

# Policy Gradient Methods

<http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html>

Acknowledgement: this slides is based on Prof. Shipra Agrawal's lecture notes

# Outline

- 1 Policy gradient methods
  - Finite horizon MDP
  - Infinite horizon case
- 2 Actor-critic methods
- 3 TRPO and PPO
- 4 MCTS and AlphaGo Zero

# Policy gradient methods

- In  $Q$ -learning function approximation was used to approximate  $Q$ -function, and policy was a greedy policy based on estimated  $Q$ -function. In policy gradient methods, we approximate a stochastic policy directly using a parametric function approximator.
- More formally, given an MDP  $(S, A, s_1, R, P)$ , let  $\pi_\theta : S \rightarrow \Delta^A$  denote a randomized policy parameterized by parameter vector  $\theta \in \mathbb{R}^d$ . For a scalable formulation, we want  $d \ll |S|$ .
- For example, the policy  $\pi_\theta$  might be represented by a neural network whose input is a representation of the state, whose output is action selection probabilities, and whose weights form the policy parameters  $\theta$ . (The architecture for such a [deep] neural network is similar to that for a multi-label classifier, with input being a state, and labels being different actions. The network should be trained to predict the probability of different actions given an input state).

# Policy gradient methods

- For simplicity, assume that  $\pi_\theta$  is differentiable with respect to  $\theta$ , i.e.  $\frac{\partial \pi_\theta(s,a)}{\partial \theta}$  exists. This is true for example, if a neural network with differentiable activation functions is used to define  $\pi_\theta$ . Let  $\rho(\pi_\theta)$  denote the gain of policy  $\pi_\theta$ . This may be defined as long term average reward, long term discounted reward or total reward in an episode or finite horizon. Therefore, solving for optimal policy reduces to the problem of solving

$$\max_{\theta} \rho(\pi_\theta)$$

- In order to use stochastic gradient descent algorithm for finding a stationary point of the above problem, we need to compute (an unbiased) estimate of gradient of  $\rho(\pi_\theta)$  with respect to  $\theta$ .

# Finite horizon MDP

- Here performance measure to optimize is total expected reward over a finite horizon  $H$ .

$$\rho(\pi) = \mathbb{E} \left[ \sum_{t=1}^H \gamma^{t-1} r_t | \pi, s_1 \right]$$

- Let  $\pi(s, a)$  denote the probability of action  $a$  in state  $s$  for randomized policy  $\pi$ . Let  $D^\pi(\tau)$  denote the probability distribution of a trajectory (state-action sequence)  $\tau = (s_1, a_1, s_2, \dots, a_{H-1}, s_H)$  of states on starting from state  $s_1$  and following policy  $\pi$ . That is,

$$D^\pi(\tau) := \prod_{i=1}^{H-1} \pi(s_i, a_i) P(s_i, a_i, s_{i+1})$$

# Finite horizon MDP

## Theorem 2

For finite horizon MDP  $(S, A, s_1, P, R, H)$ , let  $R(\tau)$  be the total reward for an sample trajectory  $\tau$ , on following  $\pi_\theta$  for  $H$  steps, starting from state  $s_1$ . Then,

$$\nabla_{\theta} \rho(\pi_{\theta}) = \mathbb{E}_{\tau} [R(\tau) \nabla_{\theta} \log (D^{\pi_{\theta}}(\tau))] = \mathbb{E}_{\tau} \left[ R(\tau) \sum_{t=1}^{H-1} \nabla_{\theta} \log (\pi_{\theta}(s_t, a_t)) \right]$$

**Proof.** Let  $R(\tau)$  be expected total reward for an entire sample trajectory  $\tau$ , on following  $\pi_\theta$  for  $H$  steps, starting from states  $s_1$ . That is, given a sample trajectory  $\tau = (s_1, a_1, s_2, \dots, a_{H-1}, s_H)$  from distribution  $D^{\pi_\theta}$ ,

$$R(\tau) := \sum_{t=1}^{H-1} \gamma^{t-1} R(s_t, a_t)$$

## Proof of Theorem 2

Then,

$$\rho(\pi_\theta) = \mathbb{E}_{\tau \sim D^{\pi_\theta}}[R(\tau)]$$

Now, (the calculations below implicitly assume finite state and action space, so that the distribution  $D(\tau)$  has a finite support)

$$\begin{aligned} \frac{\partial \rho(\pi_\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim D^{\pi_\theta}}[R(\tau)] \\ &= \frac{\partial}{\partial \theta} \sum_{\tau: D^{\pi_\theta}(\tau) > 0} D^{\pi_\theta}(\tau) R(\tau) \\ &= \sum_{\tau: D^{\pi_\theta}(\tau) > 0} D^{\pi_\theta}(\tau) \frac{\partial}{\partial \theta} \log(D^{\pi_\theta}(\tau)) R(\tau) \\ &= \mathbb{E}_{\tau \sim D^{\pi_\theta}} \left[ \frac{\partial}{\partial \theta} \log(D^{\pi_\theta}(\tau)) R(\tau) \right] \end{aligned}$$

## Proof of Theorem 2

Further, for a given sample trajectory  $\tau^i$ .

$$\begin{aligned}\nabla_{\theta} \log (D^{\pi_{\theta}}(\tau^i)) &= \sum_{t=1}^{H-1} \nabla_{\theta} \log (\pi_{\theta}(s_t^i, a_t^i)) + \nabla_{\theta} \log P(s_t^i, a_t^i, s_{t+1}^i) \\ &= \sum_{t=1}^{H-1} \nabla_{\theta} \log (\pi_{\theta}(s_t^i, a_t^i))\end{aligned}$$



# Finite horizon MDP

- The gradient representation given by above theorem is extremely useful, as given a sample trajectory this can be computed only using the policy parameter, and does not require knowledge of the transition model  $P(\cdot, \cdot, \cdot)$ ! This does seem to require knowledge of reward model, but that can be handled by replacing  $R(\tau^i)$  by  $\hat{R}(\tau^i) = r_1 + \gamma r_2 + \dots, \gamma^{H-2} r_{H-1}$ , the total of sample rewards observed in this trajectory.
- Since, given a trajectory  $\tau$ , the quantity  $D^{\pi_\theta}(\tau)$  is determined, and  $\mathbb{E}[\hat{R}(\tau)|\tau] = R(\tau)$

$$\begin{aligned}\nabla_{\theta} \rho(\pi_{\theta}) &= \mathbb{E}_{\tau} [R(\tau) \nabla_{\theta} \log(D^{\pi_{\theta}}(\tau))] \\ &= \mathbb{E}_{\tau} [\hat{R}(\tau) \nabla_{\theta} \log(D^{\pi_{\theta}}(\tau))] \\ &= \mathbb{E}_{\tau} \left[ \hat{R}(\tau) \sum_{t=1}^{H-1} \nabla_{\theta} \log(\pi_{\theta}(s_t, a_t)) \right]\end{aligned}$$

# Unbiased estimator of gradient from samples

- From above, given sample trajectories  $\tau^i, i = 1, \dots, m$ , an unbiased estimator for gradient  $\nabla_{\theta} \rho(\pi_{\theta})$  is given as:

$$\begin{aligned}\hat{\mathbf{g}} &= \frac{1}{m} \sum_{i=1}^m \hat{R}(\tau^i) \nabla_{\theta} \log(D^{\pi_{\theta}}(\tau^i)) \\ &= \frac{1}{m} \sum_{i=1}^m \hat{R}(\tau^i) \sum_{t=1}^{H-1} \nabla_{\theta} \log(\pi_{\theta}(s_t^i, a_t^i))\end{aligned}$$

# Baseline

- Note that for any constant  $b$  (or  $b$  that is conditionally independent of sampling from  $\pi_\theta$  given  $\theta$ ), we have:

$$\mathbb{E}_\tau \left[ b \frac{\partial}{\partial \theta_j} \log (D^{\pi_\theta}(\tau)) \mid \theta, s_1 \right] = b \int_\tau \frac{\partial}{\partial \theta_j} (D^{\pi_\theta}(\tau)) = b \frac{\partial}{\partial \theta_j} \int_\tau D^{\pi_\theta}(\tau) = 0$$

- Therefore, choosing any 'baseline'  $b$ , following is also an unbiased estimator of the  $\nabla_{\theta} \rho(\pi_\theta)$ :

$$\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^{H-1} (\hat{R}(\tau^i) - b) \nabla_{\theta} \log (\pi_{\theta}(s_t^i, a_t^i))$$

# Baseline

- Or, more generally, one could even use a state and time dependent baseline  $b_t(s_t^i)$  conditionally is independent of sampling from  $\pi_\theta$  given  $s_t^i$ ,  $\theta$ , to get estimator:

$$\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^{H-1} (\hat{R}(\tau^i) - b_t(s_t^i)) \nabla_\theta \log(\pi_\theta(s_t^i, a_t^i)) \quad (1)$$

- Below we show this is unbiased. The expectations below are over trajectories  $(s_1, a_1, \dots, a_{H-1}, s_H)$ , where  $a_t \sim \pi(s_t, \cdot)$ , given  $s_t$ . For any fixed  $\theta$ ,  $t$ , the baseline  $b_t(s_t) | s_t$  needs to be deterministic or independent of  $a_t | s_t$ . For simplicity we assume it is deterministic.

# Baseline

$$\begin{aligned} & \mathbb{E}_\tau \left[ \sum_{t=1}^{H-1} b_t(s_t) \frac{\partial}{\partial \theta_j} \log(\pi_\theta(s_t, a_t)) \mid \theta, s_1 \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^{H-1} \mathbb{E} \left[ b_t(s_t) \frac{\partial}{\partial \theta_j} \log(\pi_\theta(s_t, a_t)) \mid s_t \right] \mid \theta, s_1 \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^{H-1} b_t(s_t) \mathbb{E} \left[ \frac{\partial}{\partial \theta_j} \log(\pi_\theta(s_t, a_t)) \mid s_t \right] \mid \theta, s_1 \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^{H-1} b_t(s_t) \sum_a \pi_\theta(s_t, a) \frac{\partial}{\partial \theta_j} \log(\pi_\theta(s_t, a)) \mid \theta, s_1 \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^{H-1} b_t(s_t) \sum_a \frac{\partial}{\partial \theta_j} \pi_\theta(s_t, a) \mid \theta, s_1 \right] \end{aligned}$$

# Baseline

$$\begin{aligned} &= \mathbb{E} \left[ \sum_{t=1}^{H-1} b_t(s_t) \frac{\partial}{\partial \theta_j} \sum_a \pi_\theta(s_t, a) \mid \theta, s_1 \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^{H-1} b_t(s_t) \frac{\partial}{\partial \theta_j} 1 \mid \theta, s_1 \right] \\ &= 0 \end{aligned}$$

- An example of such state dependent baseline  $b_t(s)$ , given  $s$  and  $\theta$ , is  $V_{H-t}^{\pi_\theta}(s)$ , i.e., the value of policy  $\pi_\theta$ , starting from state  $s$  at time  $t$ . We will see later that such a baseline is useful in reducing the variance of gradient estimates.

# Vanilla policy gradient algorithm

Initialize policy parameter  $\theta$ , and baseline.

In each iteration,

- Execute current policy  $\pi_\theta$  to obtain several sample trajectories  $\tau^i$ ,  $i = 1, \dots, m$ .
- Use these sample trajectories and chosen baseline to compute the gradient estimator  $\hat{\mathbf{g}}$  as in (1)
- Update  $\theta \leftarrow \theta + \alpha \hat{\mathbf{g}}$
- Update baseline as required.

Above is essentially same as the **REINFORCE** algorithm introduced by [Williams, 1988, 1992].

# Long term average reward

$\rho(\pi)$ : long-term average reward of (randomized) policy  $\pi$

$$\rho^\pi = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E} [r_1 + r_2 + \dots + r_T | \pi] = \sum_s d^\pi(s) \sum_a \pi(s, a) R(s, a)$$

where  $d^\pi(s) = \lim_{t \rightarrow \infty} \Pr(s_t = s | s_1, \pi)$ , the stationary distribution for policy  $\pi$ , is assumed to exist and independent of the starting state  $s_1$  for all policies  $\pi$ . The value of a state given a policy  $\pi$  is defined as:

$$\begin{aligned} V^\pi(s) &= \lim_{T \rightarrow \infty} \mathbb{E} \left[ \sum_{t=1}^T (r_t - \rho(\pi)) \mid s_1 = s, a_t = \pi(s_t), t = 1, \dots, T \right] \\ &= \sum_a \pi(s_1, a) Q^\pi(s_1, a) \end{aligned}$$

where,

$$Q^\pi(s, a) := \lim_{T \rightarrow \infty} \mathbb{E} \left[ \sum_{t=1}^T (r_t - \rho(\pi)) \mid s_1 = s, a_1 = a, a_t = \pi(s_t), t = 2, \dots, T \right]$$



# Discounted rewards

$$\rho(\pi, s_1) = \lim_{T \rightarrow \infty} \mathbb{E} \left[ \sum_{t=1}^T \gamma^{t-1} r_t | \pi, s_1 \right] = \sum_s d^\pi(s) \sum_a \pi(s, a) R(s, a)$$

where  $d^\pi(s) = \lim_{T \rightarrow \infty} \sum_{t=1}^T \gamma^{t-1} \Pr(s_t = s | s_1, \pi)$ , and the value of a state given a policy  $\pi$  is defined as:

$$\begin{aligned} V^\pi(s) &= \lim_{T \rightarrow \infty} \mathbb{E} \left[ \sum_{t=1}^T \gamma^{t-1} r_t | s_1 = s, a_t = \pi(s_t), t = 1, \dots, T \right] \\ &= \sum_a \pi(s_1, a) Q^\pi(s_1, a) \end{aligned}$$

where

$$Q^\pi(s, a) := \lim_{T \rightarrow \infty} \mathbb{E} \left[ \sum_{t=1}^T \gamma^{t-1} r_t | s_1 = s, a_1 = a, a_t = \pi(s_t), t = 2, \dots, T \right]$$

# Policy gradient theorem

## Theorem 3. [Policy gradient theorem [Sutton et al., 1999]]

For infinite horizon MDP (average or discounted),

$$\begin{aligned}\nabla_{\theta} \rho(\pi_{\theta}, s_1) &= \sum_s d^{\pi_{\theta}}(s) \sum_a Q^{\pi_{\theta}}(s, a) \nabla_{\theta} \pi_{\theta}(s, a) \\ &= \sum_s d^{\pi_{\theta}}(s) \left( \mathbb{E}_{a \sim \pi(s)} [Q^{\pi_{\theta}}(s, a) \nabla_{\theta} \log(\pi_{\theta}(s, a))] \right)\end{aligned}$$

That is gradient of gain with respect to  $\theta$  can be expressed in terms of gradient of policy function with respect to  $\theta$ .

# Proof of Policy gradient theorem

**Proof.** Average reward formulation.

$$Q^\pi(s, a) = R(s, a) + \sum_{s'} P(s, a, s') V^\pi(s') - \rho(\pi)$$

$$V^\pi(s) = \sum_a \pi(s, a) Q^\pi(s, a)$$

$$\nabla_\theta V^\pi(s) = \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) + \pi(s, a) \nabla_\theta Q^\pi(s, a)$$

$$= \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a)$$

$$+ \sum_a \pi(s, a) \left[ \sum_{s'} P(s, a, s') \nabla_\theta V^\pi(s') - \nabla_\theta \rho(\pi) \right]$$

$$\nabla_\theta \rho(\pi) = \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a)$$

$$+ \sum_a \pi(s, a) \left[ \sum_{s'} P(s, a, s') \nabla_\theta V^\pi(s') \right] - \nabla_\theta V^\pi(s)$$

# Proof of Policy gradient theorem

$$\begin{aligned}\sum_s d^\pi(s) \nabla_{\theta} \rho(\pi) &= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_{\theta} \pi(s, a) \\ &+ \sum_s d^\pi(s) \sum_a \pi(s, a) \left[ \sum_{s'} P(s, a, s') \nabla_{\theta} V^\pi(s') \right] \\ &- \sum_s d^\pi(s) \nabla_{\theta} V^\pi(s)\end{aligned}$$

since  $d^\pi$  is the stationary distribution of  $\pi$

$$\begin{aligned}&= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_{\theta} \pi(s, a) \\ &+ \sum_{s'} d^\pi(s') \nabla_{\theta} V^\pi(s') - \sum_s d^\pi(s) \nabla_{\theta} V^\pi(s) \\ &= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_{\theta} \pi(s, a)\end{aligned}$$

# Proof of Policy gradient theorem

Discounted reward formulation.

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V^\pi(s')$$

By Bellman equations:

$$\begin{aligned} V^\pi(s) &= \sum_a \pi(s, a) Q^\pi(s, a) \\ \nabla_\theta V^\pi(s) &= \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) + \pi(s, a) \nabla_\theta Q^\pi(s, a) \end{aligned}$$

Let  $\Pr(s \rightarrow x, k, \pi)$  is the probability of going from state  $s$  to state  $x$  in  $k$  steps under policy  $\pi$ .

$$\sum_s d^\pi(s) \nabla_\theta V^\pi(s) := \sum_s \left( \sum_{k=0}^{\infty} \gamma^k \Pr(s_1 \rightarrow s, k, \pi) \right) \nabla_\theta V^\pi(s)$$

# Proof of Policy gradient theorem

$$\begin{aligned} &= \sum_s \left( \sum_{k=0}^{\infty} \gamma^k \Pr(s_1 \rightarrow s, k, \pi) \right) \cdot \\ &\quad \left( \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) + \gamma \sum_a \pi(s, a) \sum_{s'} P(s, a, s') \nabla_\theta V^\pi(s') \right) \\ &= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) \\ &\quad + \sum_{s'} \left( \sum_s \sum_{k=0}^{\infty} \gamma^k \Pr(s_1 \rightarrow s, k, \pi) \right) \gamma \sum_a \pi(s, a) P(s, a, s') \nabla_\theta V^\pi(s') \\ &= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) \\ &\quad + \sum_{s'} \left( \sum_{k=0}^{\infty} \gamma^{k+1} \Pr(s_1 \rightarrow s', k+1, \pi) \right) \nabla_\theta V^\pi(s') \end{aligned}$$

# Proof of Policy gradient theorem

$$\begin{aligned} &= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) + \sum_{s'} (d^\pi(s') \\ &\quad - \Pr(s_1 \rightarrow s', 0, \pi)) \nabla_\theta V^\pi(s') \\ &= \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a) + \sum_{s'} d^\pi(s') \nabla_\theta V^\pi(s') - \nabla_\theta V^\pi(s_1) \end{aligned}$$

Moving the terms around

$$\nabla_\theta V^\pi(s_1) = \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a)$$

That is,

$$\nabla_\theta \rho(\pi, s_1) = \sum_s d^\pi(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(s, a)$$

## Remarks on Policy gradient theorem

- The key aspect of the expression for the gradient is that there are no terms of the form  $\frac{\partial d^{\pi_\theta}(s)}{\partial \theta}$ : the effect of policy changes on the distribution of states does not appear. This is convenient for approximating the gradient by sampling. For example, if  $s$  was sampled from the distribution obtained by following  $\pi_\theta$ , then  $\sum_a Q^{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(s, a)$  would be an unbiased estimate of  $\nabla_\theta \rho(\pi_\theta)$ . Of course,  $Q^{\pi_\theta}(s, a)$  is also not normally known and must be estimated (in an unbiased way).



# Estimation using samples

- Suppose we run policy  $\pi$  several times starting from  $s_1$  to observe sample trajectories  $\{\tau^i\}$ . Then, at each time step  $t$  in a trajectory  $\tau$ , set

$$\hat{Q}_t := \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$$

where  $r_{t'}$  is the observed reward at time  $t'$ . Then,  $\hat{Q}_t$  is an unbiased estimate of  $Q(s_t, a_t)$  at time  $t$  (almost, assuming large enough  $T$ ), i.e,

$$\mathbb{E} \left[ \hat{Q}_t | s_t, a_t \right] = Q(s_t, a_t)$$

# Estimation using samples

- Let

$$F_t := \hat{Q}_t \nabla_{\theta} \log \pi_{\theta} (s_t, a_t)$$

- Then,

$$\begin{aligned} \mathbb{E} [F_t | s_t] &= \mathbb{E} \left[ \hat{E} \left[ \hat{Q}_t | s_t, a_t \right] \nabla_{\theta} \log (\pi_{\theta} (s_t, a_t)) | s_t \right] \\ &= \mathbb{E} [Q (s_t, a_t) \nabla_{\theta} \log (\pi_{\theta} (s_t, a_t)) | s_t] \\ &= \sum_a Q (s_t, a) \nabla_{\theta} \pi_{\theta} (s_t, a) \end{aligned}$$

# Estimation using samples

- And,

$$\begin{aligned}\mathbb{E} \left[ \sum_{t=1}^{\infty} \gamma^{t-1} F_t \right] &= \sum_{t=1}^{\infty} \gamma^{t-1} \sum_s \mathbb{E} [F_t | s_t = s] \Pr (s_t = s | s_1) \\ &= \sum_{t=1}^{\infty} \gamma^{t-1} \sum_s \left( \sum_a Q(s, a) \nabla_{\theta} \pi_{\theta}(s, a) \right) \Pr (s_t = s | s_1) \\ &= \sum_s \left( \sum_a Q(s, a) \nabla_{\theta} \pi_{\theta}(s, a) \right) \sum_{t=1}^{\infty} \gamma^{t-1} \Pr (s_t = s | s_1) \\ &= \sum_s \left( \sum_a Q(s, a) \nabla_{\theta} \pi_{\theta}(s, a) \right) d^{\pi}(s) \\ &= \nabla_{\theta} \rho(\pi_{\theta})\end{aligned}$$

where the last step follows from the policy gradient theorem.

# Estimation using samples

- Therefore, following is an unbiased (almost, for large  $T$ ) estimate of gradient of  $\nabla_{\theta} \rho(\pi, s_1)$  of policy  $\pi$  at  $\hat{\theta}$ , starting in state  $s_1$ :

$$\hat{\mathbf{g}} = \sum_{t=1}^T \gamma^{t-1} F_t = \sum_{t=1}^T \gamma^{t-1} \hat{Q}_t \nabla_{\theta} \log(\pi_{\theta}(s_t, a_t))$$

- From  $N$  sample trajectories  $\{\tau^i, i = 1, \dots, N\}$  we can develop sample average estimate

$$\hat{\mathbf{g}} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} F_t^i = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} \hat{Q}_t^i \nabla_{\theta} \log(\pi_{\theta}(s_t^i, a_t^i))$$

# Baseline

- We can obtain another unbiased estimate by replacing  $F_t$  by

$$F'_t = \left( \hat{Q}_t - b_t(s_t) \right) \nabla_{\theta} \log(\pi_{\theta}(s_t, a_t))$$

for an arbitrary baseline function  $b_t(s)$ . Then,

$$\begin{aligned} \mathbb{E} \left[ \sum_t \gamma^{t-1} F'_t \right] &= \mathbb{E} \left[ \sum_t \gamma^{t-1} F_t \right] - \mathbb{E} \left[ \sum_t \gamma^{t-1} b_t(s_t) \nabla_{\theta} \log(\pi_{\theta}(s_t, a_t)) \right] \\ &= \mathbb{E} \left[ \sum_t \gamma^{t-1} F_t \right] \end{aligned}$$

# Baseline

- The last step follows because:

$$\mathbb{E} [b_t(s_t) \nabla_{\theta} \log(\pi_{\theta}(s_t, a_t)) | s_t] = b_t(s_t) \sum_a \nabla_{\theta} \pi_{\theta}(s_t, a) = b_t(s_t) \nabla_{\theta}(1) = 0$$

- That is, following is also an unbiased gradient estimate

$$\hat{\mathbf{g}} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} (\hat{Q}_t^i - b_t(s_t^i)) \nabla_{\theta} \log(\pi(s_t^i, a_t^i)) \quad (2)$$

- The difference  $\hat{Q}_t - b_t(s_t)$  is also referred to as **Advantage**. This terminology appears in more recent algorithms like 'Asynchronous Advantage Actor Critic Algorithm (A3C)' [Mnih et al., 2016] and 'Generalized Advantage Estimation (GAE)' [Schulman et al., 2015].

# Vanilla Policy Gradient Algorithm

Initialize policy parameter  $\theta$ , and baseline function  $b_t(s), \forall s$ .

In each iteration,

- Execute current policy  $\pi_\theta$  to obtain several sample trajectories  $\tau^i$ ,  $i = 1, \dots, m$ .
- For any given sample trajectory  $i$ , use observed rewards  $r_1, r_2, \dots$ , to compute  $\hat{Q}_t^i := \frac{1}{\pi(s_t^i, a_t^i)} \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$ .
- Use  $\hat{Q}_t^i$  and baseline function  $b_t(s)$  to compute the gradient estimator  $\hat{\mathbf{g}}$  as in (2).
- Update  $\theta \leftarrow \theta + \alpha \hat{\mathbf{g}}$ .
- Re-optimize baseline.

# Softmax policies

- Consider policy set parameterized by  $\theta \in \mathbb{R}^d$  such that given  $s \in S$ , probability of picking action  $a \in A$  is given by:

$$\pi_{\theta}(s, a) = \frac{e^{\theta^{\top} \phi_{sa}}}{\sum_{a' \in A} e^{\theta^{\top} \phi_{sa'}}$$

where each  $\phi_{sa}$  is an  $d$ -dimensional feature vector characterizing state-action pair  $s, a$ . This is a popular form of policy space called softmax policies. Here,

$$\nabla_{\theta} \log(\pi_{\theta}(s, a)) = \phi_{sa} - \left( \sum_{a' \in A} \phi_{sa'} \pi_{\theta}(s, a') \right) = \phi_{sa} - \mathbb{E}_{a' \sim \pi(s)} [\phi_{sa'}]$$



# Gaussian policy for continuous action spaces

- In continuous action spaces, it is natural to use Gaussian policies. Given state  $s$ , the probability of action  $a$  is given as:

$$\pi_{\theta}(s, a) = \mathcal{N}(\phi(s)^T \theta, \sigma^2)$$

for some constant  $\sigma$ . Here  $\phi(s)$  is a feature representation of  $s$ . Then,

$$\nabla_{\theta} \log(\pi_{\theta}(s, a)) = \nabla_{\theta} \frac{-(a - \theta^T \phi(s))^2}{2\sigma^2} = \frac{(\theta^T \phi(s) - a)}{\sigma^2} \phi(s)$$

# Outline

- 1 Policy gradient methods
  - Finite horizon MDP
  - Infinite horizon case
- 2 Actor-critic methods
- 3 TRPO and PPO
- 4 MCTS and AlphaGo Zero

# Actor-critic methods

The methods we have seen so far can be divided into two categories [Konda and Tsitsiklis, 2003]

- Actor-only methods (vanilla policy gradient) work with a parameterized family of policies. The gradient of the performance, with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in a direction of improvement. A possible drawback of such methods is that the gradient estimators may have a large variance. Furthermore, as the policy changes, a new gradient is estimated independently of past estimates (by sampling trajectories). Hence, there is no "learning", in the sense of accumulation and consolidation of older information.

# Actor-critic methods

- Critic-only methods (e.g.,  $Q$ -learning, TD-learning) rely exclusively on value function approximation and aim at learning an approximate solution to the Bellman equation, which will then hopefully prescribe a near-optimal policy. Such methods are indirect in the sense that they do not try to optimize directly over a policy space. A method of this type may succeed in constructing a "good" approximation of the value function, yet lack reliable guarantees in terms of near-optimality of the resulting policy.

# Actor-critic methods

Actor-critic methods aim at combining the strong points of actor-only and critic-only methods, by incorporating value function approximation in the policy gradient methods. We already saw the potential of using value function approximation for picking baseline for variance reduction. Another more obvious place to incorporate  $Q$ -value approximation is for approximating  $Q$ -function in the policy gradient expression. Recall, by policy gradient theorem:

$$\nabla_{\theta} \rho(\pi_{\theta}) = \sum_s d^{\pi_{\theta}}(s) \mathbb{E}_{a \sim \pi(s)} [(Q^{\pi_{\theta}}(s, a) - b^{\pi_{\theta}}(s)) \nabla_{\theta} \log(\pi_{\theta}(s, a))]$$

for any baseline  $b^{\pi_{\theta}}(\cdot)$ .

## Actor-critic methods

In the vanilla policy gradient algorithm, we essentially approximated the  $Q$ -value function by Monte-Carlo estimation. In every iteration, we sampled multiple independent trajectory to implicitly do a  $Q$ -value estimation. Thus, as the policy changes, a new gradient is estimated independently of past estimates. In the actor-critic method, we use  $Q$ -function approximation in the gradient estimate. The actor-critic algorithm simultaneously/alternatively updates the  $Q$ -function approximation as well as policy parameters, as it sees more samples. Over iterations, as policy changes, the  $Q$ -function approximations also improves with more sample observations.

Before describing the algorithm, let's first understand the requirements from such an approximation - what kind of  $Q$ -function approximations are desirable (at least in ideal conditions)?

## Theorem 1 (Sutton et al. [1999])

If function  $f_\omega$  is compatible with policy parametrization  $\theta$  in the sense that for every  $s, a$ ,

$$\nabla_\omega f_\omega(s, a) = \frac{1}{\pi_\theta(s, a)} \nabla_\theta \pi_\theta(s, a) = \nabla_\theta \log(\pi_\theta(s, a))$$

And, further we are given parameter  $\omega$  which is a stationary point of the following least squares problem:

$$\min_\omega \mathbb{E}_{s \sim d^{\pi_\theta}, a \sim \pi_\theta(s)} \left[ (Q^{\pi_\theta}(s, a) - b(s; \theta) - f_\omega(s, a))^2 \right]$$

where  $b(\cdot; \theta)$  is any baseline, which may depend on the current policy  $\pi_\theta$ . Then,

$$\nabla_\theta \rho(\pi_\theta) = \mathbb{E}_{s \sim d^{\pi_\theta}} \mathbb{E}_{a \in \pi_\theta(s)} [f_\omega(s, a) \nabla_\theta \log(\pi_\theta(s, a))]$$

That is, function approximation  $f_\omega$  can be used in place of  $Q$ -function to obtain gradient with respect to  $\theta$ .

# Proof of Policy gradient theorem

(Here, we abuse the notation and use  $\mathbb{E}_{s \sim d^{\pi_\theta}} [x]$  as a shorthand for  $\sum_s d^{\pi_\theta}(s)x$ . This is not technically correct in the discounted case since in that case  $d^{\pi_\theta}(s) = \mathbb{E}_{s_1} [\sum_{t=1}^{\infty} \text{Pr}(s_t = s; \pi, s_1) \gamma^{t-1}]$ , which is not a distribution. In fact in discounted case,  $(1 - \gamma)d^{\pi_\theta}$  is a distribution.)

**Proof.** Given  $\theta$ , for stationary point  $\omega$  of the least squares problem:

$$\mathbb{E}_{s \sim d^{\pi_\theta}, a \sim \pi_\theta(s, \cdot)} [(Q^{\pi_\theta}(s, a) - b(s; \theta) - f_\omega(s, a)) \nabla_\omega f_\omega(s, a)] = 0$$

Substituting the compatibility condition:

$$\mathbb{E}_{s \sim d^{\pi_\theta}, a \sim \pi_\theta(s, \cdot)} \left[ (Q^{\pi_\theta}(s, a) - b(s; \theta) - f_\omega(s, a)) \nabla_\theta \pi_\theta(s, a) \frac{1}{\pi_\theta(s, a)} \right] = 0$$



# Proof of Policy gradient theorem

Or,

$$\sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) (Q^{\pi_\theta}(s, a) - b(s; \theta) - f_\omega(s, a)) = 0$$

Since  $b(s; \theta) \sum_a \nabla_\theta \pi_\theta(s, a) = 0$

$$\sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) (Q^{\pi_\theta}(s, a) - f_\omega(s, a)) = 0$$

using this with the policy gradient theorem, we get

$$\nabla_\theta \rho(\pi_\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) f_\omega(s, a)$$

## Example: softmax policy

- Consider policy set parameterized by  $\theta$  such that given  $s \in S$ , probability of picking action  $a \in A$  is given by:

$$\pi_{\theta}(s, a) = \frac{e^{\theta^{\top} \phi_{sa}}}{\sum_{a' \in A} e^{\theta^{\top} \phi_{sa'}}$$

where each  $\phi_{sa}$  is an  $\ell$ -dimensional feature vector characterizing state-action pair  $s, a$ . This is a popular form of parameterization. Here,

$$\nabla_{\theta} \pi_{\theta}(s, a) = \phi_{sa} \pi_{\theta}(s, a) - \left( \sum_{a' \in A} \phi_{sa'} \pi_{\theta}(s, a') \right) \pi_{\theta}(s, a)$$

## Example: softmax policy

- Meeting the compatibility condition in Theorem 1 requires that

$$\nabla_{\omega} f_{\omega}(s, a) = \frac{1}{\pi_{\theta}(s, a)} \nabla_{\theta} \pi_{\theta}(s, a) = \phi_{sa} - \sum_{a' \in A} \phi_{sa'} \pi_{\theta}(s, a')$$

- A natural form of  $f_{\omega}(s, a)$  satisfying this condition is:

$$f_{\omega}(s, a) = \omega^{\top} \left( \phi_{sa} - \sum_{b \in A} \phi_{sb} \pi_{\theta}(s, b) \right)$$

- Thus  $f_{\omega}$  must be linear in the same features as the policy, except normalized to be mean zero for each state. In this sense it is better to think of  $f_{\omega}$  as an approximation of the advantage function,  $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$ , rather than  $Q^{\pi}$ .

## Example: Gaussian policy for continuous action spaces

- In continuous action spaces, it is natural to use Gaussian policy. Given state  $s$ , the probability of action  $a$  is given as:

$$\pi_{\theta}(s, a) = \mathcal{N}(\phi(s)^T \theta, \sigma^2)$$

for some constant  $\sigma$ . Here  $\phi(s)$  is a feature representation of  $s$ . Then, compatibility condition for  $f_{\omega}(s, a)$ :

$$\nabla_{\omega} f_{\omega}(s, a) = \nabla_{\theta} \log(\pi_{\theta}(s, a)) = \nabla_{\theta} \frac{-(a - \theta^T \phi(s))^2}{2\sigma^2} = \frac{(\theta^T \phi(s) - a)}{\sigma^2} \phi(s)$$

- For  $f_{\omega}$  to satisfy this, it must be linear in  $\omega$ , e.g.,

$$f_{\omega}(s, a) = \frac{(\theta^T \phi(s) - a)}{\sigma^2} \phi(s)^T \omega$$

# Policy iteration algorithm with function approximation

Let  $f_\omega(\cdot, \cdot)$  be such that  $\nabla f_\omega(s, a) = \nabla_\theta \log \pi_\theta(s, a)$  for all  $\omega, \theta, s, a$ .

Initialize  $\theta_1, \pi_1 := \pi_{\theta_1}$ . Pick step sizes  $\alpha_1, \alpha_2, \dots$ .

In iteration  $k = 1, 2, 3, \dots$ ,

- **Policy evaluation:** Find  $w_k = w$  such that

$$\mathbb{E}_{s \sim d^{\pi_k}} \mathbb{E}_{a \sim \pi_k(s)} [(Q^{\pi_k}(s, a) - b_k(s) - f_\omega(s, a)) \nabla_\theta \log(\pi_k(s, a))] = 0$$

(Here,  $d^{\pi_k}$  is not normalized to 1, and sums to  $1/(1 - \gamma)$ .)

- **Policy improvement:**

$$\theta_{k+1} \leftarrow \theta_k + \alpha_k \mathbb{E}_{s \sim d^{\pi_k}} \mathbb{E}_{a \sim \pi_k(s)} [f_\omega(s, a) \nabla_\theta \log(\pi_k(s, a))]$$

A similar algorithm appears in Konda and Tsitsiklis [1999].

# Convergence Guarantees

Following version of convergence guarantees were provided by Sutton et al. [1999] for infinite horizon MDPs (average or discounted).

## Theorem 2 (Sutton et al. [1999])

Given  $\alpha_1, \alpha_2, \dots$ , such that

$$\lim_{T \rightarrow \infty} \sum_{k=1}^T \alpha_k = \infty, \lim_{T \rightarrow \infty} \sum_{k=1}^T \alpha_k^2 < \infty$$

and  $\max_{\theta, s, a, i, j} \frac{\partial^2 \pi_{\theta}(s, a)}{\partial \theta_i \partial \theta_j} < \infty$ . Then, for  $\theta_1, \theta_2, \dots$ , obtained by the above algorithm,

$$\lim_{k \rightarrow \infty} \nabla_{\theta} \rho(\theta)|_{\theta_k} = 0$$

# Pseudocode: Vanilla Policy Gradient Algorithm

---

## Algorithm 1 Vanilla Policy Gradient Algorithm

---

- 1: Input: initial policy parameters  $\theta_0$ , initial value function parameters  $\phi_0$
- 2: **for**  $k = 0, 1, 2, \dots$  **do**
- 3:   Collect set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_k = \pi(\theta_k)$  in the environment.
- 4:   Compute rewards-to-go  $\hat{R}_t$ .
- 5:   Compute advantage estimates,  $\hat{A}_t$  (using any method of advantage estimation) based on the current value function  $V_{\phi_k}$ .
- 6:   Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t.$$

- 7:   Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.

- 8:   Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.

- 9: **end for**
-

# Outline

- 1 Policy gradient methods
  - Finite horizon MDP
  - Infinite horizon case
- 2 Actor-critic methods
- 3 TRPO and PPO**
- 4 MCTS and AlphaGo Zero



# Trust Region Policy Optimization (John Schulman, etc)

- Assume start-state distribution  $d_0$  is independent with policy
- Total expected discounted reward with policy  $\pi$

$$\eta(\pi) = E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t) \right]$$

- Advantage function:  $A_{\pi}(s, a) = Q_{\pi}(s, a) - V_{\pi}(s)$
- Between any two different policy  $\tilde{\pi}$  and  $\pi$

$$\begin{aligned} \eta(\tilde{\pi}) &= \eta(\pi) + E_{\tilde{\pi}} \left[ \sum_{t=0}^{\infty} \gamma^t A_{\pi}(s_t, a_t) \right] \\ &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_s P(s_t = s | \tilde{\pi}) \sum_a \tilde{\pi}(a|s) \gamma^t A_{\pi}(s, a) \\ &= \eta(\pi) + \sum_s \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \tilde{\pi}) \sum_a \tilde{\pi}(a|s) A_{\pi}(s, a) \\ &= \eta(\pi) + \sum_s d_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_{\pi}(s, a). \end{aligned}$$

# Trust Region Policy Optimization

- Find new policy  $\tilde{\pi}$  to maximize  $\eta(\tilde{\pi}) - \eta(\pi)$  for given  $\pi$ , that is

$$\max_{\tilde{\pi}} \eta(\pi) + \sum_s d_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_{\pi}(s, a)$$

- For simpleness, maximize the approximator

$$L_{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_s d_{\pi}(s) \sum_a \tilde{\pi}(a|s) A_{\pi}(s, a)$$

- Parameterize the policy  $\pi(a|s) := \pi_{\theta}(a|s)$

$$L_{\pi_{\theta_{old}}}(\pi_{\theta}) = \eta(\pi_{\theta_{old}}) + \sum_s d_{\pi_{\theta_{old}}}(s) \sum_a \pi_{\theta}(a|s) A_{\pi_{\theta_{old}}}(s, a)$$

# Why $L_{\pi_{\theta_{old}}}(\pi_{\theta})$ ?

- A sufficiently small step  $\theta_{old} \rightarrow \theta$  improves  $L_{\pi_{\theta_{old}}}(\pi_{\theta})$  also improves  $\eta$

$$L_{\pi_{\theta_{old}}}(\pi_{\theta_{old}}) = \eta(\pi_{\theta_{old}}),$$
$$\nabla_{\theta} L_{\pi_{\theta_{old}}}(\pi_{\theta})|_{\theta=\theta_{old}} = \nabla_{\theta} \eta(\pi_{\theta})|_{\theta=\theta_{old}}.$$

- Lower bounds on the improvement of  $\eta$

$$\eta(\pi_{\theta_{new}}) \geq L_{\pi_{\theta_{old}}}(\pi_{\theta_{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

where

$$\epsilon = \max_s |E_{a \sim \pi_{\theta_{new}}} A_{\pi_{\theta_{old}}}(s, a)|$$

$$\alpha = D_{TV}^{max}(\pi_{\theta_{old}} || \pi_{\theta_{new}}) = \max_s D_{TV}(\pi_{\theta_{old}}(\cdot|s) || \pi_{\theta_{new}}(\cdot|s))$$

# Lower bound

- TV divergence between two distribution  $p, q$  (discrete case)

$$D_{TV}(p||q) = \frac{1}{2} \sum_X |p(X) - q(X)|$$

- KL divergence between two distribution  $p, q$  (discrete case)

$$D_{KL}(p||q) = \sum_X p(X) \log \frac{p(X)}{q(X)}$$

- $(D_{TV}(p||q))^2 \leq D_{KL}(p||q)$  (Pollard(2000),Ch.3)
- Thus obtain a lower bound

$$\eta(\pi_{\theta_{new}}) \geq L_{\pi_{\theta_{old}}}(\pi_{\theta_{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha$$

where

$$\alpha = D_{KL}^{max}(\pi_{\theta_{old}}||\pi_{\theta_{new}}) := \max_s D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_{\theta_{new}}(\cdot|s))$$

# Practical algorithm

- The penalty coefficient  $\frac{2\epsilon\gamma}{(1-\gamma)^2}$  is large in practice, which yields small update
- Take a constraint on the KL divergence, i.e., a trust region constraint:

$$\begin{aligned} \max_{\theta} \quad & L_{\pi_{\theta_{old}}}(\pi_{\theta}) \\ \text{s.t.} \quad & D_{KL}^{max}(\pi_{\theta_{old}} || \pi_{\theta}) \leq \delta \end{aligned}$$

- A heuristic approximation

$$\begin{aligned} \max_{\theta} \quad & L_{\pi_{\theta_{old}}}(\pi_{\theta}) \\ \text{s.t.} \quad & \overline{D}_{KL}^{\rho_{\pi_{\theta_{old}}}}(\pi_{\theta_{old}} || \pi_{\theta}) \leq \delta \end{aligned}$$

where

$$\overline{D}_{KL}^{\rho_{\pi_{\theta_{old}}}}(\pi_{\theta_{old}} || \pi_{\theta}) = E_{\pi_{\theta_{old}}}(D_{KL}(\pi_{\theta_{old}}(\cdot|s) || \pi_{\theta}(\cdot|s)))$$

- The objective and constraint are both zero when  $\theta = \theta_k$ . Furthermore, the gradient of the constraint with respect to  $\theta$  is zero when  $\theta = \theta_k$ .
- The theoretical TRPO update isn't the easiest to work with, so TRPO makes some approximations to get an answer quickly. We Taylor expand the objective and constraint to leading order around  $\theta_k$ :

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

resulting in an approximate optimization problem,

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

# TRPO

- By happy coincidence, the gradient  $g$  of the surrogate advantage function with respect to  $\theta$ , evaluated at  $\theta = \theta_k$ , is exactly equal to the policy gradient,  $\nabla_{\theta} J(\pi_{\theta})$
- This approximate problem can be analytically solved by the methods of Lagrangian duality, yielding the solution:

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

- TRPO adds a modification to this update rule: a backtracking line search,

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

where  $\alpha \in (0, 1)$  is the backtracking coefficient, and  $j$  is the smallest nonnegative integer such that  $\pi_{\theta_{k+1}}$  satisfies the KL constraint and produces a positive surrogate advantage.

- computing and storing the matrix inverse,  $H^{-1}$ , is painfully expensive when dealing with neural network policies with thousands or millions of parameters. TRPO sidesteps the issue by using the ‘conjugate gradient’ algorithm to solve  $Hx = g$  for  $x = H^{-1}g$ , requiring only a function which can compute the matrix-vector product  $Hx$  instead of computing and storing the whole matrix  $H$  directly.

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



# Pseudocode: TRPO

---

## Algorithm 2 Trust Region Policy Optimization

---

1: Input: initial policy  $\theta_0$ , initial value function  $\phi_0$ , KL-divergence limit  $\delta$ , backtracking coefficient  $\alpha$

2: **for**  $k = 0, 1, 2, \dots$  **do**

3:   Collect set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_k = \pi(\theta_k)$  in the environment.

4:   Compute rewards-to-go  $\hat{R}_t$ .

5:   Compute advantage estimates,  $\hat{A}_t$  (using any method of advantage estimation) based on the current value function  $V_{\phi_k}$ .

6:   Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t.$$

7:   Use the conjugate gradient algorithm to compute  $\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k$ , where  $\hat{H}_k$  is the Hessian of the sample average KL-divergence.

8:   Update the policy by backtracking line search with  $\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k$ , where  $j$  is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

9:   Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.

10: **end for**

---

# Proximal Policy Optimization (PPO)

- PPO is motivated by the same question as TRPO: how can we take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause performance collapse? Where TRPO tries to solve this problem with a complex second-order method, PPO is a family of first-order methods that use a few other tricks to keep new policies close to old.
- **PPO-Penalty** approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient over the course of training so that it's scaled appropriately.
- **PPO-Clip** doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

# Key Equations

- PPO-clip updates policies via

$$\theta_{k+1} = \arg \max_{\theta} \mathbb{E}_{s,a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)],$$

typically taking multiple steps of (usually minibatch) SGD to maximize the objective. Here  $L$  is given by

$$L(s, a, \theta_k, \theta) = \min \left( \frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}(s, a)}, \text{clip} \left( \frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}(s, a)} \right),$$

in which  $\epsilon$  is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old.

- <https://openai.com/research/openai-baselines-ppo>

- This is a pretty complex expression, and it's hard to tell at first glance what it's doing, or how it helps keep the new policy close to the old policy. As it turns out, there's a considerably simplified version of this objective which is a bit easier to grapple with

$$L(s, a, \theta_k, \theta) = \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}(s, a)}, g(\epsilon, A^{\pi_{\theta_k}(s, a)}) \right),$$

where

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0. \end{cases}$$

- To figure out what intuition to take away from this, let's look at a single state-action pair  $(s, a)$ , and think of cases.

# Pseudocode: PPO

---

## Algorithm 3 PPO-Clip

---

- 1: Input: initial policy parameters  $\theta_0$ , initial value function parameters  $\phi_0$
- 2: **for**  $k = 0, 1, 2, \dots$  **do**
- 3:   Collect set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_k = \pi(\theta_k)$  in the environment.
- 4:   Compute rewards-to-go  $\hat{R}_t$ .
- 5:   Compute advantage estimates,  $\hat{A}_t$  (using any method of advantage estimation) based on the current value function  $V_{\phi_k}$ .
- 6:   Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

- typically via stochastic gradient ascent with Adam.
- 7:   Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

- typically via some gradient descent algorithm.
- 8: **end for**
-

# Reinforcement Learning from Human Feedback

RLHF: Nisan Stiennon, et al, Learning to summarize with human feedback, NeurIPS 2020

## 1 Collect human feedback

A Reddit post is sampled from the Reddit TL;DR dataset.



Various policies are used to sample a set of summaries.



Two summaries are selected for evaluation.



A human judges which is a better summary of the post.



"j is better than k"

## 2 Train reward model

One post with two summaries judged by a human are fed to the reward model.



The reward model calculates a reward  $r$  for each summary.



$r_j$

$r_k$

The loss is calculated based on the rewards and human label, and is used to update the reward model.

$$\text{loss} = \log(\sigma(r_j - r_k))$$

"j is better than k"

## 3 Train policy with PPO

A new post is sampled from the dataset.



The policy  $\pi$  generates a summary for the post.



The reward model calculates a reward for the summary.



The reward is used to update the policy via PPO.

$r$

# InstructGPT

provide detailed, accurate, and instructive responses to user queries.

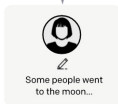
Step 1

**Collect demonstration data, and train a supervised policy.**

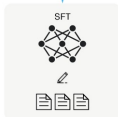
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



This data is used to fine-tune GPT-3 with supervised learning.



Step 2

**Collect comparison data, and train a reward model.**

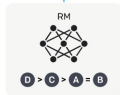
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



Step 3

**Optimize a policy against the reward model using reinforcement learning.**

A new prompt is sampled from the dataset.

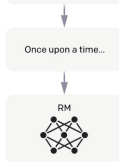


The policy generates an output.

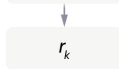


Once upon a time...

The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.



# ChatGPT

generate human-like text based on the input it's given, and it can carry out a wide-ranging conversation on various topics.

## Step 1

Collect demonstration data and train a supervised policy.

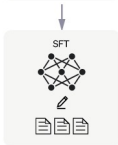
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



This data is used to fine-tune GPT-3.5 with supervised learning.



## Step 2

Collect comparison data and train a reward model.

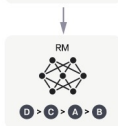
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



## Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.



The PPO model is initialized from the supervised policy.

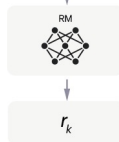


The policy generates an output.

The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.





Simulated exams	GPT-4 <small>estimated percentile</small>	GPT-4 (no vision) <small>estimated percentile</small>	GPT-3.5 <small>estimated percentile</small>
Uniform Bar Exam (MBE+MEE+MPT) <sup>1</sup>	298/400 <small>-90th</small>	298/400 <small>-90th</small>	213/400 <small>-10th</small>
LSAT	163 <small>-88th</small>	161 <small>-83rd</small>	149 <small>-40th</small>
SAT Evidence-Based Reading & Writing	710/800 <small>-93rd</small>	710/800 <small>-93rd</small>	670/800 <small>-87th</small>
SAT Math	700/800 <small>-89th</small>	690/800 <small>-89th</small>	590/800 <small>-70th</small>
Graduate Record Examination (GRE) Quantitative	163/170 <small>-80th</small>	157/170 <small>-62nd</small>	147/170 <small>-25th</small>
Graduate Record Examination (GRE) Verbal	169/170 <small>-99th</small>	165/170 <small>-96th</small>	154/170 <small>-63rd</small>
Graduate Record Examination (GRE) Writing	4/6 <small>-54th</small>	4/6 <small>-54th</small>	4/6 <small>-54th</small>
USABO Semifinal Exam 2020	87/150 <small>99th-100th</small>	87/150 <small>99th-100th</small>	43/150 <small>31st-33rd</small>
USNCO Local Section Exam 2022	36/60	38/60	24/60
Medical Knowledge Self-Assessment Program	75%	75%	53%
Codeforces Rating	392 <small>below 5th</small>	392 <small>below 5th</small>	260 <small>below 5th</small>
AP Art History	5 <small>86th-100th</small>	5 <small>86th-100th</small>	5 <small>86th-100th</small>
AP Biology	5 <small>85th-100th</small>	5 <small>85th-100th</small>	4 <small>62nd-85th</small>
AP Calculus BC	4 <small>43rd-59th</small>	4 <small>43rd-59th</small>	1 <small>0th-7th</small>

# ChatGPT: PPO over DQN

🚀 Model: GPT-4

why is PPO used to train chatgpt and gpt4? can the algorithms based on q-learning be applied as well?

Proximal Policy Optimization (PPO) is used to train models like ChatGPT and GPT-4 because it is a widely recognized and effective algorithm for reinforcement learning (RL) in complex environments with large state and action spaces. PPO is particularly well-suited for training large-scale models due to its combination of stability, scalability, and sample efficiency.

The main reason PPO is preferred over algorithms based on Q-learning, such as Deep Q-Network (DQN) or Deep Deterministic Policy Gradient (DDPG), is that PPO is designed for policy optimization in continuous action spaces. Q-learning based algorithms, on the other hand, work well for discrete action spaces, but can be less efficient and less stable in continuous action spaces, which are often encountered in natural language processing tasks.

However, this does not mean Q-learning-based algorithms cannot be applied to train models like ChatGPT or GPT-4. Researchers can still experiment with these algorithms to find novel ways to adapt them to large-scale language models or for specific applications. It's just that, based on current understanding and empirical results, PPO has shown to be more effective and reliable for this purpose.

# ChatGPT: PPO over SAC

how about SAC for ChatGPT or GPT-4?

Soft Actor-Critic (SAC) is another reinforcement learning algorithm that is designed to handle continuous action spaces. It combines the advantages of policy gradient methods, like PPO, with the stability and exploration properties of Q-learning algorithms, like DDPG. SAC has been successful in various continuous control tasks, and it could potentially be applied to train models like ChatGPT or GPT-4.

However, SAC has not been extensively tested and applied to large-scale language models like GPT-4, so it is difficult to predict its performance and stability compared to PPO. There may be challenges in adapting SAC for natural language processing tasks due to the unique characteristics of language data and the large state and action spaces involved. Furthermore, text generation tasks may require additional modifications to the SAC algorithm to handle the complexities associated with language.

In summary, while SAC has potential as an alternative reinforcement learning algorithm for training ChatGPT or GPT-4, more research and experimentation would be necessary to determine its effectiveness and suitability for this purpose. The existing success of PPO in training large-scale language models makes it a more established and reliable choice at the moment. However, exploring the use of SAC and other RL algorithms in this context could lead to new insights and improved performance.

# ChatGPT: PPO challenges

what are the main challenges or difficulties of applying PPO to ChatGPT or GPT-4?

Applying Proximal Policy Optimization (PPO) to large-scale language models like ChatGPT or GPT-4 presents several challenges and difficulties: 

1. **Large action space:** Language models have a vast action space, as they need to choose from a large vocabulary of words or tokens at each time step. This makes the learning problem more complex compared to many traditional reinforcement learning tasks.
2. **Sparse and delayed rewards:** In natural language processing tasks, rewards are often sparse and delayed, making it difficult to assign credit or blame to specific actions. For example, the quality of a generated sentence or paragraph may depend on several words, and the impact of a single word might not be immediately apparent.
3. **Exploration vs. exploitation:** Striking the right balance between exploration (trying new actions) and exploitation (using actions that are known to work well) is critical in reinforcement learning. In the context of language models, this can be particularly challenging due to the vast action space and the difficulty in evaluating the quality of generated text.
4. **Non-stationary and non-Markovian environments:** In language generation tasks, the model needs to adapt to different contexts and respond coherently to user inputs. This means that the environment is often non-stationary, as the model needs to adjust its policy based on context. Additionally, the natural language generation process often has long-term dependencies, which violate the Markov property assumed in many reinforcement learning algorithms, including PPO.
5. **Curriculum learning and reward shaping:** Designing an appropriate curriculum for training language models and defining a suitable reward function can be challenging. The reward function needs to encourage both syntactically correct and semantically meaningful text generation while avoiding pitfalls like over-optimizing for specific metrics or generating overly generic responses.
6. **Computational resources:** Training large-scale language models like GPT-4 using PPO requires significant computational resources, both in terms of hardware and time. This can make it difficult for researchers to experiment with different configurations, hyperparameters, and reward functions.

# Outline

- 1 Policy gradient methods
  - Finite horizon MDP
  - Infinite horizon case
- 2 Actor-critic methods
- 3 TRPO and PPO
- 4 MCTS and AlphaGo Zero

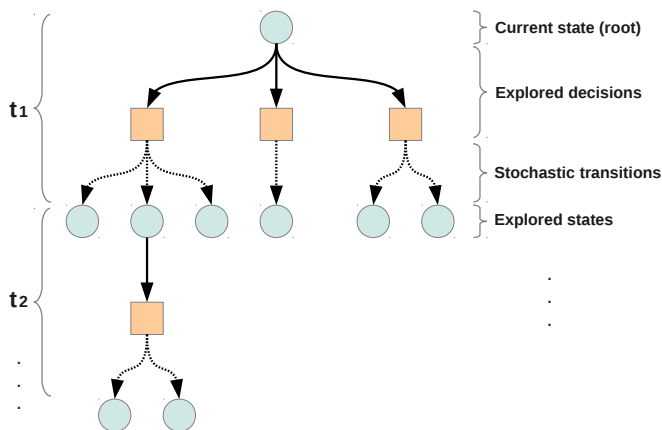
# Monte-Carlo Tree Search (MCTS)

- MCTS is a recent algorithm for *sequential decision making*
- MCTS is a versatile algorithm (it does not require knowledge about the problem)
- especially, does not require any knowledge about the Bellman value function
- stable on high dimensional problems
- it outperforms all other algorithms on some problems (difficult games like Go, general game playing, . . . )

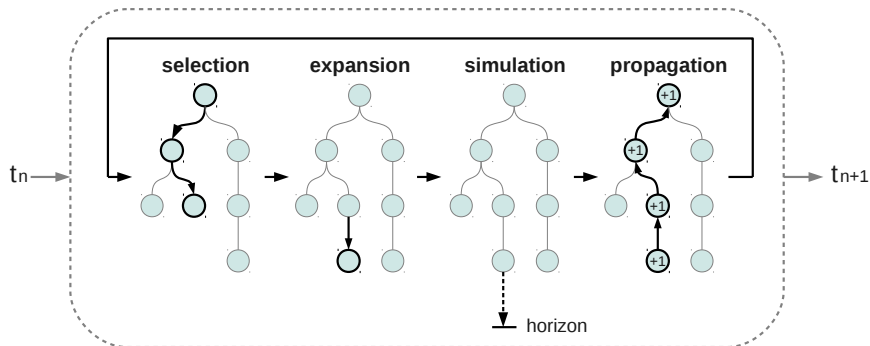
# MCTS

Problems are represented as a tree structure:

- blue circles = states
- plain edges + red squares = decisions
- dashed edges = stochastic transitions between two states



# Main steps of MCTS



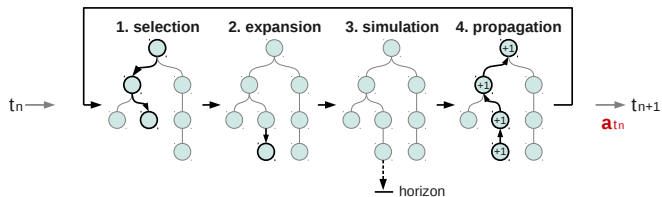


# Main steps of MCTS

Starting from an initial state:

- 1 **Selection:** select the state we want to expand from
- 2 **Expansion:** One (or more) child nodes are added to expand the tree, according to the available actions.
- 3 **Simulation:** A simulation is run from the new node(s) according to the default policy (pick actions randomly) to produce an outcome.
- 4 **Back-propagation** of some information:
  - $N(s, a)$  : number of times decision  $a$  has been simulated in  $s$
  - $N(s)$  : number of time  $s$  has been visited in simulations
  - $Q(s, a)$  : mean reward of simulations where  $a$  was chosen in  $s$

# Main steps of MCTS

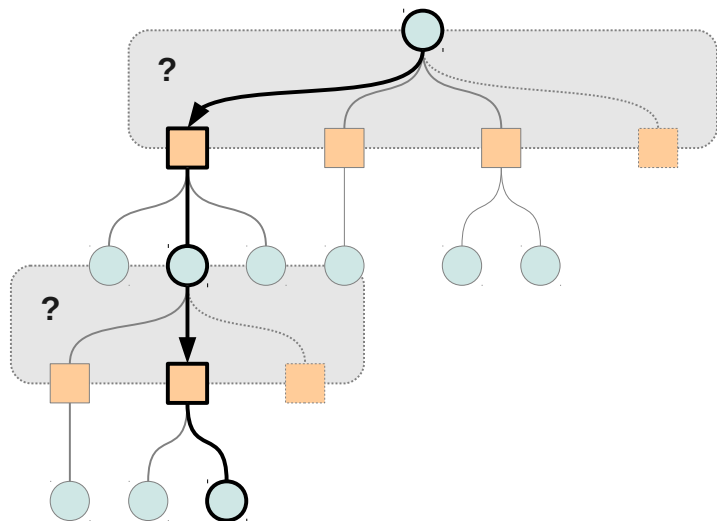


## The selected decision

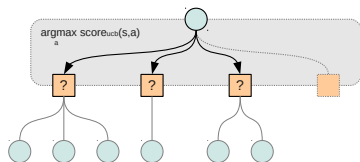
$a_{t_n}$  = the most visited decision from the current state (root node)

# Selection step

How to select the state to expand ?



# How to select the state to expand ?

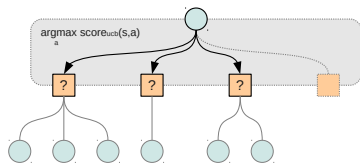


The *selection* phase is driven by Upper Confidence Bound (UCB):

$$\text{score}_{\text{ucb}}(s, a) = \underbrace{Q(s, a)}_1 + \underbrace{\sqrt{\frac{\log(2 + N(s))}{2 + N(s, a)}}}_2$$

- 1 mean reward of simulations including action  $a$  in state  $s$
- 2 the uncertainty on this estimation of the action's value

# How to select the state to expand ?



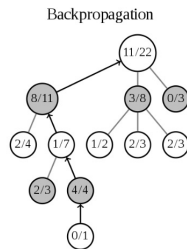
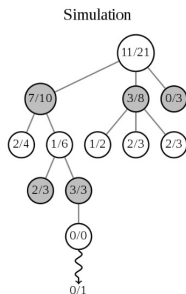
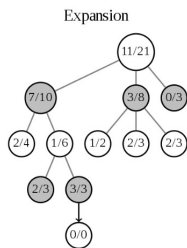
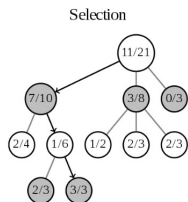
The *selection* phase is driven by Upper Confidence Bound (UCB):

$$\text{score}_{\text{ucb}}(s, a) = \underbrace{Q(s, a)}_1 + \underbrace{\sqrt{\frac{\log(2 + N(s))}{2 + N(s, a)}}}_2$$

The selected action:

$$a^* = \arg \max_a \text{score}_{\text{ucb}}(s, a)$$

# Example: Back-propagation



# AlphaZero

2018/12/7, AlphaZero at “Science”. It demonstrates learning chess, shogi and go, tabula rasa without any domain-specific human knowledge or data, only using self-play. The evaluation is performed against strongest programs available.

A

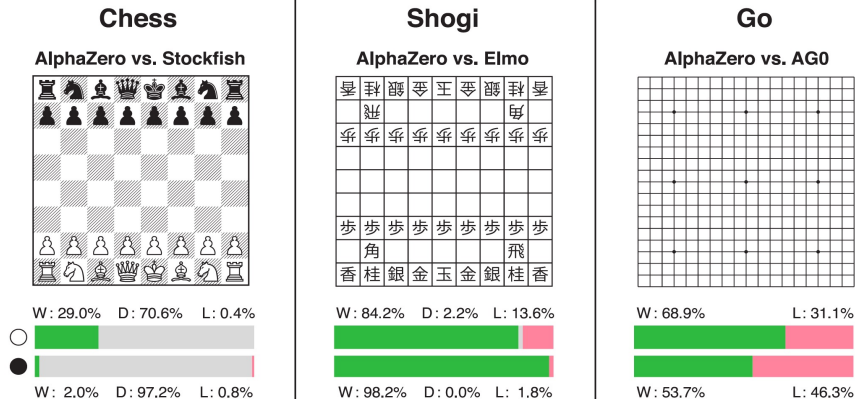


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

# AlphaZero

- AlphaZero uses a neural network predicting  $(p(s), v(s)) = f(s, \theta)$  for a given state  $s$ 
  - $p(s)$  is a vector of move probabilities
  - $v(s)$  is expected outcome of the game in range  $[-1, 1]$ .
- Unlike the standard MCTS, Alpha Go Zero does not use a default policy to perform a rollout in order to achieve an estimate of the value of a state.
- By a sequence of simulated self-play games, the search can improve the estimate of  $p$  and  $v$ , and can be considered a powerful policy evaluation operator given a network  $f$  predicting policy  $p$  and value estimate  $v$ , MCTS produces a more accurate policy  $\pi$  and better value estimate  $w$  for a given state:

$$(\pi(s), w(s)) \leftarrow \text{MCTS}(p(s), v(s), f) \text{ for } (p(s), v(s)) = f(s, \theta).$$



# AlphaZero: MCTS

MCTS keeps a tree of currently explored states from a fixed root state. Each node corresponds to a game state and to every non-root node we got by performing an action  $a$  from the parent state. Each state-action pair  $(s, a)$  stores the following set of statistics:

- visit count  $N(s, a)$
- total action-value  $W(s, a)$
- mean action-value  $Q(s, a) = W(s, a)/N(s, a)$ , which is not stored explicitly
- prior probability  $P(s, a)$  of selecting action  $a$  in state  $s$

# AlphaZero: UCB

- Each simulation starts in the root node and finishes in a leaf node  $s_L$ . In a state  $s_t$ , an action is selected using a variant of PUCT algorithm as

$$a_t = \arg \max_a Q(s_t, a) + U(s_t, a)$$

where

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

with  $C(s) = \log \left( \frac{1+N(s)+c_{base}}{c_{base}} \right) + c_{init}$ .

- In the Alphazero paper,  $c_{init} = 1.25$  and  $c_{base} = 19652$ .

# AlphaZero

When reaching a leaf node  $s_L$ ,

- evaluate it by the network, generating  $(p, v)$
- add all its children with  $N = W = 0$  and the prior probability  $p$ ,
- in the backward pass for all  $t \leq L$ , we update the statistics in nodes by performing
  - $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$ , and
  - $W(s_t, a_t) \leftarrow W(s_t, a_t) \pm v$ , depending on the player on turn.

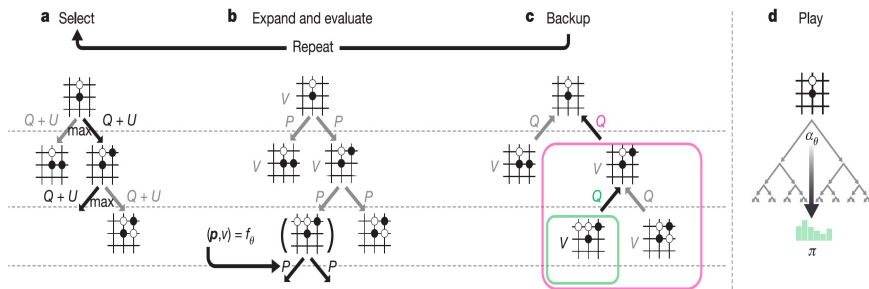


Figure 2 of the paper "Mastering the game of Go without human knowledge" by David Silver et al.

# AlphaZero

- The MCTS runs usually several hundreds simulations in a single tree. The result is a distribution proportional to exponentiated visit counts  $N(s_{root}, a)^{1/\tau}$  using a temperature ( $\tau = 1$  is mostly used), together with the predicted value function.
- The next move is chosen as either:
  - proportional to visit counts  $N(s_{root}, \cdot)^{1/\tau}$

$$\pi_{root}(a) \sim N(s_{root}, \cdot)^{1/\tau}$$

- deterministically as the most visited action

$$\pi_{root} = \arg \max_a N(s_{root}, a)$$

- During self-play, the stochastic policy is used for the first 30 moves of the game, while the deterministic is used for the rest of the moves. (This does not affect the internal MCTS search, there we always sample according to PUCT rule.)

# AlphaZero: Loss function

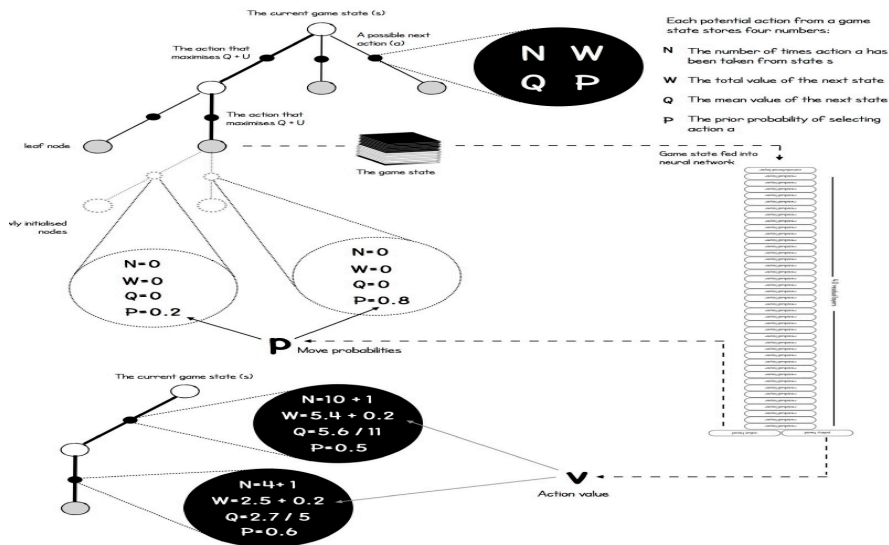
- The network is trained from self-play games.
- A game is played by repeatedly running MCTS from a state  $s_t$  and choosing a move  $a_t \sim \pi_t$ , until a terminal position  $s_T$  is encountered, which is then scored according to game rules as  $z \in \{-1, 0, 1\}$ .
- Finally, the network parameters are trained to minimize

$$L = (z - v)^2 + \pi^\top \log p + c \|\theta\|^2$$

- a mean squared error between the predicted outcome  $v$  and the simulated outcome  $z$
- a crossentropy/KL divergence for the action distribution, i.e., the similarity of the policy vector  $p$  and the search probabilities  $\pi$
- L2 regularization

# AlphaGo Zero

## How AlphaGo Zero chooses its next move

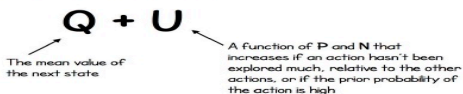


# AlphaGo Zero

## First, run the following simulation 1,600 times...

Start at the root node of the tree (the current game state)

### 1. Choose the action that maximises...



Early on in the simulation,  $U$  dominates (more exploration), but later,  $Q$  is more important (less exploration)

### 2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

$P$  Move probabilities

$V$  Value of the state (for the current player)

The move probabilities  $p$  are attached to the new feasible actions from the leaf node

### 3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$$\begin{aligned} N &\rightarrow N + 1 \\ W &\rightarrow W + v \\ Q &= W / N \end{aligned}$$

# AlphaGo Zero

## ...then select a move

After 1,600 simulations, the move can either be chosen:

**Deterministically** (for competitive play)

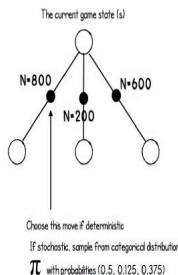
Choose the action from the current state with greatest N

**Stochastically** (for exploratory play)

Choose the action from the current state from the distribution

$$\pi \sim N^{1/\tau}$$

where  $\tau$  is a temperature parameter, controlling exploration



## Other points

- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded



## SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored



The game state  
(see 'What is a Game  
State section')

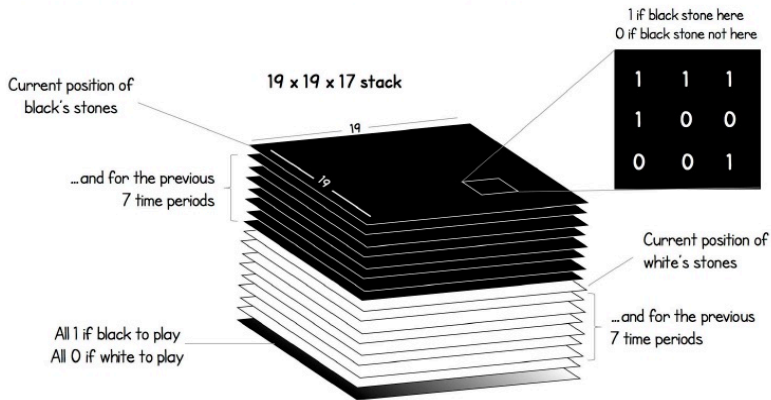


The search probabilities  
(from the MCTS)



The winner  
(+1 if this player won, -1 if  
this player lost - added once  
the game has finished)

## WHAT IS A 'GAME STATE'



This stack is the input to the deep neural network

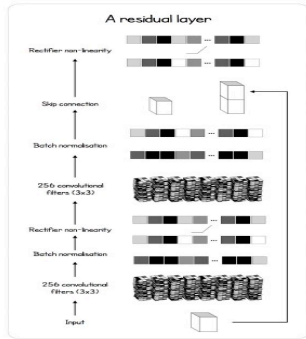
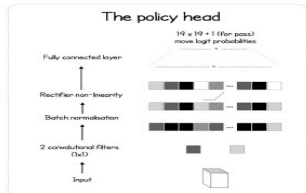
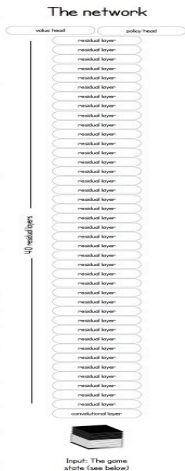
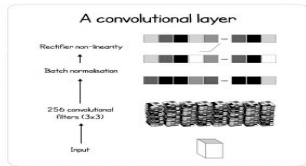
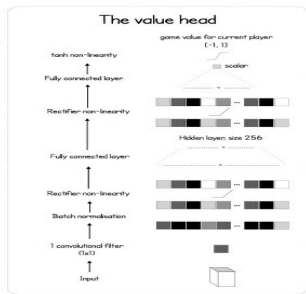
# AlphaGo Zero

## THE DEEP NEURAL NETWORK ARCHITECTURE

How AlphaGo Zero assesses new positions

The network learns 'tabula rasa' (from a blank slate)

At no point is the network trained using human knowledge or expert moves



## RETRAIN NETWORK

Optimise the network weights

### A TRAINING LOOP

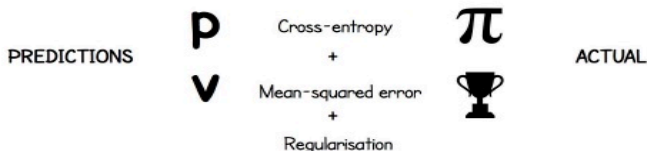
Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions

– The game states are the input (see 'Deep Neural Network Architecture')

#### Loss function

Compares predictions from the neural network with the search probabilities and actual winner



After every 1,000 training loops, evaluate the network

## EVALUATE NETWORK

Test to see if the new network is stronger

Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player

