Let's Play Congestion Control*

Extended Abstract†

Maximilian Bachl
Institute of Telecommunications
Vienna, Austria
maximilian.bachl@tuwien.ac.at

Tanja Zseby
Institute of Telecommunications
Vienna, Austria
tanja.zseby@tuwien.ac.at

Joachim Fabini
Institute of Telecommunications
Vienna, Austria
joachim.fabini@tuwien.ac.at

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings.[1]

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Put some comma-separated keywords here

ACM Reference Format:
Maximilian Bachl, Tanja Zseby, and Joachim Fabini. 2017. Let's Play Congestion Control. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97).* ACM, New York, NY, USA, 3 pages.
https://doi.org/10.475/123_4

1 PARTIAL ACTION ACTOR CRITIC LEARNING

Partial Action Actor Critic Learning (PAL) is a modification of the Asynchronous Advantage Actor Critic algorithm proposed by Mnih et al. [1]. It consists of two Artificial Neural Networks (ANNs), the Actor Network and the Value Network (the Critic part in the abbreviation stands for the Value Network). Given a state, the Actor Networks outputs what it deems to be the optimum action to perform in that certain state. The Value Network estimates what long-term reward can be expected in this state. So an action is considered good if it achieved a long-term reward that is higher than the long-term reward expected by the Value Network and it is considered bad if the reward was lower than expected. The long-term reward is implemented as a moving average of future rewards. So if a high reward can be achieved right now this is more favorable than if it can be achieved in the future. However it can also be beneficial to get a low reward now and instead get a very large one in the future.

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document
[1]This is an abstract footnote

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

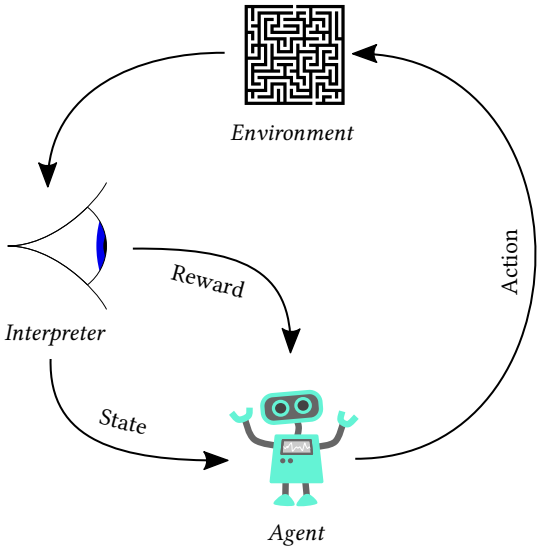

Figure 1: The classical reinforcement learning approach.[a]

[a]adapted from https://upload.wikimedia.org/wikipedia/commons/1/1b/Reinforcement_learning_diagram.svg

1.1 Algorithm

In Algorithm 1 we show the code that runs in one of the learning threads. As proposed in literature, we use several concurrent thread to improve training performance. A major difference to previous approaches to reinforcement learning is the fact that we allow more actions to be taken before the previous actions' rewards have been received. In classical reinforcement learning, an action is always followed by a reward and a reward is always followed by a action. In our proposed concept, however, it is possible to take new actions while the previous action hasn't received its reward yet.

Another major difference in PAL is that one action generates a number of partial actions (≥ 1) (see Figure 2). Each partial action generates a partial reward upon interacting with the environment. Upon receiving a partial reward, the agent determines the current state and triggers a new action. When all partial rewards of one action were received, the agent combines them to form the reward and updates the value and actor networks (see Algorithm 2).

1.2 Value Network

The value network estimates the expected long-term reward given a state s_t.

Let r_t be the reward that was received at time t. $V(s_t; \theta_v)$ designates the expected mean reward in state s_t given the parameters

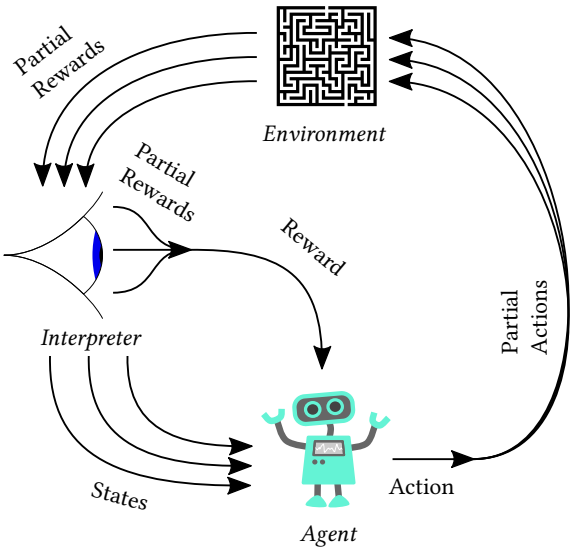

Figure 2: Partial Action Actor Critic Learning: An action consists of zero or more partial actions which trigger partial rewards upon interacting with the environment. Each partial reward updates the state. The value and actor networks are updated upon receiving all partial rewards of one action.[a]

[a]adapted from https://upload.wikimedia.org/wikipedia/commons/1/1b/Reinforcement_learning_diagram.svg

(neural network weights) of the value network θ_v. Then, with γ being the roll-off factor, which designates the influence that future reward has on the moving average, (set to 0.99), we define the average reward as

$$R_t = \left(\left(\sum_{i=0}^{k-1} \gamma^i r_{t+i} \right) + \gamma^k V(s_{t+k}; \theta_\text{v}) \right),$$

where k is upper-bounded by t_max (t_max is a fixed hyperparameter that indicates how many steps should be performed before updating the neural network). So R is simply a moving average of rewards. However, usual moving averages take into account values from the past while this one uses values from the future; it runs reversely.

The loss function, which the value network tries to minimize, is the square of the difference of the actual average reward received and the expected average reward

$$l_{\text{v},t} = (R_t - V(s_t; \theta_\text{v}))^2 .$$

1.3 Actor Network

Let a_t be the action that was taken at time step t, v_t the value that the value network estimated as the future reward given the current state s_t at time step t (v_t is just an abbreviation for $V(s_t; \theta_\text{v})$) and θ_a the parameters (neural network weights) of the actor network. Furthermore, let β be a factor that specifies the importance of the entropy H. π designates the probability density function, which means that $\pi(a_t \mid s_t; \theta_\text{a})$ is the value of the probability density function of taking action a_t in state s_t with the current weights of

Algorithm 1 Partial Action Actor Critic Learning – pseudocode for each learning thread. Only $\theta_{\text{a,g}}$ and $\theta_{\text{v,g}}$ are shared between the threads. They are initialized randomly in the beginning.

loop
  $l_\text{actions} \leftarrow []$
  $l_\text{states} \leftarrow []$
  $l_\text{values} \leftarrow []$
  $l_\text{rewards} \leftarrow []$
  $l_\text{estimatedValues} \leftarrow []$
  $l_\text{snapshots} \leftarrow []$
  $t \leftarrow 0$
  $s_0 \leftarrow$ initialState()
  $\theta_\text{a} \leftarrow \theta_{\text{a,g}}$
  $\theta_\text{v} \leftarrow \theta_{\text{v,g}}$
  repeat
    l_states.append(s_t)
    $l_\text{estimatedValues}$.append($V(s_t; \theta_\text{v})$)
    if t mod $t_\text{max} = 0$ **then**
      $\theta_\text{a} \leftarrow \theta_{\text{a,g}}$
      $\theta_\text{v} \leftarrow \theta_{\text{v,g}}$
      $l_\text{snapshots}$.append($(\theta_\text{a}, \theta_\text{v})$)
    end if
    Sample a_t from the
      Actor Network's probability distribution
    l_actions.append(a_t)
    $l_{\text{partialActions},t} \leftarrow$ partialActions(a_t)
    for all partial actions $a_{\text{p},n,t}$ in $l_{\text{partialActions},t}$ **do**
      Take partial action $a_{\text{p},n,t}$
    end for
    Receive partial reward $r_{\text{p},n,t'}$
      where $t' \leq t$ and $0 \leq n \leq \#(l_{\text{partialActions},t'})$
    if all partial rewards of $a_{t'}$ were received **then**
      $r_{t'} \leftarrow$ combination of all partial rewards $r_{\text{p},n,t'}$
      l_rewards.append($r_{t'}$)
      if $\#(l_\text{rewards}) > t_\text{max}$ or the episode is over **then**
        computeGradients
      end if
    end if
    Generate s_{t+1} using $r_{\text{p}_{t'}}$
    $t \leftarrow t + 1$
  until reaching the end the episode
end loop

the actor network θ_a. Then

$$l_{\text{a},t} = -\log(\pi(a_t \mid s_t; \theta_\text{a}))(R_t - v_t) - \beta H(\pi(s_t; \theta_a))$$

is the actor loss that we try to minimize.

2 CONGESTION CONTROL SPECIFICS

The classical Reinforcement Learning assumes that a reward follows an action and vice-versa (see Figure 1). However, in the case of congestion control, it is desirable to perform a new action without having received a reward for the previous action. For example, imagine that you receive two acknowledgements directly after

Algorithm 2 Partial Action Actor Critic Learning – procedure which computes and applies the gradients.

function computeGradients
  $t_{\text{end}} = \min(t_{\max}, \#(l_{\text{rewards}}))$
  $\theta_{\text{backup}} \leftarrow (\theta_{\text{a}}, \theta_{\text{v}})$
  $\theta_{\text{a}}, \theta_{\text{v}} \leftarrow l_{\text{snapshots}}[0]$
  $R_{i+1} \leftarrow \frac{l_{\text{estimatedValues}}[t_{\text{end}}]}{1-\gamma}$
  for $i \leftarrow t_{\text{end}} - 1, 0$ **do**
    $R_i \leftarrow r_i + \gamma R_{i+1}$
    $a \leftarrow l_{\text{actions}}[i]$
    $s \leftarrow l_{\text{states}}[i]$
    $v \leftarrow l_{\text{values}}[i]$
    $d\theta_{\text{v}} \leftarrow \frac{\partial (R_i - v)}{\partial \theta_v}$
    $d\theta_{\text{v}} \leftarrow -\frac{\partial \log(\pi_W(a-1<W\leq a \mid s;\theta_{\text{a}}))(R_i - v)}{\partial \theta_a} - \frac{\partial \beta H(\pi(s;\theta_a))}{\partial \theta_a}$
    $\theta_{\text{a,g}} \leftarrow \theta_{\text{a,g}} + d\theta_{\text{a}}$
    $\theta_{\text{v,g}} \leftarrow \theta_{\text{v,g}} + d\theta_{\text{v}}$
  end for
  Remove first t_{end} elements from $l_{\text{actions}}, l_{\text{states}}, l_{\text{values}}, l_{\text{rewards}}, l_{\text{estimatedValues}}$
  Remove first element from $l_{\text{snapshots}}$
  $\theta_{\text{a}}, \theta_{\text{v}} = \theta_{\text{backup}}$
end function

each other. For each of these two acknowledgements an action has to be performed but it does not seem feasible for the second action to wait until the first action has received a reward. The Asynchronous Actor Critic framework as described by [1] cannot be applied to congestion control as it assumes that actions and rewards are synchronized and thus we use the proposed Partial Action Actor Critic Learning (see section 1)

In the following a time step t corresponds to the reception of an acknowledgement. The beginning of the flow, before any packet is sent, corresponds to time step 0.

- $\boldsymbol{s_t}$ The state describes the current "congestion state". Various features that indicate congestion are included in it. Each time an acknowledgement is received, the state is updated and the actor network is asked for the next action.
- $\boldsymbol{a_t}$ Based on a given state and the history of previous states, the actor network returns an action a_t, which is an integer (≥ 1) that stands for the congestion window to be used until the next acknowledgement is received.
- $\boldsymbol{r_t}$ The reward is a tuple of at least one reward metrics. These are discussed in more detail in subsection 2.1
- $\boldsymbol{v_t}$ The value is a tuple of the expected average reward estimated by the value network (see subsection 2.1) for each of the reward metrics.

We actually use four different types of reward:

- $\boldsymbol{r_{\text{packet},t}}$ is the number of the packets that the sender sent during time step t and that were not lost (so they were acknowledged at some point by the receiver).
- $\boldsymbol{r_{\text{byte},t}}$ is the sum of the bytes of the packets that the sender sent and that were not lost. Example: During t, three packets were sent with 300, 200, and 1500 bytes each so $r_{\text{byte},t} = 2000$
- $\boldsymbol{r_{\text{delay},t}}$ is the sum of the round trip times of the packets that the sender sent and that were not lost.
- $\boldsymbol{r_{\text{duration},t}}$ is the sum of the time between receiving the last packet and receiving this packet ("inter-receive time") for the packets that the sender sent and that were not lost.

2.1 Value Network

In the case of congestion control, the loss function $l_{\text{v},t}$ of the value network is actually the sum of the square of the difference for each of the four moving averages for each type of reward:

$$\begin{aligned} l_{\text{v},t} = &\left(R_{\text{packet},t} - V_{\text{packet}}(s_t;\theta_{\text{v}})\right)^2 \\ &+ \left(R_{\text{byte},t} - V_{\text{byte}}(s_t;\theta_{\text{v}})\right)^2 \\ &+ \left(R_{\text{delay},t} - V_{\text{delay}}(s_t;\theta_{\text{v}})\right)^2 \\ &+ \left(R_{\text{duration},t} - V_{\text{duration}}(s_t;\theta_{\text{v}})\right)^2 \end{aligned}$$

2.2 Actor Network

The actor network outputs two parameters: The mean of a log-normal distribution $\mu_{\log}$ and its standard deviation $\sigma_{\log}$.

We calculate the means and standard deviation of the underlying normal distribution as

$$\sigma_{\text{normal}} = \sqrt{\log\left(\frac{\sigma_{\log}}{\mu_{\log}} + 1\right)}$$

$$\mu_{\text{normal}} = \log\left(\mu_{\log}\right) - \frac{{\sigma_{\text{normal}}}^2}{2}.$$

Each time an action a_t is requested, a value X is sampled from the current log-normal distribution defined by the parameters μ_{normal} and σ_{normal}:

$$Z \sim \mathcal{N}(\mu_{\text{normal}}, {\sigma_{\text{normal}}}^2)$$
$$X = \exp(Z)$$

We can also equally write

$$X \sim \text{Lognormal}\left(\mu_{\text{normal}}, {\sigma_{\text{normal}}}^2\right).$$

The sampled value X is then discretized by rounding up, which also prevents window sizes smaller than one from occurring:

$$W = \lceil X \rceil$$

a_t is the sampled value W at time t and it is used as the window until the next time step $t+1$ begins (at the reception of the next acknowledgement).

With H being the entropy, the actor network minimizes the loss

$$\begin{aligned} l_{\text{a},t} = &-\log\left(\pi_W\left(a_t - 1 < W \leq a_t \mid s_t;\theta_{\text{a}}\right)\right) \\ &\left(\left(\frac{R_{\text{byte},t}}{R_{\text{duration},t}} - \frac{v_{\text{byte},t}}{v_{\text{duration},t}}\right) - \left(\frac{R_{\text{delay},t}}{R_{\text{packet},t}} - \frac{v_{\text{delay},t}}{v_{\text{packet},t}}\right)\right) \\ &+ \beta H\left(\pi\left(s_t;\theta_a\right)\right). \end{aligned}$$

REFERENCES

[1] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.