Experience-driven Networking: A Deep Reinforcement Learning based Approach (Infocom 2018)

Siddartha Devic

The University of Texas at Dallas

sid.devic@utd.edu

March 12, 2019

Overview

1. Introduction and Problem Overview
   - Existing Solutions and Previous Work
2. Reinforcement Learning
   - Q-learning
   - Continuous Control with DDPG
3. Problem Statement
4. DRL for TE
   - Implementation
   - Results

Introduction and Problem Overview

- Complicated and dynamic communication networks $\rightarrow$ need for better prediction models
- Emerging technology (SDNs) support experience/data approach
  - Validate the reinforcement learning approach
- **Problem**: Given a set of network flows with source and destination nodes, find a solution to forward the data traffic with the objective of maximizing a utility function.
  - OSPF
  - Valient Load Balancing (VLB, evenly distribute traffic)

Existing Solutions and Previous Work

- OSPF, VLB obviously suboptimal
- **Queuing Theory**: Sometimes used, but relies on heavy assumptions which may not hold true in complex networks (i.e. packet arrivals are Poisson)
- **Network Utility Maximization**: Resource allocation by solving an optimization problem
  - Usually assumes some key things are given (user demands, link usages)
- First to apply Deep RL for model-free control in communication networks (claim)

Reinforcement Learning

- Actor/agent interacting with an environment
- Feedback, unsupervised

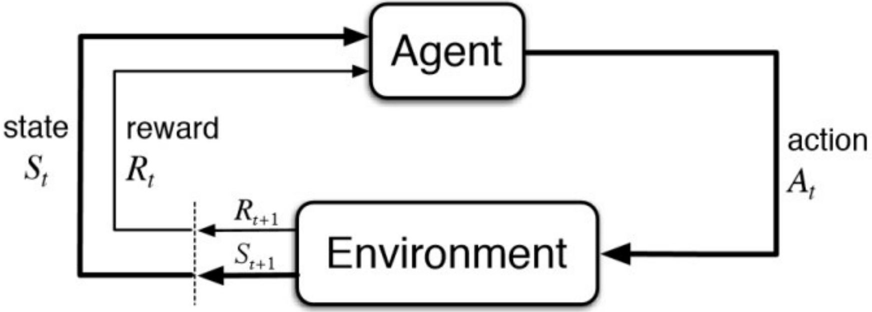
Goal

At each decision epoch t, we observe a state s_t and take an action a_t. Find a policy π which maximizes the total discounted reward

$$R = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$$

$s.t. \gamma \in [0, 1]$ and $r(s_t, a_t)$ is the reward function. $\pi(s) = a$ is the decision of the agent at state s, and maps to some action a.

Reinforcement Learning



Q-Learning

Q-Value: Expected reward for taking action a_t in state s_t, and thereafter following policy π

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

Where R is the reward function.

1. "Off-policy": We always update Q-values based on the assumption that we take the greedy step in the next state.
2. Simple greedy policy: $\pi(s_t) = \arg\max_{a_t \in A} Q(s_t, a_t)$
3. Update according to gradients of loss function

Training a Q-Network

Traditional Q-Learning

Traditional Q-Learning is table based, we keep a current Q-value for every (s_t, a_t) pair. Furthermore, we update these Q-values based on the off-policy algorithm.

- Instead, we can take a DL-based approach
- Train a NN to approximate the Q-values given by a table
- $L(\theta^Q) = \mathbb{E}\left[y_t - Q(s_t, a_t|\theta^Q)\right]$, where $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta^\pi)|\theta^Q)$
- The input to the neural network is s_t, the output is a $|a|$-length vector of Q-values.

Continuous Control

Not applicable for continuous control due to $|a|$

Deep Deterministic Policy Gradient (DDPG)

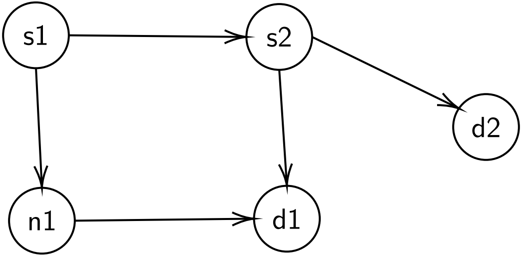
Separate actor and critic network. Actor $\pi(s_t|\theta^{\pi})$ outputs action for a given state, and critic $Q(s_t, a_t|\theta^{Q})$ can both be implemented using neural networks. Furthermore, gradients can be computed using chain rule.

Other technicalities for better stability during training (target/eval network, replay memory, etc.)

Problem Statement

1. Communication session 3-tuple $k = (s_k, d_k, P_k)$ where s_k is source node, d_k is destination node, P_k is a set of directed paths

Problem Statement



$$k_1 = (s_1, d_1, \{\{s_1, s_2, d_1\}, \{s_1, n_1, d_1\}\}$$
$$k_2 = (s_2, d_2, \{s_2, d_2\})$$

Problem Statement

1. Communication session 3-tuple $k = (s_k, d_k, P_k)$ where s_k is source node, d_k is destination node, P_k is a set of directed paths
2. Specify $f_{k,j}$, the load through the jth path of P_k in k
3. Then we take path $j \in P_k$ with probability $w_{k,j} = \frac{f_{k,j}}{\sum_{j=1}^{|P_k|} f_{k,j}}$, or the split ratio.

Evaluation

- Maximize $\sum_{k=1}^{K} U_\alpha(x)$, where $U_\alpha(x) = \frac{x^{1-\alpha}}{1-\alpha}$ and α is a trade-off between fairness and efficiency ($\alpha = 1$).
- Throughput and delay: $U(x_k, z_k) = U_{\alpha_1}(x_k) - \sigma U_{\alpha_2}(z_k)$
- Traffic engineering problem: Maximize $\sum_{k=1}^{K} U_\alpha(x_k, z_k)$

DRL for TE

Agent Observations

- $s = [(x_1, z_1), ..., (x_k, z_k), ..., (x_K, z_K)]$,
- vector of split ratios $a = [w_{1,1}, ..., w_{k,j}, ..., w_{K,|Pk|}]$,
- reward $r = \sum_{k=1}^{K} U_\alpha(x_k, z_k)$

DRL for TE

Q-learning *not applicable* for continuous control due to $|a|$

Deep Deterministic Policy Gradient (DDPG)

Separate actor and critic network. Actor $\pi(s_t|\theta^\pi)$ outputs action for a given state, and critic $Q(s_t, a_t|\theta^Q)$ can both be implemented using neural networks. Furthermore, gradients can be computed using chain rule.

DDPG: Poor performance due to naive exploration method and uniform experience replay sampling

Algorithm 1: DRL-TE

1: Randomly initialize critic network $Q(\cdot)$ and actor network $\pi(\cdot)$ with weights $\boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^\pi$ respectively;

2: Initialize target networks $Q'(\cdot)$ and $\pi'(\cdot)$ with weights $\boldsymbol{\theta}^{Q'} := \boldsymbol{\theta}^Q$, $\boldsymbol{\theta}^{\pi'} := \boldsymbol{\theta}^\pi$;

3: Initialize prioritized replay buffer $\mathbf{B}$ and $p_1 := 1$;

/**Online Learning**/

4: Receive the initial observed state $\mathbf{s}_1$;

/**Decision Epoch**/

5: **for** t = 1 **to** T **do**

6: Apply the TE-aware exploration method to obtain $\mathbf{a}_t$;

7: Execute action $\mathbf{a}_t$ and observe the reward r_t;

8: Store transition sample $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ into $\mathbf{B}$ with maximal priority $p_t = \max_{j<t} p_j$;

9: /**Prioritized Transition Sampling**/

10: **for** i = 1 **to** N **do**

11: Sample a transition $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})$ from $\mathbf{B}$ where $i \sim P(i) := p_i^{\beta_0} / \sum_j p_j^{\beta_0}$;

12: Compute important-sampling weight: $\omega_i := (|\mathbf{B}| \cdot P(i))^{-\beta_1} / \max_j \omega_j$;

13: Compute target value for critic network: $Q(\cdot)$ $y_i := r_i + \gamma \cdot Q'(\mathbf{s}_{i+1}, \pi'(\mathbf{s}_{i+1}))$;

14: Compute TD-error: $\delta_i := y_i - Q(\mathbf{s}_i, \mathbf{a}_i)$;

15: Compute gradient: $\nabla_{\boldsymbol{\theta}^\pi} J_i := \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\pi(\mathbf{s}_i)} \cdot \nabla_{\boldsymbol{\theta}^\pi} \pi(\mathbf{s})|_{\mathbf{s}=\mathbf{s}_i}$;

16: Update the transition priority: $p_i := \varphi \cdot (|\delta_i| + \xi) + (1 - \varphi) \cdot \overline{|\nabla_{\mathbf{a}} Q|}$;

17: Accumulate weight-change for critic network: $Q(\cdot)$ $\Delta_{\boldsymbol{\theta}^Q} := \Delta_{\boldsymbol{\theta}^Q} + \omega_i \cdot \delta_i \cdot \nabla_{\boldsymbol{\theta}^Q} Q(\mathbf{s}_i, \mathbf{a}_i)$;

18: Accumulate weight-change for actor network: $\pi(\cdot)$ $\Delta_{\boldsymbol{\theta}^\pi} := \Delta_{\boldsymbol{\theta}^\pi} + \omega_i \cdot \nabla_{\boldsymbol{\theta}^\pi} J_i$;

19: **end for**

20: /**Network Updating**/

21: Update the weights of critic network: $Q(\cdot)$ $\boldsymbol{\theta}^Q := \boldsymbol{\theta}^Q + \eta^Q \cdot \Delta_{\boldsymbol{\theta}^Q}$, reset $\Delta_{\boldsymbol{\theta}^Q} := 0$;

22: Update the weights of actor network: $\pi(\cdot)$ $\boldsymbol{\theta}^\pi := \boldsymbol{\theta}^\pi + \eta^\pi \cdot \Delta_{\boldsymbol{\theta}^\pi}$, reset $\Delta_{\boldsymbol{\theta}^\pi} := 0$;

23: Update the weights of the corresponding target networks:
$\boldsymbol{\theta}^{Q'} := \tau \boldsymbol{\theta}^Q + (1 - \tau) \boldsymbol{\theta}^{Q'}$;
$\boldsymbol{\theta}^{\pi'} := \tau \boldsymbol{\theta}^\pi + (1 - \tau) \boldsymbol{\theta}^{\pi'}$;

24: **end for**

Novel Contributions

1. Choosing a random action during exploration: $a = a + \epsilon$ or $a = a_{base} + \epsilon$, where a_{base} is computed to be a "good" base action using programming.
2. Prioritized replay buffer for continuous action space (line 16)

NUM-TE:

$$\max_{<x_k, f_{k,j}>} \sum_k U_\alpha(x_k) \tag{6a}$$

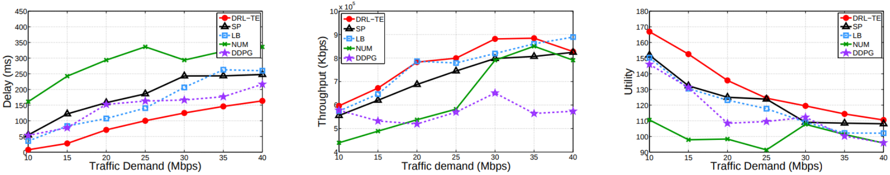
subject to:

$$\sum_{k=1}^{K} \sum_{\mathbf{p}_j \in \mathbf{P}_k : e \in \mathbf{p}} f_{k,j} \leq C_e, \forall\ e \in \mathbf{E}; \tag{6b}$$

$$x_k \leq B_k,\ k \in \{1, \cdots, K\}; \tag{6c}$$

$$\sum_{j=1}^{|\mathbf{P}_k|} f_{k,j} = x_k,\ k \in \{1, \cdots, K\}. \tag{6d}$$

Results

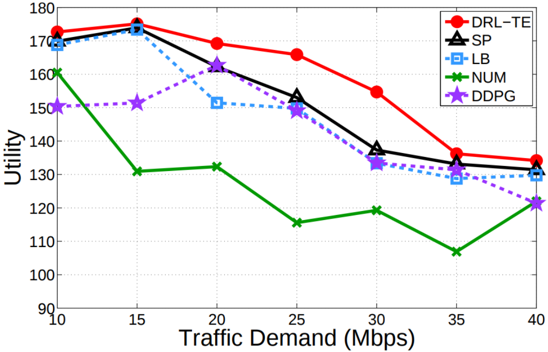


(a) End-to-end delay (b) End-to-end throughput (c) Network utility

Fig. 1: Performance of all the methods over the NSFNET topology

- SP: Shortest path
- LB: Load balancing equal allocation among candidate paths
- NUM: Programming problem

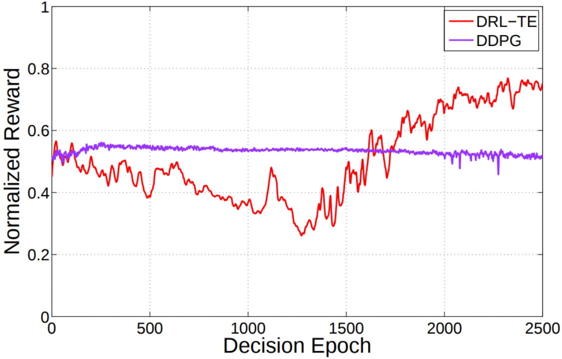
Results on Random Topology



(c) Network utility

- SP: Shortest path
- LB: Load balancing equal allocation among candidate paths
- NUM: Programming problem

Results (Training)



(c) Random topology

- DDPG seems to converge to a local minimum and stay (due to exploration)
- Note: Only test on three topologies