Contents

1 [TODO] Math fundamentals]

2 [TODO] Supervised learning fundamentals]

3 [TODO] Neural networks

3.1 LSTM

Link: https://arxiv.org/pdf/1503.04069.pdf, also http://colah.github.io/posts/2015-08-Understanding-LSTMs/
LSTM [5] [3] ...

4 [TODO] Hyper-parameter tuning

5 Reinforcement learning

RL in 3 Equations (from Nando de Freitas talk at Khipu 2019)
Most RL algorithms can be divided into three families: policy gradient algorithms (Eqn. 1), Q-value policy gradient algorithms (Eqn. 2) and Q-value algorithms (Eqn. 3).

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau[(\sum_{t=0}^{T} R_t \nabla_\theta \log \pi_\theta(a_t \mid s_t))] \quad (1)$$

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{d_{\pi(s)}\pi(a|s)}[Q_\pi(s,a)\nabla \log \pi_\theta(a \mid s)] \quad (2)$$

$$Q^*(s,a) = \mathbb{E}_{s'}[r(s,a,s') + \gamma max_{a'} Q^*(s',a') \mid s,a] \quad (3)$$

where $J(\pi_\theta)$ is the expected return of a policy π parametrized with θ, τ is a trajectory with length T, and $d_{\pi(s)}$ is the probability distribution of the states.

5.1 Key concepts

5.1.1 Policy gradient

Given π_θ, a stochastic policy parametrized with θ. Our objective is to maximize the expected return

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \quad (4)$$

where $R(\tau)$ is the sum of discounted rewards over the sampled trajectory $\tau = (s_0, a_0, \ldots, s_{t+1})$. The aim is to optimize the policy by gradient ascent, such that

$$\theta_{t+1} = \theta_t + \alpha J(\pi_\theta)\Big|_{\theta_t} \tag{5}$$

How to find a numberically computable expression? The probability of a trajectory τ whose policy is based on π with parameters θ, is given by

$$P(\tau \mid \theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t) \tag{6}$$

where $\rho_0(s_0)$ is the probability distribution of the initial state.
Considering that $\nabla \log(x) = \frac{1}{x}$, therefore

$$\begin{aligned} \nabla_\theta \log(P(\tau \mid \theta)) &= \frac{1}{P(\tau \mid \theta)} \nabla_\theta P(\tau \mid \theta) \\ P(\tau \mid \theta) \nabla_\theta \log(P(\tau \mid \theta)) &= \nabla_\theta P(\tau \mid \theta) \end{aligned}$$

On the other hand, log of a trajectory is given by

$$\log P(\tau \mid \theta) = \log \rho_0(s_0) + \sum_{t=1}^{T} [\log P(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)] \tag{7}$$

applying gradient

$$\nabla_\theta \log P(\tau \mid \theta) = \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=1}^{T} [\cancel{\nabla_\theta \log P(s_{t+1} \mid s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t \mid s_t)]$$

$$\nabla_\theta \log P(\tau \mid \theta) = \sum_{t=1}^{T} [\nabla_\theta \log \pi_\theta(a_t \mid s_t)] \tag{8}$$

Considering Eqn. 4 and Eqn. 8 and that $\nabla_\theta R(\tau)$ is also 0 as the return does not depend on θ, expanding the expectation it is derived that

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=1}^{T} [\nabla_\theta \log \pi_\theta(a_t \mid s_t)] R(\tau)] \tag{9}$$

This is called the **policy gradient theorem**. As it is an expectation, it can be estimated by a sample mean. Therefore, if we have a set of trajectories $D = \tau_1, \ldots, \tau_N$, a gradient ascent step can be done estimating the policy gradient with

$$\hat{g} = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} [\nabla_\theta \log \pi_\theta(a_t \mid s_t)] R(\tau) \tag{10}$$

A common form of Equation 9 and Equation 10 involves the estimated advantage $\hat{A}_t$ instead of the return $R(\tau)$.

Vanilla policy gradient is the simplest algorithm that makes use of this theorem. It collects trajectories $D = \tau_1, \ldots, \tau_N$ by running policy $\pi_i = \pi(\theta_i)$. Then, it uses the parameters ϕ of the value function V_ϕ to compute the advantage function $\hat{A}_t$ to estimate 10, and then performs two updates:

1. Update of the policy:
$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k$$
2. Update of the value function, by regression on the mean squared error:
$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$
where $\hat{R}_t$ are the discounted rewards.

Section based on `https://spinningup.openai.com/en/latest/user/introduction.html`

5.1.2 Importance sampling

Prediction problem: exploration policy b (considered as fixed/given, e.g. ϵ-greedy policy), target policy π (e.g. optimal policy, considered as fixed/given, a common case is the deterministic greedy policy). We have episodes of experience obtained by following policy b. How to estimate v_π or q_π having episodes that followed exploration policy b.

Assumption of coverage: $\pi(a \mid s) > 0$ implies $b(a \mid s) > 0$. Implication: b must be stochastic in state where it is not identical to π.

Importance sampling (IS) is a technique used for estimating expected values under one distribution given samples from another. In this context, IS is applied to off-policy learning by weighting returns according to the relative probability of their trajectories. Therefore, IS ratio over a trajectory is given by

$$IS = \prod_{i=1}^{t} \frac{\pi(a_i \mid s_i)}{b(a_i \mid s_i)} \tag{11}$$

Now, $V_b(s) = \mathbb{E}[R_t \mid s_t = s]$, where R_t is the sum of discounted rewards. Therefore, using the IS ratio,
$V_\pi(s) = \mathbb{E}[\prod_{i=1}^{t} \frac{\pi(a_i|s_i)}{b(a_i|s_i)} R_t \mid s_t = s]$

5.1.3 Reparametrization trick

`https://arxiv.org/pdf/1312.6114.pdf`
Let $Z \sim q_\phi(Z \mid X)$ be a conditional distribution. The reparametrization trick consists in expressing the random varible Z as a deterministic variable $Z = g_\phi(\varepsilon, X)$, being ε an auxiliary variable with an independent marginal $P(\varepsilon)$.

5.2 "Main" deep RL algorithms

5.2.1 DQN

Family: Q-value
TL;DR
Problem addressed Proposed approach

5.2.2 TRPO

Family: policy gradient
Link: `https://arxiv.org/abs/1502.05477`
TL;DR
TRPO [9] is an on-policy algorithm that updates policies π_θ by gathering pairs of (s, a) through experience, and updating iteratively the policy by finding the θ that solves an approximation of an optimization problem given by $\pi_{i+1} = \arg\max_\pi L_{\theta_i} = \mathbb{E}\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_o ld(a_t|s_t)}} A_t$ and constrained to $D_{KL}^{\max}(\theta_i, \theta) \leq \delta$. Such constraint is called the *trust region* constraint, because it allows the policy to change with a safe learning step that iteratively adapts over the course of the learning.
Problem addressed

1. Given the advantage $\eta(\tilde{\pi})$ of a policy $\tilde{\pi}$ over a policy π.
2. It was shown in Kakade and Langford (2002) that if $\pi_\theta(a \mid s)$ is a differentiable function of the parameter vector θ, then an approximation of the advantage $L_\theta(\tilde{\theta})$ (which used visitation frequency ρ_θ instead of $\rho_{\tilde{\theta}}$) will match η to first order, with a sufficiently small step α.

Problem: how to determine the sufficiently small step that improves the approximation $L_{\theta_{old}(\theta)}$, is also applicable to stochastic policies?

Proposed approach

1. Replace step-size α with a distance measure between π and $\tilde{\pi}$.
2. An algorithm similar to policy improvement is used to iteratively 1) calculate the advantages of the policy θ_i for every s, a and 2) solves the constrained optimization problem $\pi_{i+1} = \arg\max_\pi [L_{\theta_i} - CD_{KL}^{\max}(\theta_i, \theta)]$.
3. Using the penalty coefficient C will involve very small step sizes.
4. Therefore, an additional *trust region constraint* is added to relax the optimization problem, that now seeks to maximize L_θ, where the KL divergence is subject to be lesser than a certain δ.
5. Now, calculating the KL divergence as it is, is infeasible due that it is bounded by every point in the state space. Instead, it is calculated the average KL divergence using importance sampling, replacing the expectations for sample averages, Q-values for empirical estimate, and approximating through Monte Carlo.
6. Two schemes proposed for this estimation:
   (a) Single path estimation, where states s are sampled from ρ (the visitation frequency) and then a policy $\pi_{\theta_{old}}$ is simulated for a number of timesteps to generate a trajectory, making $b(a \mid s) = \pi_{\theta_{old}}$. Also, $Q_{\theta_{old}}$ is computed at each state-action pair taking the discounted sum of rewards along the trajectory.
   (b) Vine, where states s are sampled from ρ (the visitation frequency) and then a policy $\pi_{\theta_{old}}$ is simulated for a number of timesteps to generate a trajectory, just like in single path estimation. Then, a subset of N states of the trajectories are chosen (rollout set), and each of them are used to sample K actions according to $b(a_{k\in K} \mid s_{n\in N})$. According to the authors, $b(s \mid a) = \pi_{\theta_i(s|a)}$ performs well in continuous problems, while the uniform distribution performs well on discrete tasks such as Atari.
   $\hat{Q}_{\theta_i}(s_n, a_{n,k})$ is estimated by performing a rollout (i.e. a short trajectory) starting from state s_n and action $a_{n,k}$.
   Upsides against single path: better advantage estimation, less variance of the Q value. Downsides: need more simulations.

Summarized algorithm:

1. Use single path or vine to gather (s, a) pairs, estimating its Q-values with Monte Carlo and using the importance sampling ratio.

2. Averaging over samples, construct the estimated objective function and constraint.

3. Solve this optimization problem with constraint to update the policy vector θ. In the paper, authors use conjugate gradient algorithm followed by a line search.

5.2.3 DDPG

Family: Q policy gradient

Link: `https://arxiv.org/pdf/1509.02971.pdf`

TL;DR DDPG [6] is an off-policy actor critic algorithm that trains simultaneously a deterministic actor function a and a critic function Q with an experience memory. As in continuous tasks is expensive to find $\max_a Q(s, a)$, the maximum of Q is instead approximated by $Q(s, a(s \mid \theta_a))$.

Problem addressed

How to use Q-learning algorithms when the action space is continuous?

Proposed approach

DQN modified for actor critic. It learns a deterministic actor function $a(s \mid \theta_a)$ and a critic function $Q(s, a \mid \theta_c)$. As finding $\max_a Q(s, a)$ in continuous action spaces is computationally expensive, the max is instead approximated by $\max_a Q(s, a \mid \theta_c) \approx Q(s, a(s \mid \theta_a))$.

Critic is updated with a mean squared error loss as in DQN, and actor policy is updated using the sampled policy gradient and applying the chain rule to find the θ_a that maximizes

$$\nabla_{\theta_a} J(\pi_{\theta_a}) \approx \frac{1}{|N|} \sum_{s \in N} \nabla_{\theta_a} Q(s, a(s \mid \theta_a) \mid \theta_c) \nabla_{\theta_a} a(s \mid \theta_a) \tag{12}$$

where N is the batch $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{|N|}$ and $|N|$ its size.

As it is off-policy, DDPG uses a replay memory as in DQN to iteratively update its actor and critic networks by

$$\theta'_a = \rho \theta'_a + (1 - \rho) \nabla J(\pi_{\theta_a}) \tag{13}$$

$$\theta'_c = \rho \theta'_c + (1 - \rho) \theta_c \tag{14}$$

where $\rho << 1$. On the other hand, actions are selected by $Q(s, a(s \mid \theta_a)) + \varepsilon$, where ε is a random exploration noise.

5.2.4 A3C

Family: policy gradient, distributed

Link: `https://arxiv.org/pdf/1602.01783.pdf`

TL;DR

Asynchronous Advantage Actor Critic (A3C) [7] is an asyncronous policy-based model free algorithm that consists on multiple agents running on environment in parallel, accumulating gradients and updating it after several timesteps.

Problem addressed

Provide a different paradigm for deep RL algorithms that does not rely on experience replay for stabilizing learning and reducing training time and memory requirements.

Proposed approach

- Instead of using experience replay, multiple agents are run in parallel on multiple instances of the environment. Each agent runs in its own thread and interacts with its own copy of the environment, and they apply the gradients after accumulating it over multiple timesteps, instead of using minibatches. In addition, agents may use different exploration policies in each thread so as to improve robustness.
- As an actor critic architecture, it concurrently maintains a policy $\pi(a \mid s, \theta)$ and an estimation of the value function $V(s \mid \phi)$. Both are updated after a certain amount of timesteps t, and it is given by the estimate $\nabla_{\theta^-} \log \pi(a \mid s, \theta) A(s, a \mid \theta, \theta_v)$, being A an estimate of the advantage function. A target network θ^- is shared among all agents.
- Their synchronous, determinisic variant is called A2C.

5.2.5 [TODO] GAIL

Link: `https://arxiv.org/pdf/1606.03476.pdf`

TL;DR

GAIL [**?**] uses expert trajectories to learn a cost function over them and then learn an action policy...

Problem addressed

Learning a policy from example expert behavior in an efficient way. Other approaches include

- Behavioral cloning, which learns a policy as a supervised learning problem, over state-action pairs over expert trajectores. Downside: compounding error caused by covariance shift.

- Inverse Reinforcement Learning (IRL), which consists in learning a cost function over expert trajectories in which the expert is uniquely optimal. Downside: expensive to run.

Proposed approach

Generative Adversarial Imitation Learning (GAIL) ...

TODO

5.2.6 [TODO] ACER

Link: `https://arxiv.org/abs/1611.01224`

5.2.7 PPO

Link: `https://arxiv.org/abs/1707.06347`

TL;DR

PPO [10] is an on-policy algorithm that updates its policy $\pi_{\theta_{k+1}}$ by taking gradient ascent over an objetive function that limits (by clipping or applying a penalization based on the KL divergence) how far the new policy can go.

Problem addressed

How to take the biggest possible improvement step on a policy update without it being too much that the learning may collapse the performance?

TRPO implements a penalty that is necessary, but as it is, optimizing its objective may involve excessively large update steps on the policy, prone to unstabilities.

Proposed approach

Being $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ the probability ratio between the old policy and new policy, being $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} = 1$ when it is the same policy.

The approach of Proximal Policy Optimization (PPO) is to penalize changes of the policy that moves the coefficient $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ away from 1. For that, the objective function proposed is given by

$$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}[\min(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t(s,a), \text{clip}(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t(s,a))] \quad (15)$$

Put in other terms, if $\hat{A}_t(s,a) > 0$, this means that action a is improving the policy. Therfore, the objective will be given by

$$L_{\text{CLIP}}(\theta) = \min(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1+\epsilon)\hat{A}_t(s,a)$$

Otherwise, if $\hat{A}_t(s,a) < 0$, action a is detrimental for the policy, and the objective will be given by

$$L_{\text{CLIP}}(\theta) = \max(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1-\epsilon)\hat{A}_t(s,a)$$

In Equation 15, the aim is 1) to prevent the surrogate (approximated) objective funtion to fall in ranges of $r_t(\theta)$ outside the interval $[1-\epsilon, 1+\epsilon]$ so as to limit the change in the policy, and 2) to take the minimum of the clipped and unclipped objective so the final objective is a lower (pessimistic) bound.

As an alternative, a penalty that uses KL divergence such as in TRPO and adapts its cost parameter β according to the magnitude of the divergence may be used.

5.2.8 [TODO] ACKTR

Link: `https://arxiv.org/abs/1708.05144`

5.2.9 TD3

Family: Q Policy-gradient

Link: `https://arxiv.org/pdf/1802.09477.pdf`

TL;DR

Twin Delayed DDPG (TD3) [2] is an improvement of DDPG that uses two Q functions and a delayed update of the actor policy in order to avoid overestimation of the Q values.

Problem addressed

How to avoid overestimation on the Q function in DDPG?

Proposed approach

Twin Delayed DDPG boosts DDPG by:

- Using two Q functions Q_{ϕ_1} and Q_{ϕ_2} instead of one, and using the smaller estimation of the two in order to compute the target y. The same target is used to update both.
- Using a random noise for exploration, but clipping it in order to ensure it falls in valid action ranges.

- Learning the policy by gradient ascent such as in DDPG, but only after a delay and by maximizing $J(\theta_a)$ regarding the value Q_{ϕ_1}, (i.e. not using Q_{ϕ_2}).

5.2.10 SAC

Family: Q Policy-gradient

Link: `https://arxiv.org/pdf/1801.01290.pdf`

TL;DR

SAC [4] is an off-policy actor critic algorithm based in maximum entropy reinforcement learning with an stochastic actor (i.e. an actor that learns from sampling actions based on the current policy instead of using past actions from memory) with the aim of maximizing both the reward and entropy (i.e. succeed at the task while acting as randomly as possible).

Problem addressed

Using an off-policy algorithm (to avoid sampling inefficiency), how to achieve stability and convergence in learning?

Prior work

Link: `https://arxiv.org/pdf/1702.08165.pdf`

This work is based in soft Q-learning [**?**], which consists in learning a policy with its reward augmented such that the optimal policy aims to maximize its entropy at each visited state in order to improve exploration and allow to better transfer skills between tasks. The maximum entropy policy is given by

$$\pi^*_{max\_entropy} = \arg\max_{\pi} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha H(\pi(.\mid s_t))] \tag{16}$$

For that, they want to approximate "soft" versions of the Q and V functions, proposing soft Q and V as a generalization of the conventional "hard" Q and V. Soft Q and V are defined by

$$Q^*_{soft}(s_t,a_t) = r_t + \mathbb{E}_{(s_{t+1})\sim\rho_\pi}[\sum_{l+1}^{\infty} \gamma^l (r_{t+1} + \alpha H(\pi^*_{max\_entropy}(.\mid s_{t+l})))] \tag{17}$$

$$V^*_{\text{soft}} = \alpha \log \int_A \exp\{\frac{1}{\alpha} Q^*_{soft}(s_t,a')\} da' \tag{18}$$

By Theorem 1, it is connected soft Q and V with the maximum entropy optimal policy by

$$\pi^*_{maximum\_entropy}(a_t \mid s_t) = \exp\{\frac{1}{\alpha}(Q^*_{soft}(s_t, a_t) - V^*_{soft}(s_t))\} \tag{19}$$

They propose an algorithm with an actor-critic setting based in the soft Q function and made tractable by an approximation and stochastic gradient descent.

Proposed approach

Proposed approach consisting in:

- An off policy formulation to reuse previous experience.
- An actor-critic architecture that learns 1) Q_{ϕ_1} and Q_{ϕ_2} functions as in TD3, 2) a policy π_θ and 3) a value function V_ψ. In this setting, the architecture employs a stochastic actor that sample actions from the learning policy in order to update the policy.
- Using entropy regularized reinfocement learning in order to give the agent an extra reward proportional to the entropy of the policy at each time step, given by $\alpha H(\pi(a \mid s_t)) = \mathbb{E}[-log(\pi(a \mid s_t))]$, being α an hyper-parameter that controls such bonus and therefore the stochasticity of the optimal policy.

5.2.11 Ape-X

Family: Q-value / Q policy gradient, distributed Link: `https://arxiv.org/pdf/1803.00933.pdf`

TL;DR

Ape-X [] is an off-policy distributed algorithm that consists in a set of actors that gather experience and calculate their initial priorities, and a learner that updates the priorities and samples experiences and updates its priorities. This method can be used with both Q-value and Q-policy gradient algorithms.

Proposed approach

Extend experience replay to the distributed setting

Deep RL algorithm decomposed into two parts:

1. Acting: multiple actors that interacts with an environment, evaluates a policy implemented as a deep NN, and stores experience in a shared global replay memory. Actors concurrently run on CPUs in order to generate data.
2. Learning: a single learner that samples batches of data from the memory to update its policy parameters. A single learner runs on GPU to sample the most useful experiences from the replay memory.

As the method is off-policy, it can use algorithms such as DQN and its improvements such as dueling networks or double deep Q-networks, or algorithms such as DDPG.

5.2.12 R2D2

Family: Q-value, distributed Link: `https://openreview.net/pdf?id=r1lyTjAqYX`

TL;DR

R2D2 is built over Ape-X and improves it by using an LSTM layer after the convolutional stack, and an n-step double Q-learning, among other improvements.

Proposed approach

Recurrent Replay distributed DQN (R2D2) builds over Ape-X, and applies several modifications:

- It stores fixed length ($m = 80$) sequences of transition tuples (s, a, r), overlapping each of the adjacent sequences by 40 time-steps, and not crossing episode boundaries.
- Using an LSTM layer after a convolutional stack.
- Using n-step double Q-learning, with $n = 5$
- Using an invertible value function instead of clipping rewards.
- Replay prioritization uses a mixture of max and mean absolute n-step TD errors δ_i over the sequence $\nu \max_i \delta_i + (1 - \nu)^{\bar{\delta}}$, using $\nu = \bar{\delta} = 0.9$.
- Using a higher discount factor of $\gamma = 0.997$ and disabled loss of life determining episode end in the Atari benchmark.

Improved by: R2D3 5.2.13

5.2.13 R2D3

Family: Q-value, distributed Link: `https://arxiv.org/pdf/1909.01387.pdf`

TL;DR

R2D3 [8] is an off-policy algorithm that improves R2D2 by including a buffer of expert demonstrations alongside a replay buffer, computing the policy by sampling a mixture of expert demonstrations and experience generated by the agent.

Problem addressed

Tackle the problem of learning from demonstrations in hard exploration sparse-rewarded tasks with partially observable environments and highly variable initial conditions.

Proposed approach

Recurrent Replay Distributed DQN from Demonstrations (R2D3) builds upon R2D2 including a demo replay buffer with expert demostrations of the task to be solved. The sampling proportion of this buffer vs the experience replay buffer is determined by a demo ratio hyper-parameter ρ.

5.2.14 TODOs

https://arxiv.org/abs/1707.01495 https://arxiv.org/pdf/1806.06923.pdf

Policy-gradient

- ACER `https://arxiv.org/pdf/1611.01224.pdf`
- IMPALA `https://arxiv.org/pdf/1802.01561.pdf`
- GAE `https://arxiv.org/pdf/1506.02438.pdf`

Q Policy-gradient

-

Q-value algorithms

- DQN
- Dueling DQN
- Double DQN
- PER
- Rainbow

Other

-

5.3 [TODO] Milestones in RL

2019:

- AlphaStar []
- Capture the Flag []
- Dota 2 []
- Multiagent []

5.4 [TODO] Benchmarks

- OpenAI Gym
- Atari
- DeepMind Control Suite
- DeepMind BSuite
- DeepMind Lab

5.5 [TODO] Hyper-parameter tuning in RL

5.6 Meta RL and Transfer Learning

5.6.1 [TODO] RL^2

Link:

5.6.2 Pretraining

TODO
https://arxiv.org/abs/1612.07307
In [**?**], network is pretrained ... **TODO**

5.6.3 MAML RL

Link: `https://arxiv.org/pdf/1703.03400.pdf`

The goal in few-shot meta-learning in RL is to enable an agent to quickly acquire a policy for a goal or a new test task (e.g. a new environment) using a small amount of experience in the test task.

As in RL the expected reward is normally not differentiable due to unknown dynamics, in MAML [1], it is proposed a meta-learning algorithm with the following main loop:

1. Sample a batch of tasks over a distribution of different tasks $T = \{T_1, \dots T_n\}$.
2. For each sampled task T_i, it samples K trajectories following the policy parametrized with the global parameters θ, in order to compute the gradient $\nabla_\theta J(\pi_\theta)$.
3. Apply gradient ascent to update a new set of weights θ' with a step size α, used to sample H new trajectories. Save parameters θ' until all the tasks of the batch are sampled.
4. For each sampled task in the batch, update global weights θ with gradient ascent over θ and using each respective policy parametrized by θ'.

5.6.4 REPTL

5.6.5 Fine-tuning

5.7 [TODO] Frameworks

- Dopamine
- Tensorflow Agents
- Stable baselines

5.8 [TODO] Other algorithms and interesting papers

5.8.1 HAC

Paper: `https://arxiv.org/pdf/1712.00948.pdf` Blog post: `http://bigai.cs.brown.edu/2019/09/03/hac.html`

Hierarchical Arctor Critic (HAC) [**?**] is an algorithm that jointly learns a hierarchy of policies, by breaking a complex task into a set of subtasks consisting in short sequences of actions.

Authors propose a framework consisting of

1. A hierarchical policy architecture that yields transitions and reward functions at all levels, treating the policies generated by the below levels as optimal.
2. A method for learning multiple levels of policies independently and in parallel.

5.8.2 TODO

TODO
https://openreview.net/pdf?id=r1etN1rtPB

5.9 Good references

- Open AI - Spinning Up as a Deep RL Researcher
- R. Sutton and A. Barto - Reinforcement Learning: An Introduction (2nd Edition) and Python implementation of the exercises.
- A. Irpan - Deep Reinforcement Learning Doesn't Work Yet
- L. Weng: A (Long) Peek into Reinforcement Learning
- Y. Li - Deep Reinforcement Learning: An Overview

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*, March 2017.

[2] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv:1802.09477 [cs, stat]*, October 2018.

[3] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, October 2017. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2016.2582924.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, August 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.8.1735.

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, September 2015.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]*, February 2016.

[8] Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, Gabriel Barth-Maron, Ziyu Wang, Nando de Freitas, and Worlds Team. Making Efficient Use of Demonstrations to Solve Hard Exploration Problems. *arXiv:1909.01387 [cs]*, September 2019.

[9] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *arXiv:1502.05477 [cs]*, February 2015.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, July 2017.