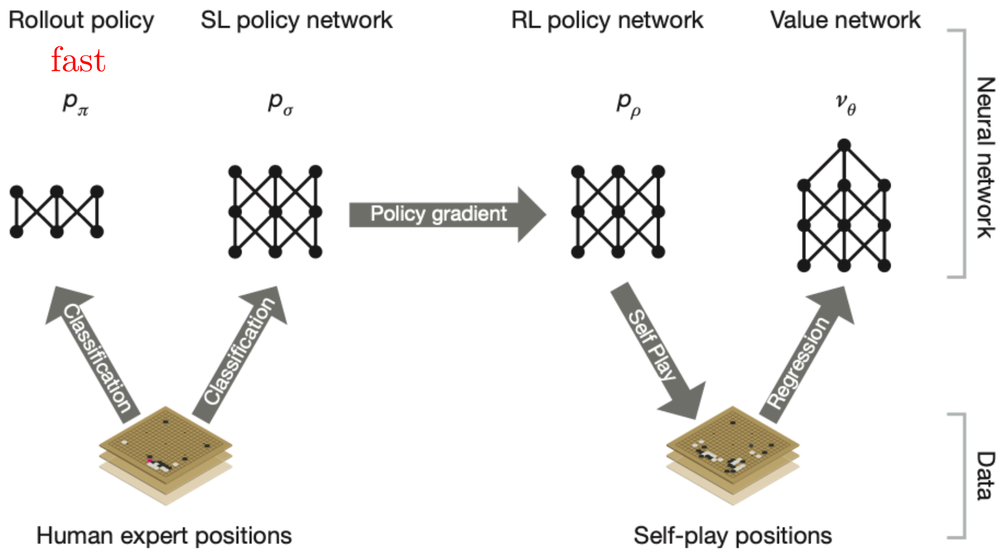
$$\alpha_1 = \frac{p_1^T(g - Hx_1)}{p_1^T H p_1}$$

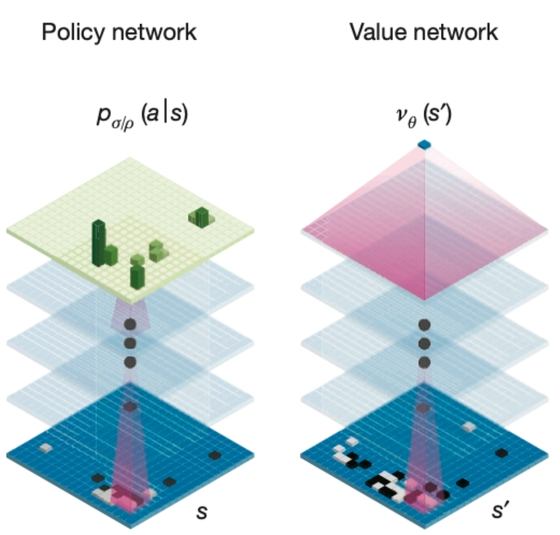
https://bit.ly/2Q7UAXI

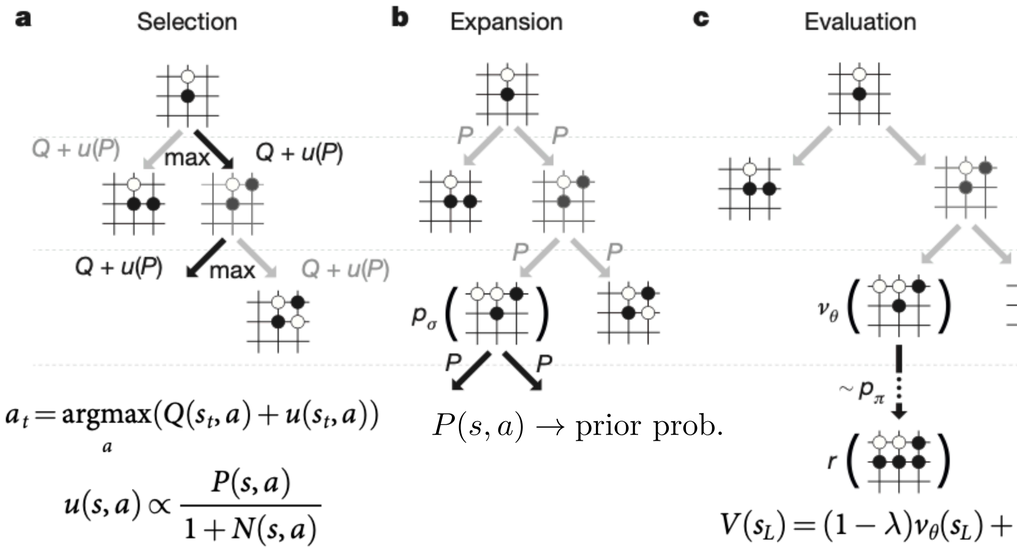


Mastering the game of Go with deep neural networks and tree search

YouTube Video







$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$P(s,a) \rightarrow$ prior prob.

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

Supervised learning of policy networks

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma} \rightarrow \text{stochastic gradient ascent (likelihood)}$$

Reinforcement learning of policy networks

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho} z_t \rightarrow \text{stochastic gradient ascent (expected outcome)}$$

policy gradient reinforcement learning

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho}(z_t - v(s_t)) \rightarrow \text{REINFORCE}$$

variance reduction

$$z_t = \sum_{t'=t+1}^{T} r_{t'} = r_T \text{ since } r_{t'} = 0 \text{ for } t' < T$$

https://bit.ly/31OImVQ

$$r_T = \begin{cases} +1 & \text{winning} \\ -1 & \text{losing} \end{cases}$$

https://bit.ly/2RCWHn1

Reinforcement learning of value networks

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s)) \rightarrow \text{stochastic gradient descent (MSE: mean squared error)}$$

Monte Carlo tree search (MCTS)

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i) \rightarrow \text{visit count}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i) \rightarrow \text{action value}$$

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *nature* 529.7587 (2016): 484-489.

Asynchronous Methods for Deep Reinforcement Learning

YouTube Video

value-based model-free methods

$Q(s, a; \theta) \rightarrow$ approximate action-value function

$Q^*(s, a) \approx Q(s, a; \theta) \rightarrow$ Q-Learning

$$L_i(\theta_i) = \mathbb{E}[\underbrace{(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})}_{\text{one-step return}} - Q(s, a; \theta_i))^2]$$

$$\underbrace{r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \max_a \gamma^n Q(s_{t+n}, a)}_{n\text{-step return}}$$

A single reward directly affects the values of n preceding state action pairs

policy-based model-free methods

$\pi(a|s; \theta)$

gradient ascent on $\mathbb{E}[R_t]$

$$\underbrace{\nabla_\theta \log \pi(a_t|s_t; \theta) R_t}_{\text{as an unbiased estimate of } \nabla_\theta \mathbb{E}[R_t]} \rightarrow \text{REINFORCE}$$

$$\nabla_\theta \log \pi(a_t|s_t; \theta)(R_t - \underbrace{b_t(s_t)}_{\text{baseine}})$$

$b_t(s_t) \approx V^\pi(s_t)$

$R_t - b_t \approx$ advantage of action a_t in state s_t

$$A(a_t, s_t) = \underbrace{Q(a_t, s_t)}_{\approx R_t} - \underbrace{V(s_t)}_{\approx b_t} \qquad \underbrace{\text{actor}}_{\pi} - \underbrace{\text{critic}}_{b}$$

Asynchronous Advantage Actor-Critic (A3C)

Algorithm Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
https://www.youtube.com/watch?v=0xo1Ldx3L5Q&feature=youtu.be
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
https://www.youtube.com/watch?v=Ajjc08-iPx8&feature=youtu.be
  repeat
    Perform a_t according to policy $\pi(a_t|s_t; \theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
https://www.youtube.com/watch?v=nMR5mjCFZCw&feature=youtu.be
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

$$\nabla_\theta \log \pi(a_t|s_t; \theta) \underbrace{A(s_t, a_t; \theta, \theta_v)}_{\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)} + \beta \nabla_\theta \underbrace{H(\pi(s_t; \theta))}_{\text{entropy}}$$

Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." *International conference on machine learning*. 2016.

Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning

YouTube Video

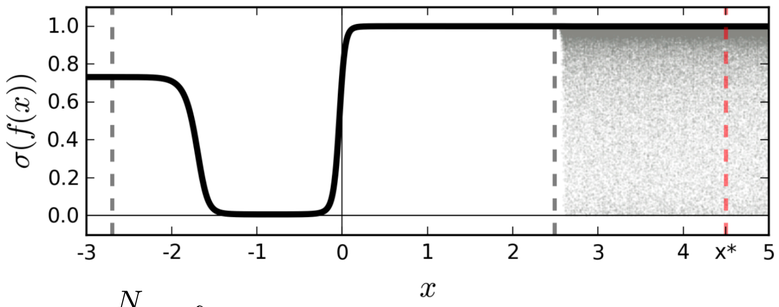
$$p(y|x, W^1, W^2) = \text{softmax}(W^2 \text{ReLU}(W^1 x))$$

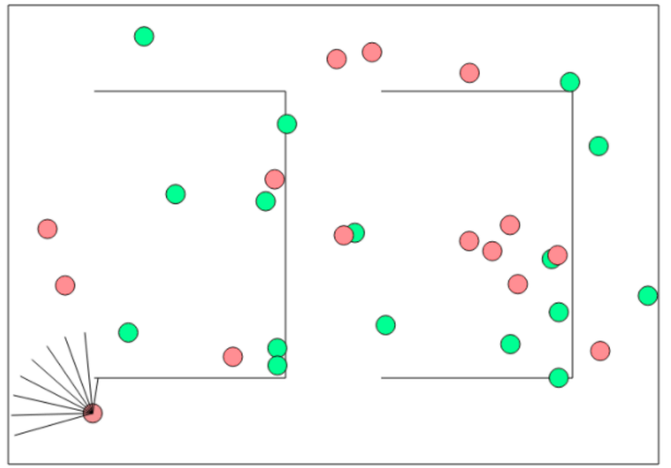
$$x \in \mathbb{R}^{d_0}, W^1 \in \mathbb{R}^{d_1 \times d_0}, W^2 \in \mathbb{R}^{d_2 \times d_1}$$

$$p(W^1) = \mathcal{N}(0, I), p(W^2) = \mathcal{N}(0, I)$$

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, W^1, W^2) \underbrace{p(W^1, W^2|\mathcal{D})}_{\text{not tractable}} dW^1 dW^2$$

$\mathcal{D} \rightarrow$ dataset





$$KL[q(W^1, W^2)||p(W^1, W^2|\mathcal{D})] \leq KL[q(W^1, W^2)||p(W^1)p(W^2)] - \sum_{n=1}^{N} \int q(W^1, W^2) \log p(y_n|x_n, W^1, W^2)$$

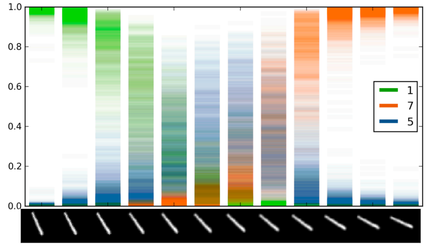
$$q(W^1, W^2) = q(W^1)q(W^2) \rightarrow \text{Variational Distribution}$$

$$q(W^\ell) = p^\ell \mathcal{N}(M^\ell, \sigma^2 I) + (1 - p^\ell)\mathcal{N}(0, \sigma^2 I) \rightarrow \text{Gaussian mixture distribution}$$

$$KL[q(W^\ell)||p(W^\ell)] \approx d^{\ell-1} d^\ell (\sigma^2 - \log(\sigma^2) - 1) + \frac{p^\ell}{2} \|W^\ell\|_2^2 + \text{constant}$$

$$\sigma = 10^{-33} \implies \log(\sigma) = -76$$

$$W^\ell \approx \text{diag}(z^\ell) M^\ell, M^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, z^\ell \sim \text{Bernoulli}(p^\ell)$$

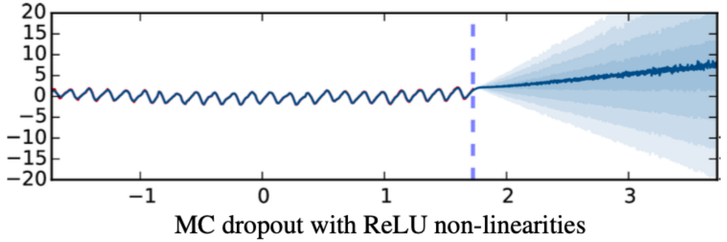


(b) Softmax *output* scatter

$$KL[q(W^1, W^2)||p(W^1, W^2|\mathcal{D})] \leq -\sum_{n=1}^{N} \log p(y_n|x_n, W_n^1, W_n^2) + \frac{p^1}{2} \|M_1\|_2^2 + \frac{p^2}{2} \|M_2\|_2^2$$

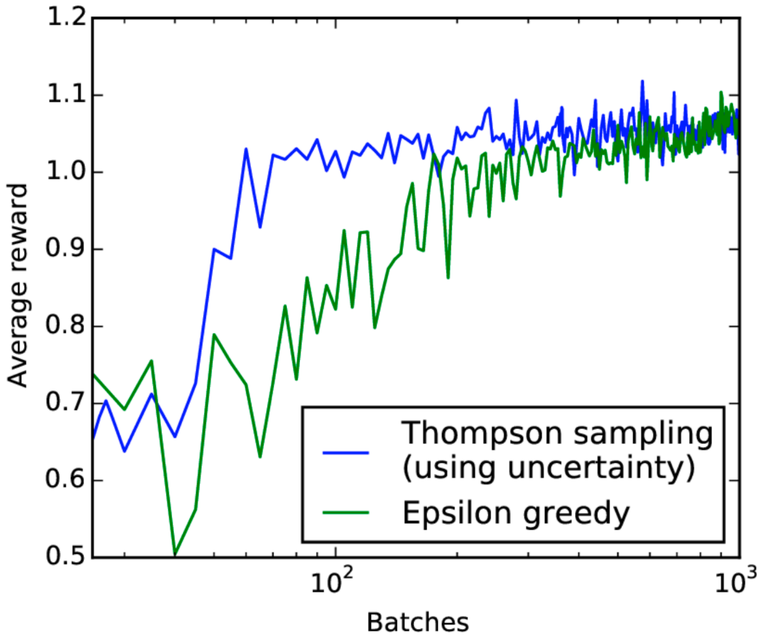
$W_n^\ell \approx \text{diag}(z_n^\ell) M^\ell \rightarrow$ Monte Carlo

Same loss as the one used by dropout!


MC dropout with ReLU non-linearities

$$p(y^*|x^*, \mathcal{D}) \approx \int p(y^*|x^*, W^1, W^2) q(W^1) q(W^2) dW^1 dW^2$$

Use Monte Carlo to obtain the mean and variance of the predictions!



Gal, Yarin, and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." *international conference on machine learning*. 2016.

Deep Reinforcement Learning with Double Q-Learning

YouTube Video

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Bellman Optimality Equation

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

Value Iteration Algorithm

$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{s'} Q_k(s',a')]$$

Q-Value Iteration Algorithm

$$V_{k+1}(s) = (1-\alpha)V_k(s) + \alpha(r + \gamma V_k(s'))$$
$$= V_k(s) + \alpha \underbrace{(r + \gamma V_k(s') - V_k(s))}_{\delta_k(s,r,s')}$$

Temporal Difference Learning

$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha(r + \gamma \max_{a'} Q_k(s',a'))$$
$$= Q_k(s,a) + \alpha(r + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a))$$

Q-Learning

Q-Learning is a form of Temporal Difference Learning

Q-Learning overestimates action values

$$Q_\pi(s,a) \equiv \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi]$$

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

$$Q(s,a;\theta_t)$$

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t,A_t;\theta_t))\nabla_{\theta_t} Q(S_t,A_t;\theta_t)$$

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1},a;\theta_t)$$

$$Q(S_t,A_t;\theta_{t+1}) \approx Q(S_t,A_t,\theta_t) + \alpha(Y_t^Q - Q(S_t,A_t;\theta_t))\|\nabla_{\theta_t} Q(S_t,A_t;\theta_t)\|^2$$

Taylor Series Expansion

Deep Q Networks

⋆ target network

⋆ experience replay

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1},a;\theta_t^-)$$

Double Q-Learning

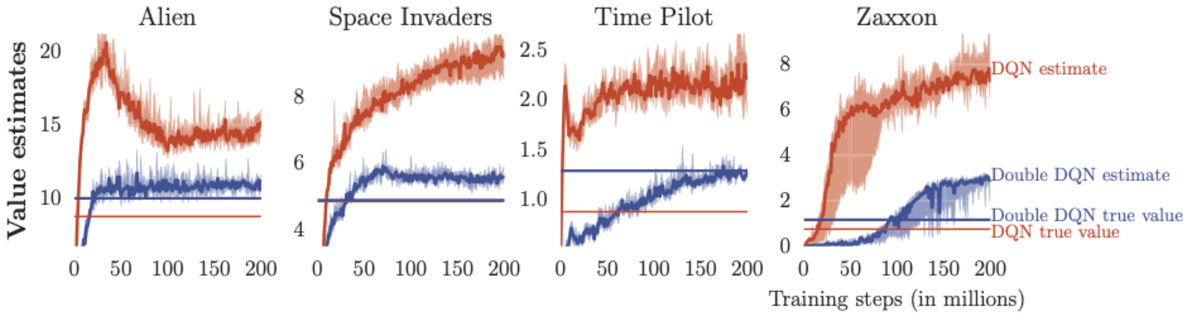
⋆ decouple selection from evaluation

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t);\theta_t)$$

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t);\theta_t') \rightarrow \text{two value functions}$$

Double DQN

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t);\theta_t^-) \rightarrow \text{target network}$$

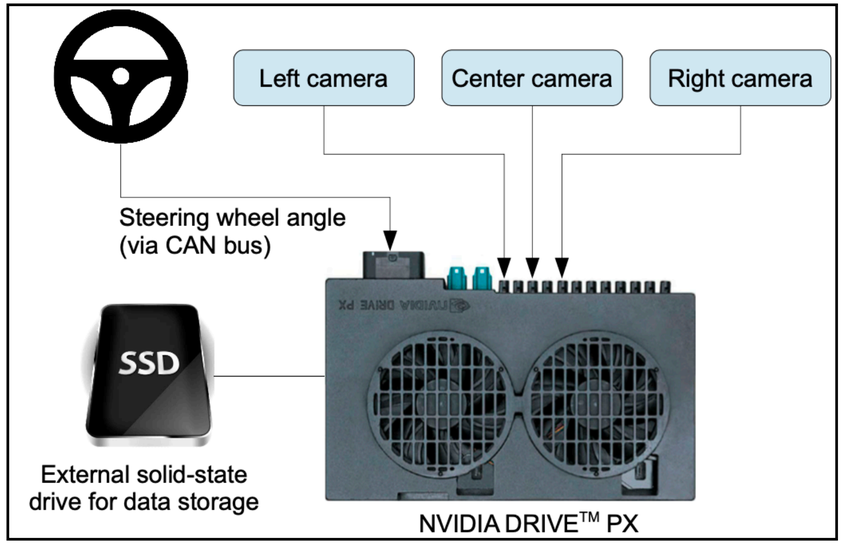


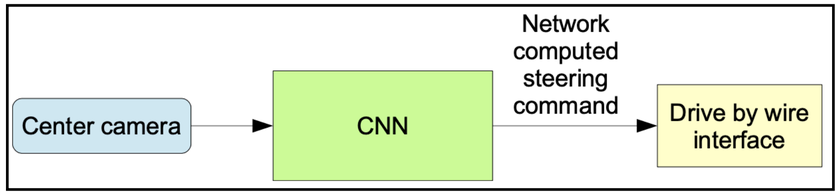
Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." *arXiv preprint arXiv:1509.06461* (2015).

End to End Learning for Self-Driving Cars

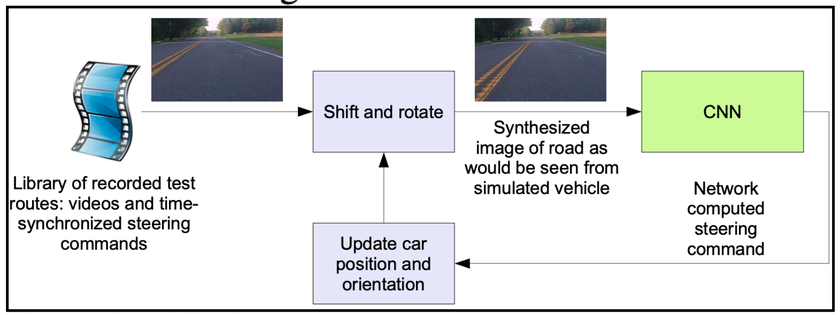
YouTube Video

High-level view of the data collection system.





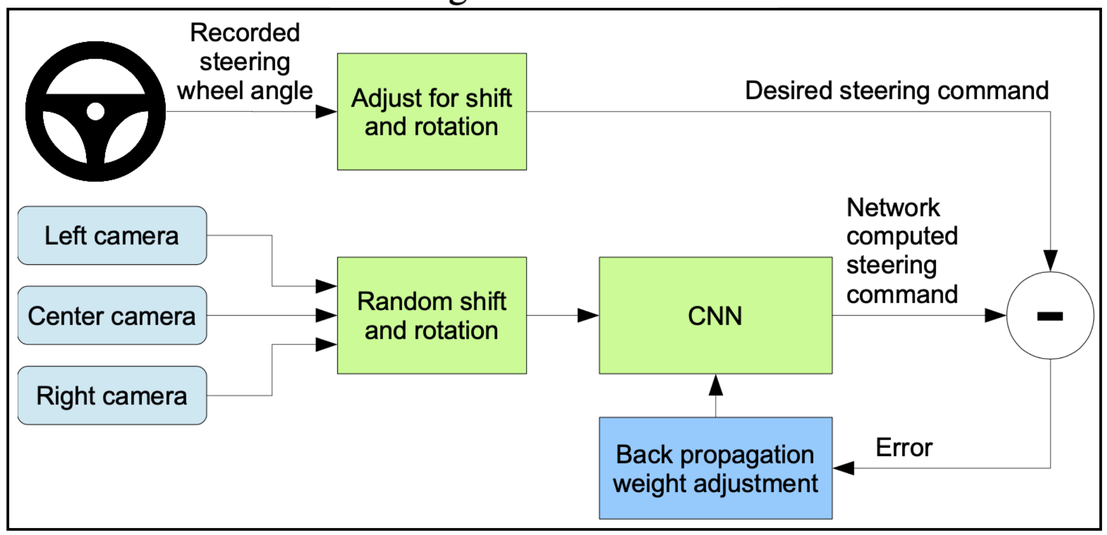
Block-diagram of the drive simulator.

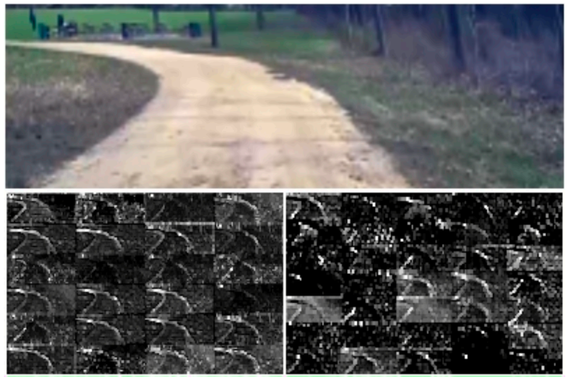


Evaluation Metric

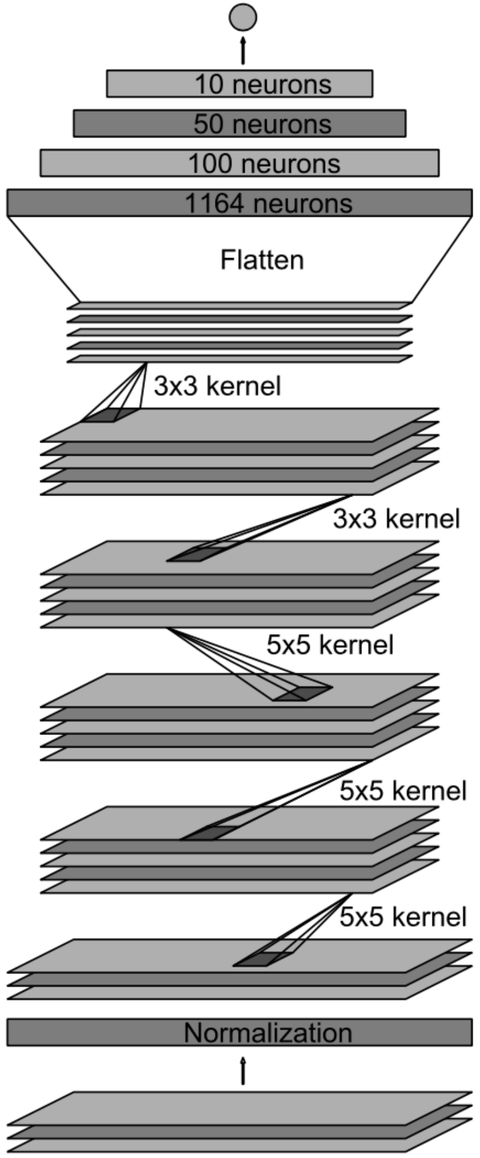
$$\text{autonomy} = (1 - \frac{(\text{number of interventions}) \cdot 6 \text{ seconds}}{\text{elapsed time [seconds]}}) \cdot 100$$

Training the neural network.





Activations of the first & second layer feature maps.



Output: vehicle control

Fully-connected layer
Fully-connected layer
Fully-connected layer

Convolutional feature map 64@1x18

Convolutional feature map 64@3x20

Convolutional feature map 48@5x22

Convolutional feature map 36@14x47

Convolutional feature map 24@31x98

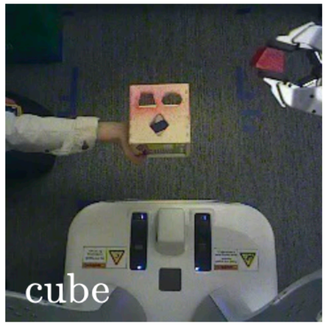
Normalized input planes 3@66x200

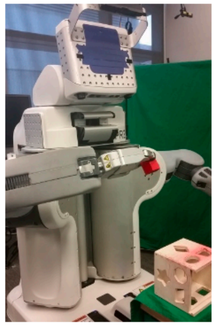
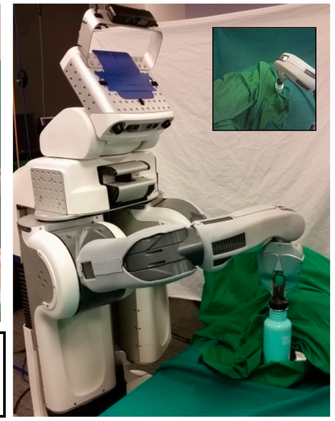
Input planes 3@66x200

Bojarski, Mariusz, et al. "End to end learning for self-driving cars." *arXiv preprint arXiv:1604.07316* (2016).

End-To-End Training Of Deep Visuomotor Policies

YouTube Video



http://sites.google.com/site/visuomotorpolicy

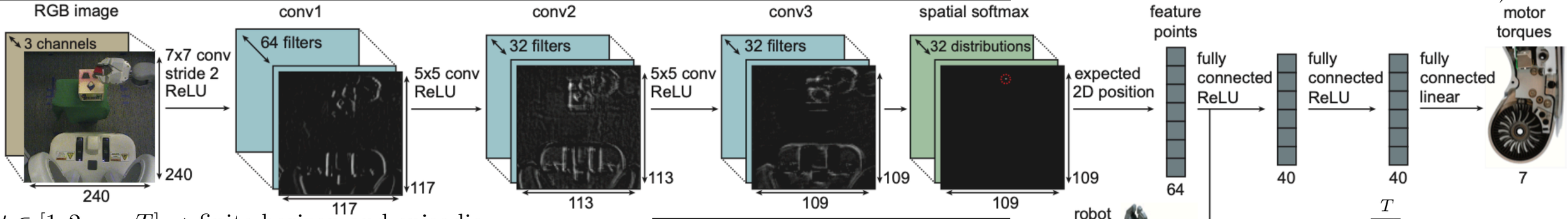
$\pi_\theta(u_t|o_t) \rightarrow$ policy

$u_t \rightarrow$ actions (motor torque commands)

$o_t \rightarrow$ observations (an image from robot's onboard camera)

$x_t \rightarrow$ states (joint angles of the robot, positions of objects in the world, and their time derivatives)

Inserting a block into a shape sorting cube, screwing a cap onto a bottle, fitting the claw of a toy hammer under a nail with various grasps, and placing a coat hanger on a rack with a PR2 robot



$t \in [1, 2, \ldots, T] \rightarrow$ finite horizon and episodic

$p(x_{t+1}|x_t, u_t) \rightarrow$ state dynamics (not assumed to be known)

$$f_{cx} = \sum_{ij} s_{cij} x_{ij} \text{ and } f_{cy} = \sum_{ij} s_{cij} y_{ij}$$

$p(o_t|x_t) \rightarrow$ observations are a stochastic consequence of the states (not assumed to be known)

$\pi_\theta(u_t|x_t) = \int \pi_\theta(u_t|o_t) p(o_t|x_t) do_t \rightarrow$ distribution over actions under the policy conditioned on the state

$\tau = \{x_1, u_1, x_2, u_2, \ldots, x_T, u_T\} \rightarrow$ trajectory

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$$

$\pi_\theta(\tau) = p(x_1) \prod_{t=1}^{T} \pi_\theta(u_t|x_t) p(x_{t+1}|x_t, u_t) \rightarrow$ distribution over trajectories

$\ell(x_t, u_t) \rightarrow$ cost function

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t + f_{ct}, \mathbf{F}_t)$$

$$\min_\theta \mathbb{E}_{\pi_\theta(\tau)}[\underbrace{\sum_{t=1}^{T} \ell(x_t, u_t)}_{=:\ell(\tau)}]$$

Guided Policy Search

$$\min_{\theta, p} \quad \mathbb{E}_{p(\tau)}[\ell(\tau)]$$

$$\text{s.t.} \quad p(u_t|x_t) = \pi_\theta(u_t|x_t)$$

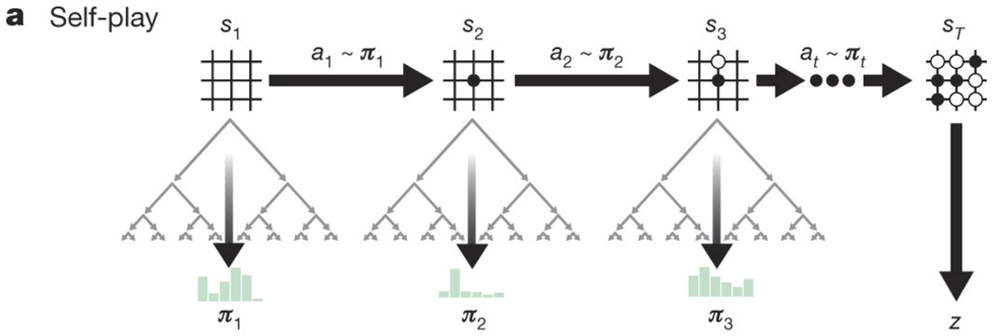
$p(\tau) \rightarrow$ guiding distribution

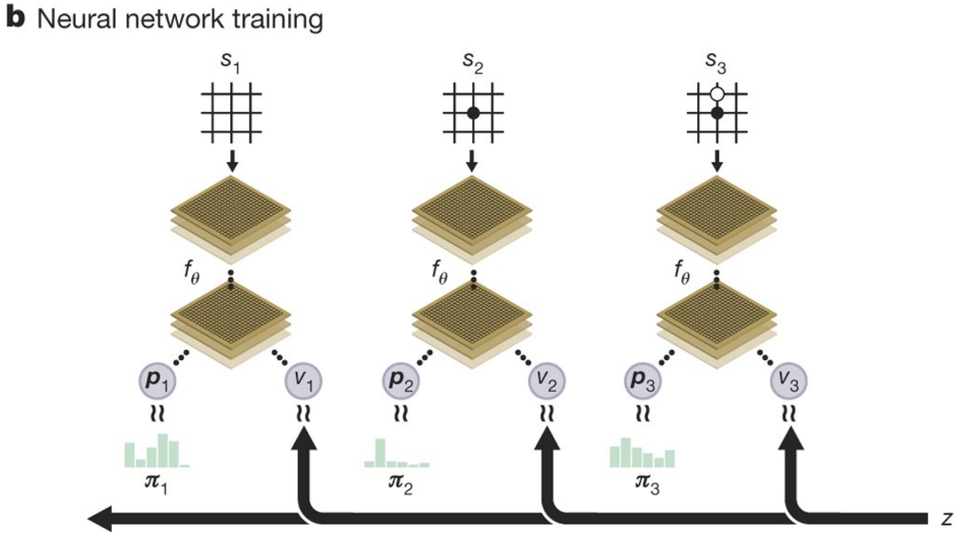
Alternating Direction Method of Multipliers

Levine, Sergey, et al. "End-to-end training of deep visuomotor policies." *The Journal of Machine Learning Research* 17.1 (2016): 1334-1373.

Mastering the game of Go without human knowledge

YouTube Video

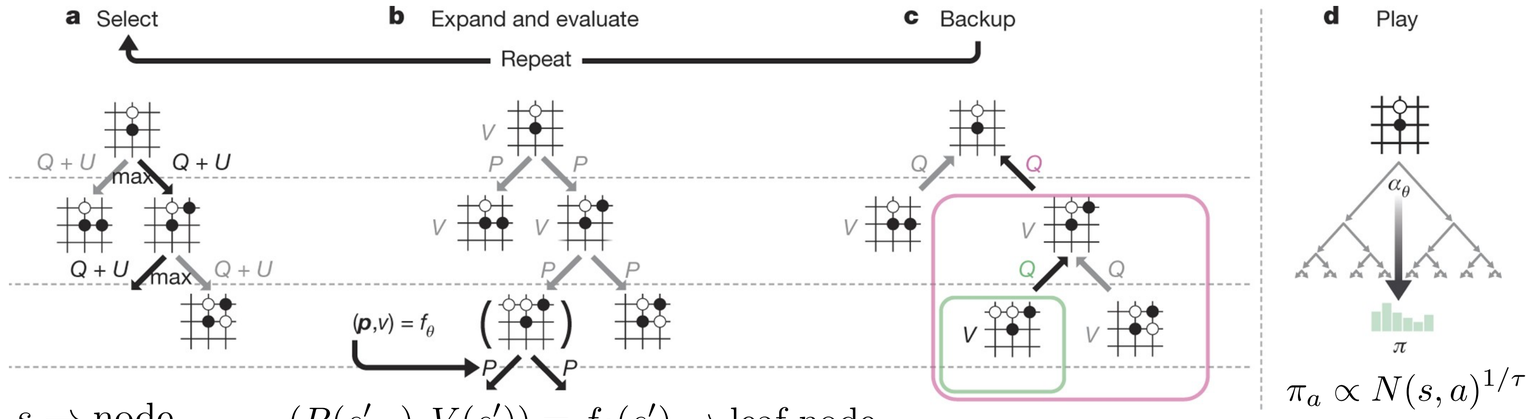




$(\mathbf{p}, v) = f_\theta(s) \rightarrow$ policy & value networks

$$\ell = \underbrace{(z-v)^2}_{\text{MSE}} - \underbrace{\pi^T \log \mathbf{p}}_{\text{cross-entropy}} + \underbrace{c\|\theta\|^2}_{\text{L2 regularization}} \rightarrow \text{loss}$$

$z \rightarrow$ winning or losing



$s \rightarrow$ node $\quad (P(s',.), V(s')) = f_\theta(s') \rightarrow$ leaf node

$(s,a) \rightarrow$ edge

$P(s,a) \rightarrow$ prior prob.

$N(s,a) \rightarrow$ visit count

$Q(s,a) \rightarrow$ action value

$Q(s,a) + U(s,a) \rightarrow$ upper confidence bound

$$U(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|(s,a)\rightarrow s'} V(s')$$

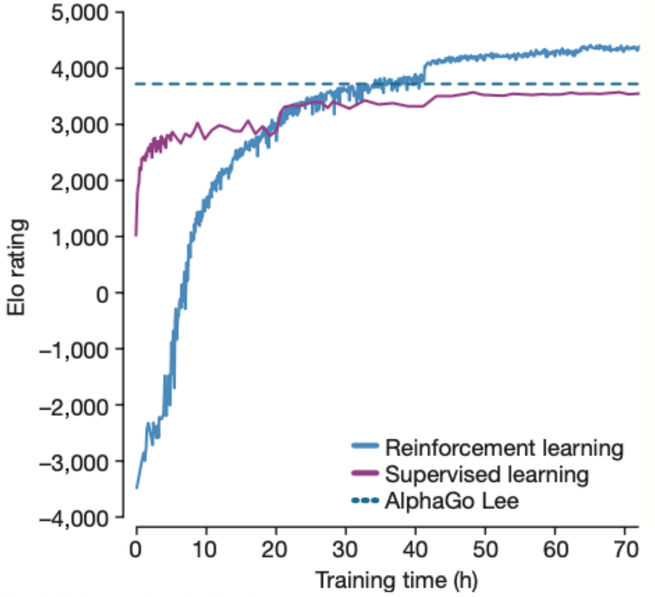
Evaluation

Bayesian Logistic Regression

$$p(a \text{ defeats } b) = \frac{1}{1+\exp(c_{\text{elo}}(e(b)-e(a))}$$

$e(a) \rightarrow$ Elo rating of player a

$c_{\text{elo}} = 1/400$



Silver, David, et al. "Mastering the game of go without human knowledge." *nature* 550.7676 (2017): 354-359.

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

YouTube Video

$(\mathbf{p}, v) = f_\theta(s)$

$s \rightarrow$ board position

$\mathbf{p} \rightarrow$ move probability

$a \rightarrow$ action

$p_a = Pr(a|s)$

$v \rightarrow$ expected outcome z of the game from position s

$v \approx \mathbb{E}[z|s]$

$z = +1$ (win), 0 (draw), -1 (lose)

Monte-Carlo Tree Search (MCTS)

$s_{\text{root}} \rightarrow$ root state

$\pi_a = Pr(a|s_{\text{root}})$

$\ell = (z - v)^2 - \pi^T \log \mathbf{p} + c\|\theta\|^2 \rightarrow$ loss

Search

$\{ \underbrace{N(s,a)}_{\text{visit count}}, \underbrace{W(s,a)}_{\text{total action-value}}, \underbrace{Q(s,a)}_{\text{mean action-value}}, \underbrace{P(s,a)}_{\text{prior prob.}} \}$

set of statistics stored at each state-action pair (s, a)

$s_0 \rightarrow$ root node of the tree search

$s_L \rightarrow$ leaf node at time-step L

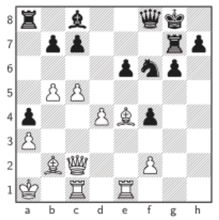
$a_t = \arg\max_a (Q(s_t, a) + U(s_t, a))$, for $t < L$

$U(s,a) = C(s)P(s,a)\sqrt{N(s)}/(1 + N(s,a))$

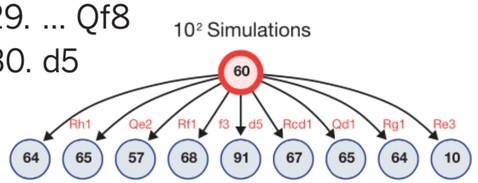
parent visit count

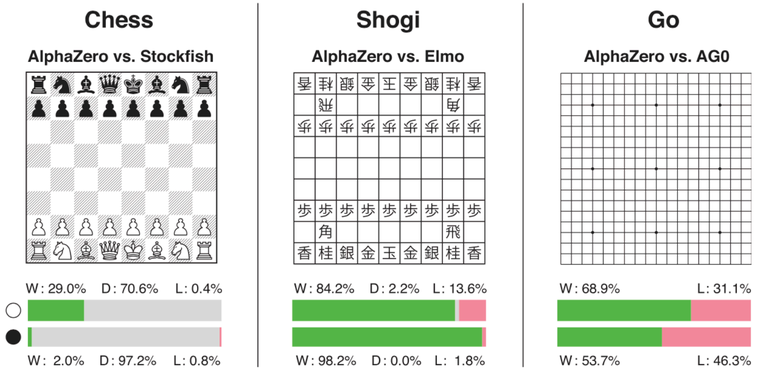
$C(s) = \log\left((1 + N(s) + c_{\text{base}})/c_{\text{base}}\right) + c_{\text{init}}$

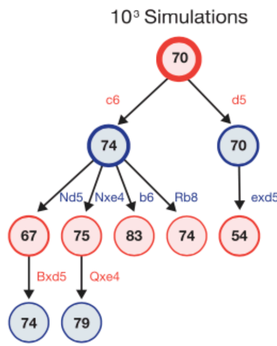
exploration rate

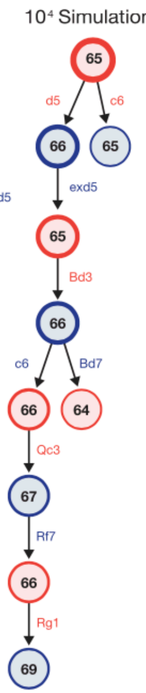


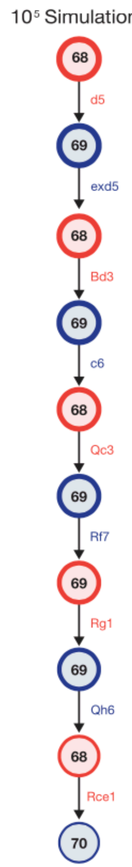
29. ... Qf8
30. d5

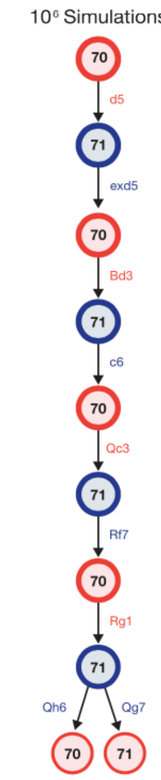


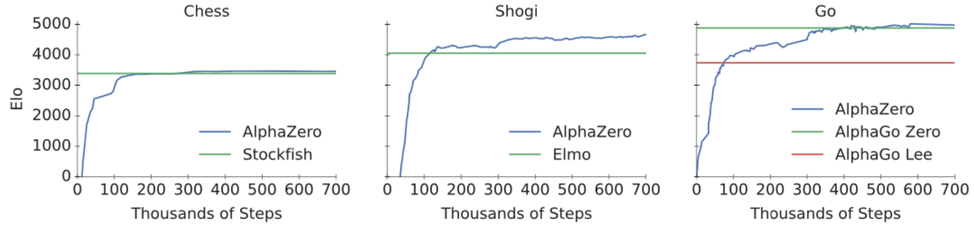












$(\mathbf{p}, v) = f_\theta(s_L)$

$\{N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = p_a\}$

$$\left.\begin{aligned} N(s_t, a_t) &= N(s_t, a_t) + 1 \\ W(s_t, a_t) &= W(s_t, a_t) + v \\ Q(s_t, a_t) &= W(s_t, a_t)/N(s_t, a_t) \end{aligned}\right\} \text{for } t \leq L$$

Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362.6419 (2018): 1140-1144.

Proximal Policy Optimization Algorithms

YouTube Video

Policy Gradient Methods

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \rightarrow \text{policy gradient estimator}$$

- $\hat{\mathbb{E}}_t$: empirical average over a finite batch of samples
- π_θ: policy
- $\hat{A}_t$: estimator of the advantage function at timestep t

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t]$$

Trust Region Methods

$$\max_\theta \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \rightarrow \text{linear approximation}$$

conjugate gradient algorithm

$$\text{s.t.} \quad \hat{\mathbb{E}}_t[\text{KL}(\pi_{\theta_{\text{old}}}(.|s_t)\|\pi_\theta(.|s_t))] \leq \delta \rightarrow \text{quadratic approximation}$$

$$\max_\theta \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t - \beta\text{KL}(\pi_{\theta_{\text{old}}}(.|s_t)\|\pi_\theta(.|s_t))\right]$$

Clipped Surrogate Objective

$$r_t(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \implies r_t(\theta_{\text{old}}) = 1$$

$r_t(\theta)$: prob. ratio

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t]$$

CPI: Conservative Policy Iteration

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(\underbrace{r_t(\theta)\hat{A}_t}_{L^{CPI}}, \underbrace{\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)}_{\text{removes the incentive for moving } r_t \text{ outside of the interval } [1-\epsilon, 1+\epsilon]}\hat{A}_t)\Big]$$

min: pessimistic lower bound

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\big[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\big]$$

$L_t^{VF}(\theta)$: $(V_\theta(s_t) - V_t^{\text{targ}})^2$; $S[\pi_\theta](s_t)$: entropy bonus

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$$

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

same as above if $\lambda = 1$

Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1,2,... do
    for actor=1,2,...,N do
        Run policy π_θold in environment for T timesteps
        Compute advantage estimates Â_1,...,Â_T
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
    θ_old ← θ
end for
```

Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Boulder

YouTube Playlist

Goal: Train a model on a variety of learning tasks such that it can solve new learning tasks using only a small number of training examples

Tasks as training examples!

$f \rightarrow$ model

$x \longmapsto a$ outputs

observations

episode length

transition distribution

$$\mathcal{T} = \{\mathcal{L}(x_1, a_1, \ldots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$$

task

loss function

distribution over initial observations

$H = 1$ in $i.i.d$ supervised problems

$p(\mathcal{T}) \rightarrow$ distribution over tasks

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

Optimize the model parameters such that one or a small number of gradient steps on a new task will produce maximally effective behavior on that task

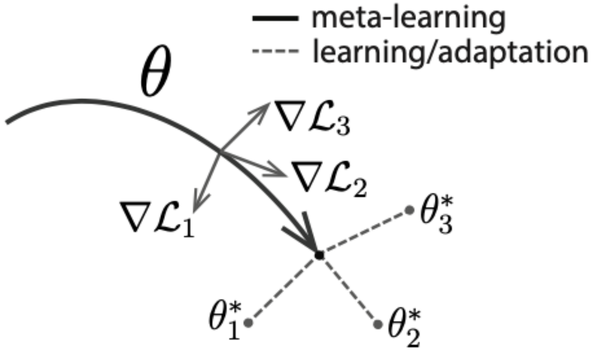
$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \|f_\phi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)}\|_2^2, \rightarrow \text{Regression}$$

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) \rightarrow \text{Classification}$$

$$+ (1 - \mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)}))$$

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**



$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = -\mathbb{E}_{\mathbf{x}_t, \mathbf{a}_t \sim f_\phi, q_{\mathcal{T}_i}} \left[\sum_{t=1}^{H} R_i(\mathbf{x}_t, \mathbf{a}_t) \right]$$

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
11: **end while**

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, \ldots \mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, \ldots \mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: **end while**

Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." *arXiv preprint arXiv:1703.03400* (2017).

Overcoming catastrophic forgetting in neural networks

Continual learning: Learning multiple tasks sequentially

$\theta \rightarrow$ weight and biases of a neural network

$\theta^*_B \rightarrow$ solution to task B

$\theta^*_A \rightarrow$ solution to task A

Over-parametrization makes it likely that θ^*_B is close to θ^*_A

Elastic Weight Consolidation (EWC)

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \frac{1}{2}\lambda \sum_i F_i(\theta_i - \theta^*_{A,i})^2 \rightarrow \text{loss function}$$

Constraining "important" parameters to stay close to their old values!

$p(\theta|\mathcal{D}) \rightarrow$ posterior

$$\log p(\theta|\mathcal{D}) = \underbrace{\log p(\mathcal{D}|\theta)}_{-\mathcal{L}(\theta)} + \log p(\theta) - \log p(\mathcal{D})$$

$\mathcal{D} = \mathcal{D}_A \cup \mathcal{D}_B$

$\mathcal{D}_A \rightarrow$ data on task A

$\mathcal{D}_B \rightarrow$ data on task B

$$\log p(\theta|\mathcal{D}) = \underbrace{\log p(\mathcal{D}_B|\theta)}_{-\mathcal{L}_B(\theta)} + \underbrace{\log p(\theta|\mathcal{D}_A)}_{\text{interactable}} - \log p(\mathcal{D}_B)$$

Laplace Approximation

$$f(\theta) := \log p(\theta|\mathcal{D}_A) = \underbrace{\log p(\mathcal{D}_A|\theta)}_{-\mathcal{L}_A(\theta)} + \log p(\theta) - \log p(\mathcal{D}_A)$$

$$f(\theta) \approx \underbrace{f(\theta^*_A)}_{\text{const.}} + \underbrace{\nabla_\theta f(\theta^*_A)}_{=0}(\theta - \theta^*_A) + \frac{1}{2}(\theta - \theta^*_A)^T \underbrace{\nabla^2_\theta f(\theta^*_A)}_{\text{Hessian}}(\theta - \theta^*_A)$$

$$\nabla^2_\theta f(\theta^*_A) = -\underbrace{\nabla^2_\theta \mathcal{L}_A(\theta^*_A)}_{\lambda F} + \underbrace{\nabla^2_\theta \log p(\theta)}_{\text{ignore}}$$

$F \rightarrow$ empirical Fisher information matrix

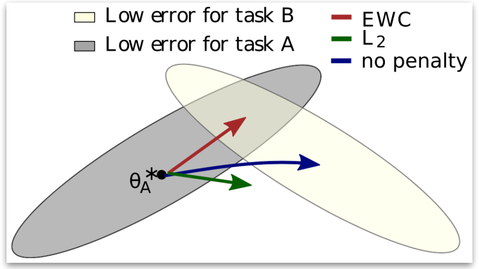
$p(\theta|\mathcal{D}_A) \approx \mathcal{N}(\theta^*_A, \text{diag}(\lambda F)^{-1})$

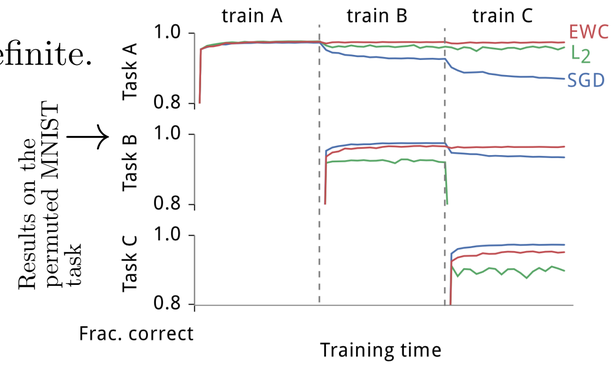
Three key properties of F:

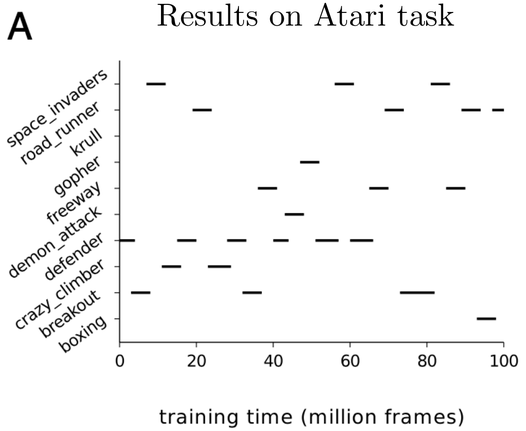
– It is equivalent to the second derivative of the loss near a minimum.

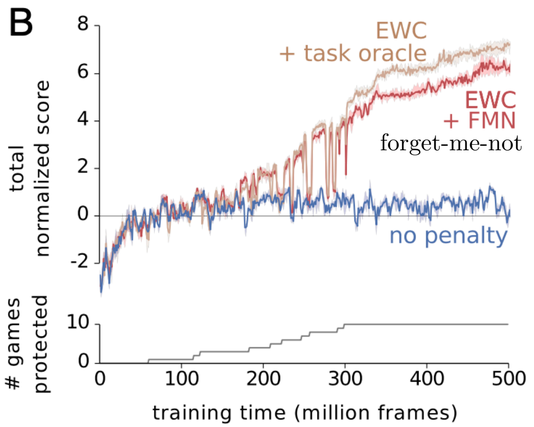
– It can be computed from first-order derivatives alone and is thus easy to calculate even for large models.

– It is guaranteed to be positive semidefinite.









Kirkpatrick, James, et al. "Overcoming catastrophic forgetting in neural networks." *Proceedings of the national academy of sciences* 114.13 (2017): 3521-3526.

Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection

Boulder

YouTube Video

https://youtu.be/cXaic_k80uM

Visual servoing: vision-based robot control

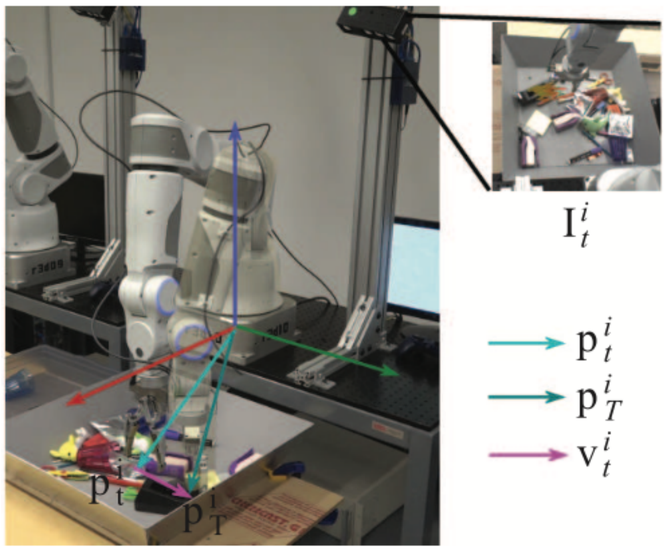
$g \rightarrow$ grasp success prediction network

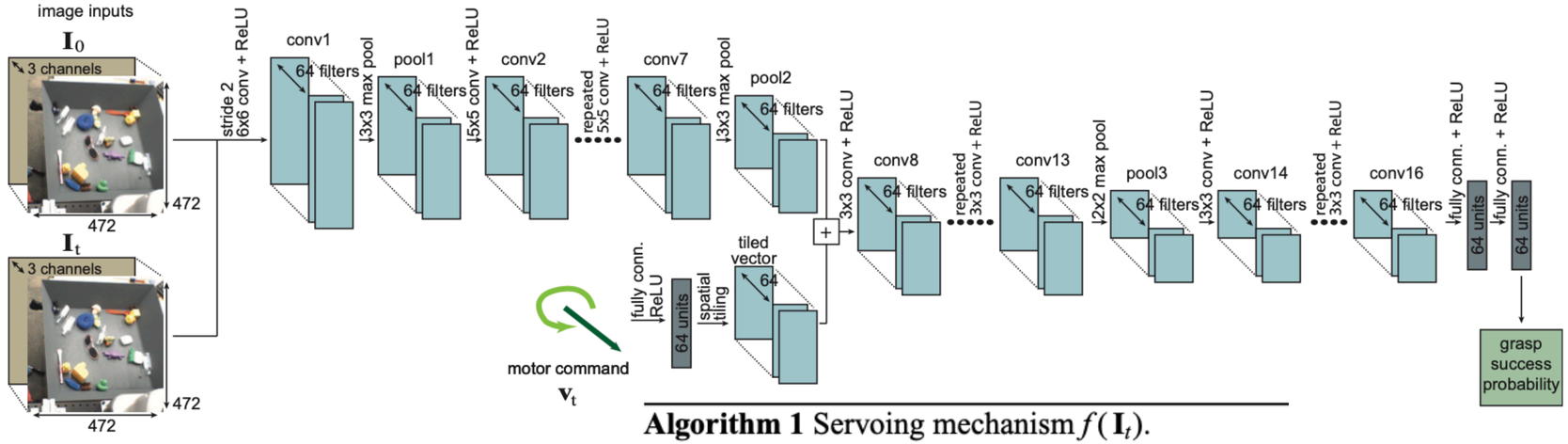
$I_t \rightarrow$ visual input

$v_t \rightarrow$ task-space motion command

$g(I_t, v_t) \rightarrow$ predicted probability that executing the command v_t will produce a successful grasp

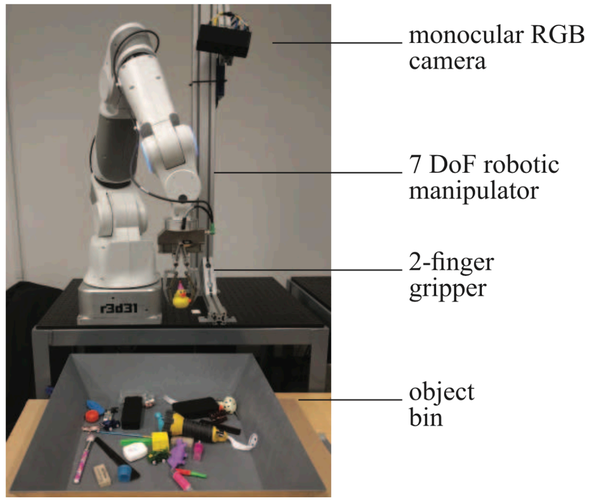
$f(I_t) \rightarrow$ servoing mechanism





$(I_t^i, \underbrace{P_T^i - p_t^i}_{v_t^i}, \ell_i) \rightarrow T$ training samples

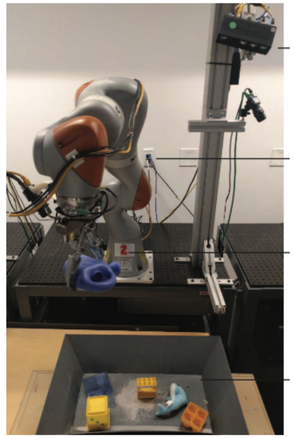
label



Algorithm 1 Servoing mechanism $f(\mathbf{I}_t)$.

1: Given current image $\mathbf{I}_t$ and network g.
2: Infer $\mathbf{v}_t^\star$ using g and CEM.
3: Evaluate $p = g(\mathbf{I}_t, \emptyset) / g(\mathbf{I}_t, \mathbf{v}_t^\star)$.
4: **if** $p > 0.9$ **then**
5: Output $\emptyset$, close gripper.
6: **else if** $p \leq 0.5$ **then**
7: Modify $\mathbf{v}_t^\star$ to raise gripper height and execute $\mathbf{v}_t^\star$.
8: **else**
9: Execute $\mathbf{v}_t^\star$.
10: **end if**

No Motion



Cross-Entropy Method (CEM)

A simple derivative-free optimization algorithm

1) Sample N=64 values from $\mathcal{N}(0, I)$
2) Choose the best M = 6 and fit a Gaussian to these M samples
3) Sample N=64 value from this Gaussian
4) Repeat steps 2 & 3 three times
5) Return the best v_t^*

Levine, Sergey, et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." *The International Journal of Robotics Research* 37.4-5 (2018): 421-436.

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Boulder

YouTube Video

Model-free Deep Reinforcement Learning
– very high sample complexity
– brittle convergence properties (meticulous hyper-parameter tuning)

Continuous state and action spaces

$(\mathcal{S}, \mathcal{A}, p, r) \rightarrow$ infinite-horizon Markov decision process (MDP)

$\mathcal{S} \rightarrow$ state space (continuous)

$\mathcal{A} \rightarrow$ action space (continuous)

$p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$

└ unknown state transition probability
probability density of the next state $s_{t+1} \in \mathcal{S}$
given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$

$r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{\min}, r_{\max}]$

└ reward emitted on each transition

$\rho_\pi(s_t) \rightarrow$ state marginal of the trajectory distribution induced by a policy $\pi(a_t|s_t)$

$\rho_\pi(s_t, a_t) \rightarrow$ state-action marginal

Maximum Entropy Reinforcement Learning

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t)) \right]$$

$Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

└ soft Q-value

$\mathcal{T}^\pi \rightarrow$ Bellman backup operator

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V(\mathbf{s}_{t+1}) \right]$$

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi} \left[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t) \right]$$

└ soft state value function

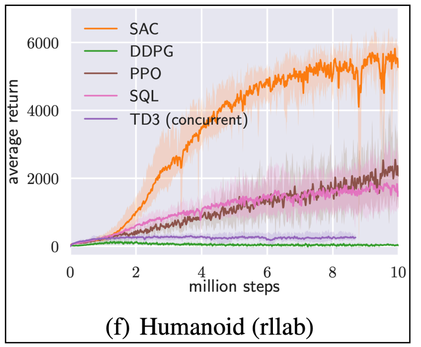
Lemma (Soft Policy Evaluation)

$$Q^{k+1} = \mathcal{T}^\pi Q^k \implies Q^k \rightarrow \text{soft } Q\text{-value as } k \rightarrow \infty$$

Lemma (Soft Policy Improvement)

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}} \left(\pi'(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q^{\pi_{\text{old}}}(\mathbf{s}_t, \,\cdot\,)\right)}{Z^{\pi_{\text{old}}}(\mathbf{s}_t)} \right)$$

$$\implies Q^{\pi_{\text{new}}}(s_t, a_t) \geq Q^{\pi_{\text{old}}}(s_t, a_t), \forall (s_t, a_t) \in \mathcal{S} \times \mathcal{A}$$



(f) Humanoid (rllab)

Theorem (Soft Policy Iteration)

Repeated application of soft policy evaluation and soft policy improvement converges to a policy π^* such that $Q^{\pi^*}(s_t, a_t) \geq Q^\pi(s_t, a_t), \forall \pi \in \Pi$.

Soft Actor-Critic

$V_\psi(s_t) \rightarrow$ parametrized state value function

$Q_\theta(s_t, a_t) \rightarrow$ parametrized soft Q-function

$\pi_\phi(a_t|s_t) \rightarrow$ tractable policy

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right] \right)^2 \right]$$

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right)^2 \right]$$

$$\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right]$$

$\bar{\psi}$ can be an exponentially moving average of the value network weights

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_\phi(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q_\theta(\mathbf{s}_t, \,\cdot\,)\right)}{Z_\theta(\mathbf{s}_t)} \right) \right]$$

$\mathbf{a}_t = f_\phi(\epsilon_t; \mathbf{s}_t) \rightarrow$ reparameterization trick

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\log \pi_\phi(f_\phi(\epsilon_t; \mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t; \mathbf{s}_t)) \right]$$

defined implicitly in terms of f_ϕ

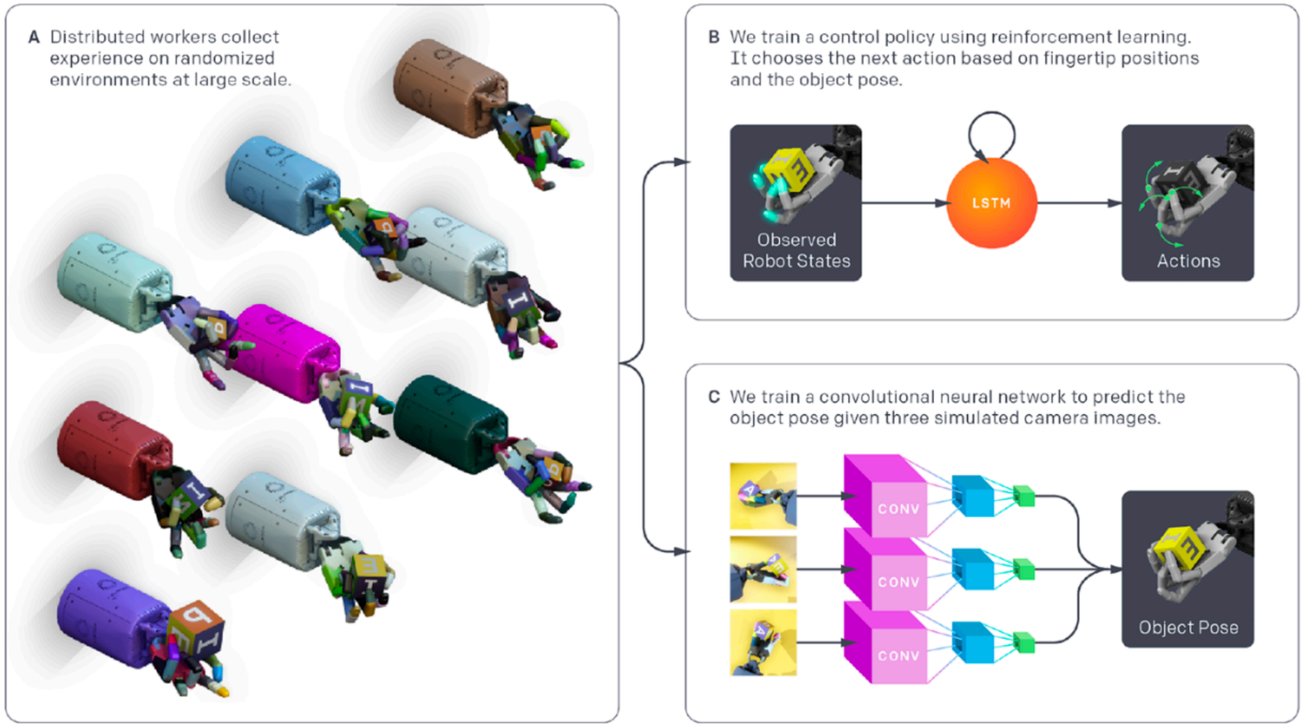
can be differentiated through!

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$.
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**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." *International Conference on Machine Learning*. PMLR, 2018.

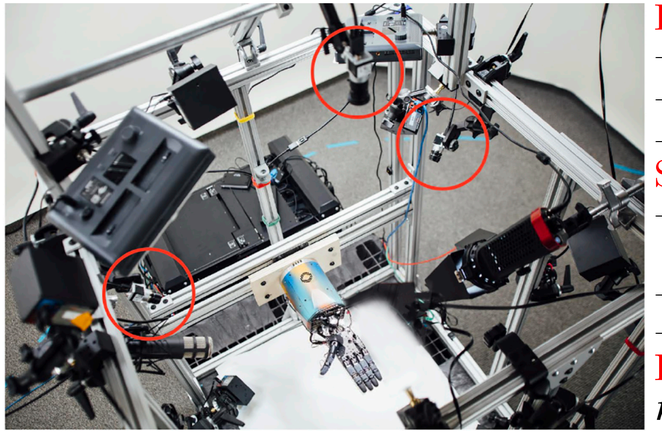
Learning Dexterous In-Hand Manipulation



Andrychowicz, OpenAI: Marcin, et al. "Learning dexterous in-hand manipulation." *The International Journal of Robotics Research* 39.1 (2020): 3-20.

Uncut and real-time video footage of a trial in which the robot hand successfully performed 50 consecutive rotations: https://youtu.be/DKe8FumoD4E

Generalized Advantage Estimator (GAE) & Proximal Policy Optimization (PPO)



Hardware
- Shadow Dexterous Hand
- PhaseSpace tracking
- RGB cameras

Simulation
- MuJoCo physics engine
  http://mujoco.org/book/
- Unity
- OpenAI Gym

Rewards

$r_t = d_t - d_{t+1}$

d_t and d_{t+1} are the rotation angles between the desired and current object orientations before and after the transition, respectively.

Reward of 5 whenever a goal is achieved and a reward of -20 (penalty) whenever the object is dropped.

Reality Gap

We face a dilemma: we cannot train on the physical robot because deep RL algorithms require millions of samples; conversely, training only in simulation results in policies that do no transfer well due to the gap between the simulated and real environments.

Domain Randomization

Distribution over many simulations: randomizing most aspects of the simulated environment (e.g., Observation noise, Physics randomizations, Unmodeled effects, and Visual appearance randomizations)

Physical task	Mean	Median	Individual trials (sorted)									
Block (vision)	15.2±14.3	11.5	46	28	26	15	13	10	8	3	2	1

Boulder

Questions?

YouTube Playlist