Dynamic resource allocation problems in communication networks:

Machine Learning for resource allocation

Alexandre Reiffers-Masson

Equipe Maths&Net, IMT Atlantique, CS department
LabSTICC (UMR CNRS 6285)

June 27, 2024

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Lab-STICC

Context

A simple optimization problem

Reinforcement Learning

Realistic telecommunication networks

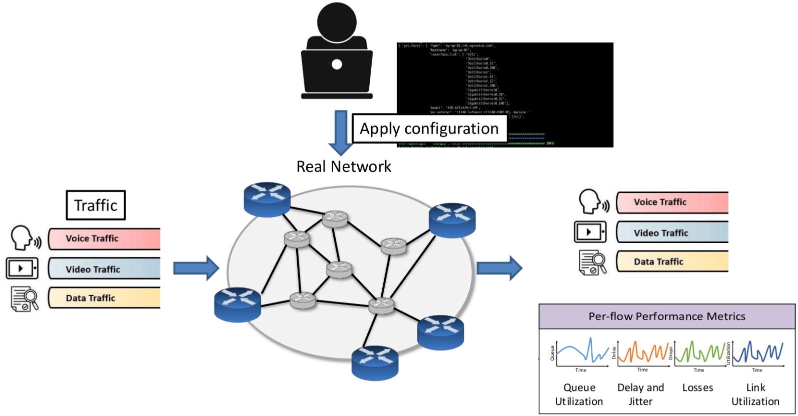

Figure: Taken from "Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks"

Issue with a realistic networking problem

- Simple models do not predict correctly the **traffic**.

Issue with a realistic networking problem

- Simple models do not predict correctly the **traffic**.
- The network is controlled through a given set of configurations. Most of the time the software used is **proprietary**, and therefore, we cannot control as much as we want.

Issue with a realistic networking problem

- Simple models do not predict correctly the **traffic**.
- The network is controlled through a given set of configurations. Most of the time the software used is **proprietary**, and therefore, we cannot control as much as we want.
- The size of the queues is not known and the routing paths are not exact, etc...

Traffic pattern

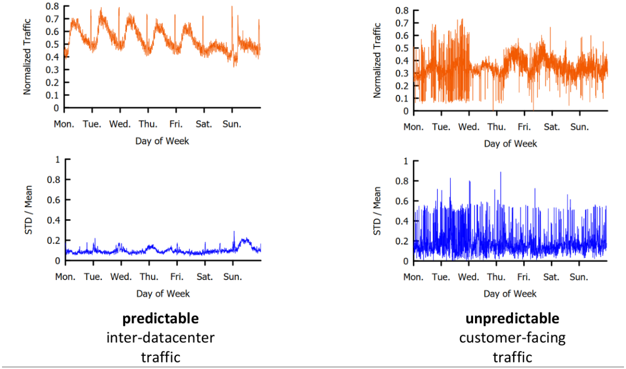

Figure: Taken from "DOTE: Rethinking (Predictive) WAN Traffic Engineering"

Control: Configuration[1]

- Topology, Link Capacity

[1] Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks.

Control: Configuration[1]

- Topology, Link Capacity
- Routing:

[1] Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks.

Control: Configuration[1]

- Topology, Link Capacity
- Routing:
  1. Overlay: SRv6, MPLS...
  2. Underlay: OSPF, BGP...

[1] Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks.

Control: Configuration[1]

- Topology, Link Capacity
- Routing:
  1. Overlay: SRv6, MPLS...
  2. Underlay: OSPF, BGP...
- Scheduling Policy (arbitrary)

[1] Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks.

Control: Configuration[1]

- Topology, Link Capacity
- Routing:
  1. Overlay: SRv6, MPLS...
  2. Underlay: OSPF, BGP...
- Scheduling Policy (arbitrary)
  1. Queue Length
  2. Policy
  3. Hierarchy of policies

[1] Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks.

Control: Configuration[1]

- Topology, Link Capacity
- Routing:
  1. Overlay: SRv6, MPLS...
  2. Underlay: OSPF, BGP...
- Scheduling Policy (arbitrary)
  1. Queue Length
  2. Policy
  3. Hierarchy of policies
- ECMP, LAG, etc

[1] Graph Neural Networking challenge 2023: Building a Network Digital Twin using data from real networks.

Optimisation

The network is modeled as a capacitated graph $G = (V, E, c)$, where the function $c()$ assigns a capacity to each edge.

Tunnels: Each source vertex s communicates with each destination vertex t via a set of network paths, or "tunnels" P_{st}. P_{st} can be interpreted as the routing matrix associated with the couple source/destination (s, t).

Optimisation

The network is modeled as a capacitated graph $G = (V, E, c)$, where the function $c()$ assigns a capacity to each edge.

Tunnels: Each source vertex s communicates with each destination vertex t via a set of network paths, or "tunnels" P_{st}. P_{st} can be interpreted as the routing matrix associated with the couple source/destination (s, t).

Traffic demands: A demand matrix (DM) D is an $|V| \times |V|$ a matrix whose (i, j)-th entry $D_{i,j}$ specifies the traffic demand between source i and destination j.

Optimisation

The network is modeled as a capacitated graph $G = (V, E, c)$, where the function $c()$ assigns a capacity to each edge.

Tunnels: Each source vertex s communicates with each destination vertex t via a set of network paths, or "tunnels" P_{st}. P_{st} can be interpreted as the routing matrix associated with the couple source/destination (s, t).

Traffic demands: A demand matrix (DM) D is an $|V| \times |V|$ a matrix whose (i, j)-th entry $D_{i,j}$ specifies the traffic demand between source i and destination j.

Configurations: A given a network graph and demand matrix, a *configuration* specifies for each source vertex s and destination vertex t how the $D_{s,t}$ traffic from s to t is split across the tunnels in P_{st}.

Objective: minimizing maximum-link utilization.

Objective: minimizing maximum-link utilization.

$$\min_{x \geq 0} \quad \max_{e} \frac{\sum_{s,t} \sum_{p \in P_{st}, e \in p} D_{s,t} x_p}{c_e} \tag{1}$$

$$s.t. \qquad \sum_{p \in P_{st}} x_p = 1, \ \forall (s,t). \tag{2}$$

Objective: minimizing maximum-link utilization.

$$\min_{x \geq 0} \quad \max_e \frac{\sum_{s,t} \sum_{p \in P_{st}, e \in p} D_{s,t} x_p}{c_e} \tag{1}$$

$$s.t. \qquad \sum_{p \in P_{st}} x_p = 1, \ \forall (s,t). \tag{2}$$

This problem is a convex optimization problem (actually an LP problem).

Objective: minimizing maximum-link utilization.

$$\min_{x\geq 0} \quad \max_{e} \frac{\sum_{s,t}\sum_{p\in P_{st},e\in p} D_{s,t}x_p}{c_e} \tag{1}$$

$$s.t. \quad \sum_{p\in P_{st}} x_p = 1,\ \forall (s,t). \tag{2}$$

This problem is a convex optimization problem (actually an LP problem).

What can we do priori knowledge of the traffic demands?

General Traffic Model

The demand matrix D_t is generated according to an **unknown** H-Markov process with transition probabilities such that:

$$\mathbb{P}(D_t|D_{t-1},\ldots,D_{t-H}) = \mathbb{P}(D_t|D_{t-1},...,D_1). \qquad (3)$$

We assume that the Markov chain has reached it's *stationary regime.*

Approach 1: Demand-Prediction-Based

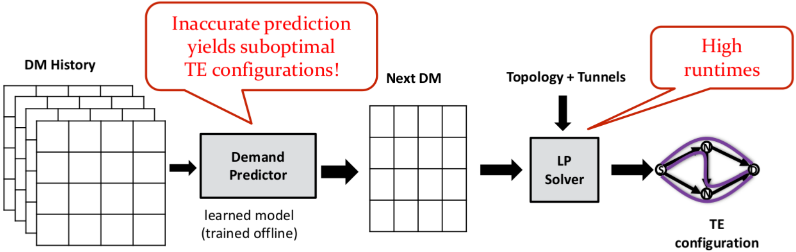

Figure: From "DOTE: Rethinking (Predictive) WAN Traffic Engineering"

Approach 1: Demand-Prediction-Based

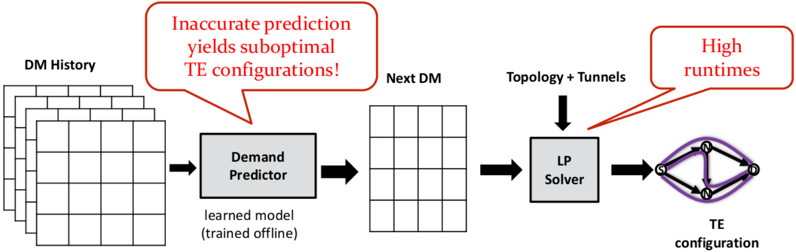

Figure: From "DOTE: Rethinking (Predictive) WAN Traffic Engineering"

- Advantages: Can be reuse for other tasks. In case of a problem it is easy to identity where the problem is coming from.

Approach 1: Demand-Prediction-Based

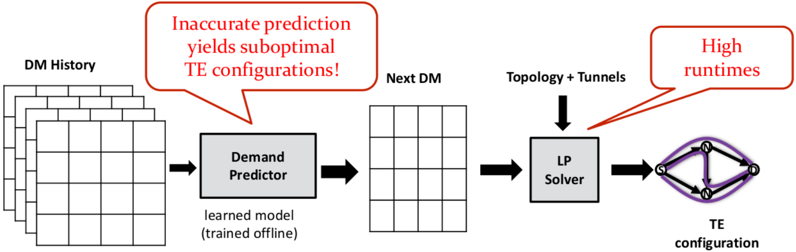

Figure: From "DOTE: Rethinking (Predictive) WAN Traffic Engineering"

- Advantages: Can be reuse for other tasks. In case of a problem it is easy to identity where the problem is coming from.
- Disadvantage: The demand predictor is not tuned for optimising Configuration.

Approach 2: Direct Optimisation

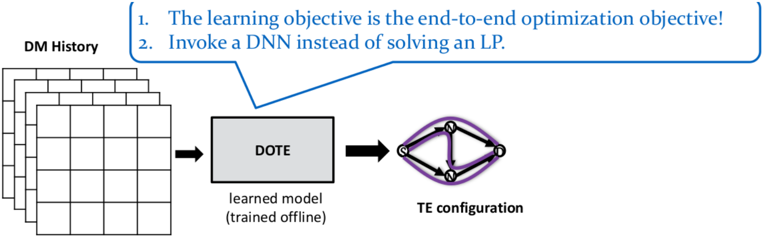

Figure: From "DOTE: Rethinking (Predictive) WAN Traffic Engineering"

- Advantage: Optimal for TE.

Approach 2: Direct Optimisation

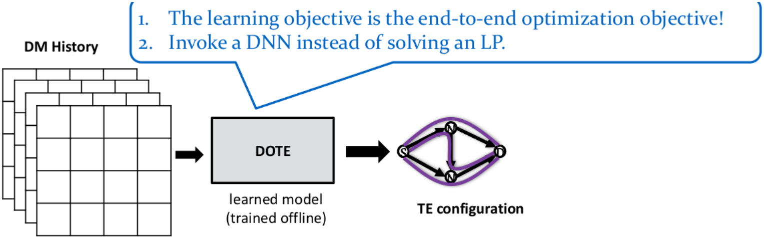

Figure: From "DOTE: Rethinking (Predictive) WAN Traffic Engineering"

- Advantage: Optimal for TE.
- Disadvantage: Cannot be used for other tasks.

Algorithm

DOTE

- **Input:** Observe $D(0)$ and the capacity c.
- **Set** $\theta := \theta_0$;
- **For** $t = 0, 2, \ldots,$ **do:**
  1. **Predict** the allocation $\pi(D_{t-1}, \ldots, D_{t-H}; \theta)$
  2. **Observe** the traffic matrix D_t for which the allocation has been done. Compute
  3. **Compute** the gradient (automatic differentiation) of :

  $$f(\theta) := \max \frac{\sum_{s,t} \sum_{p \in P_{st}, e \in p} D_{s,t} \pi(D_t, \ldots, D_{t-1-H}; \theta)}{c_e}.$$

  4. **Update** the parameter θ as follow:

  $$\theta = \theta - \alpha \nabla_\theta f(\theta).$$

Definition: Random Neural Networks

- p is a probability distribution on ω and $\phi : \mathcal{X} \times \Omega \to \mathbb{R}$ is a function such that $\sup_{x,\theta} |\phi(x, \theta)| \leq 1$.

Definition: Random Neural Networks

- p is a probability distribution on ω and $\phi : \mathcal{X} \times \Omega \to \mathbb{R}$ is a function such that $\sup_{x,\theta} |\phi(x, \theta)| \leq 1$.
- Let us define:

Definition: Random Neural Networks

- p is a probability distribution on ω and $\phi : \mathcal{X} \times \Omega \to \mathbb{R}$ is a function such that $\sup_{x,\theta} |\phi(x,\theta)| \leq 1$.
- Let us define:

$$\begin{aligned} \mathcal{F}_p &:= \left\{ f(x) = \int_\Omega \alpha(\theta)\phi(x,\theta)d\theta \,\middle|\, |\alpha(\theta)| \leq Cp(\theta) \right\} \\ \mathcal{F}_\theta &:= \left\{ f(x) = \sum_{k=1}^{K} \alpha_k \phi(x,\theta_k) \,\middle|\, |\alpha_k| \leq \frac{C}{K} \right\} \end{aligned}$$

Definition: Random Neural Networks

- p is a probability distribution on ω and $\phi : \mathcal{X} \times \Omega \to \mathbb{R}$ is a function such that $\sup_{x,\theta} |\phi(x,\theta)| \leq 1$.
- Let us define:

$$\begin{aligned}
\mathcal{F}_p &:= \left\{ f(x) = \int_\Omega \alpha(\theta)\phi(x,\theta)d\theta \,\middle|\, |\alpha(\theta)| \leq Cp(\theta) \right\} \\
\mathcal{F}_\theta &:= \left\{ f(x) = \sum_{k=1}^{K} \alpha_k \phi(x,\theta_k) \,\middle|\, |\alpha_k| \leq \frac{C}{K} \right\}
\end{aligned}$$

- Note that $\mathcal{F}_p$ consists of functions whose weights $\alpha(\theta)$ decays more rapidly than the given sampling distribution $p(\theta)$.

Performance

Let f be a function from $\mathcal{F}_p$. If μ is a probability measure on $\mathcal{X}$, $\theta_1, ..., \theta_K$ are drawn iid from p, then for all $\delta > 0$, there exist with a probability $1 - \delta$ a function $\hat{f} \in \mathcal{F}_\theta$ such that :

$$||f - \hat{f}||_{2,\mu} \leq \frac{C}{\sqrt{K}} \left(1 + \sqrt{2 \log \frac{1}{\delta}} \right),$$

with $||f - g||_{2,\mu}^2 = \int_{\mathcal{X}} (f - g)^2 d\mu$.

Numerical Illustrations

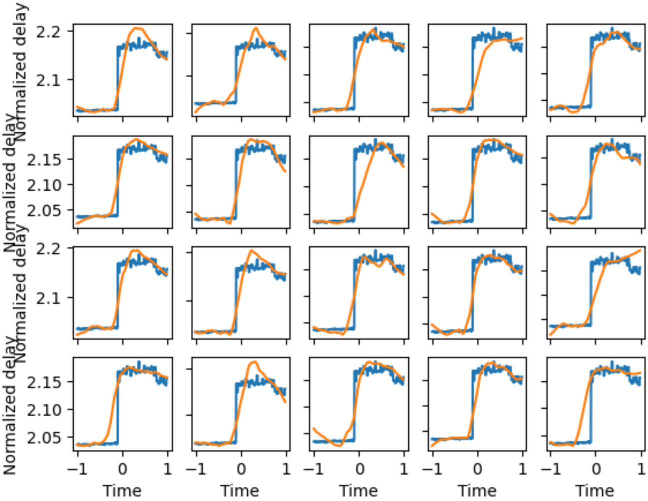

Figure: Random number of samplings: 20, $K = 20$.

Open problems

- We don't know the capacity on each link. How can we compute the gradient?
- How to transfer a learned network to another set-up?
- How to manage when the network is changing of states over time (queues, failures, etc.)?

Open problems

- We don't know the capacity on each link. How can we compute the gradient? **Sol: One point gradient estimator**
- How to transfer a learned network to another set-up? **Sol: understand the notion of scaling**. Scaling to larger networks often entails more aspects beyond the topology size. In particular, there are two main properties that we can observe as networks become larger:
    1. higher link capacities, as core links of the network typically aggregate more traffic,
    2. different flow-level delay distributions, as end-to-end paths are larger and they can traverse links with higher capacities.
- How to manage when the network is changing of states over time (queues, failures, etc.)? **Sol: Deep RL**

Value function

The ‘value function’ $V : S \to \mathbb{R}$ defined as

$$V(x) = \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{m=0}^{+\infty} \gamma^m r(X_m, U_m) \mid \ X_0 = x \right], \ x \in S$$

Theorem (Dynamic programming equation)

The value function satisfies the following dynamic programming equation:

$$V(x) = \max_{u} [r(x,u) + \gamma \sum_{y} p(y \mid x, u) V(y)], \ \forall x \in S. \tag{4}$$

Fixed point theorem

Definition

Let (X, d) be a metric space. Then a map $T : X \to X$ is called a contraction mapping on X if there exists $q \in [0, 1)$ such that :

$$d(T(x), T(y)) \leq d(x, y), \; \forall x, y \in X.$$

Theorem (Banach fixed-point theorem.)

Let (X, d) be a complete metric space with a contraction mapping $T : X \to X$. Then T admits a unique fixed-point x^* in X (i.e. $T(x^*) = x^*$). Furthermore, x^* can be found as follows: start with an arbitrary element $x_0 \in X$ and define a sequence $(x_n)_{n \in \mathbb{N}}$ by $x_n = T(x_{n-1})$ for $n \geq 1$. Then $\lim_{n \to \infty} x_n = x^*$.

Bellman optimality operator property

Let us define the Bellman optimality operator denoted by :

$$[B^* v]_x = \max_u [r(x,u) + \gamma \sum_y p(y \mid x,u) v(y)],\ \forall x.$$

Theorem

The operator B^* are contractions for the sup-norm:

$$\|B^* v - B^* w\|_\infty \leq \gamma \|v - w\|_\infty.$$

‘Value Iteration’ algorithm

The Banach fixed-point theorem gives us our first algorithm:

$$V_{n+1}(x) = \max_u [r(x,u) + \gamma \sum_y p(y \mid x,u) V_n(y)],\ \forall x \in S.$$

To obtain the optimal policy you simple need to take π_{n+1} as follows:

$$\pi_{n+1}(x) = \arg\max_u [r(x,u) + \gamma \sum_y p(y \mid x,u) V_n(y)],\ \forall x \in S. \quad (5)$$

when n goes to infinity, π_{n+1} will converge to the optimal policy.

Examples of other dynamic programming equation

- Average reward:

$$V(x) = \max_u [r(x,u) - \beta + \sum_y p(y \mid x,u) V(y)], \ \forall x \in S. \quad (6)$$

- Finite horizon:

$$\begin{aligned} V(x,t) &= \max_u [r(x,u) + \sum_y p(y \mid x,u) V(y,t+1)], \ \forall x \in S, \\ V(x,T) &= 0, \ \forall x \in S. \end{aligned}$$

Q-function

Let us recall the definition of the dynamic programming equation:

$$V(x) = \max_u [r(x,u) + \gamma \sum_y p(y \mid x,u) V(y)], \ \forall x \in S. \quad (7)$$

Now we define Q-values as the expression in square brackets in the expression below:

$$Q(x,u) = r(x,u) + \gamma \sum_y p(y \mid x,u) V(y), \ \forall x \in S, \ \forall u \in \mathcal{U}. \quad (8)$$

Observation 1: If for all x and u, $Q(x,u)$ is known then the optimal control at state x is found by simply minimizing $Q(x,\cdot)$. No need to know the reward or the transition probability.

Q-function

Let us recall the definition of the dynamic programming equation:

$$V(x) = \max_u [r(x,u) + \gamma \sum_y p(y \mid x,u) V(y)], \; \forall x \in S. \tag{9}$$

Now we define Q-values as the expression in square brackets in the expression below:

$$Q(x,u) = r(x,u) + \gamma \sum_y p(y \mid x,u) V(y), \; \forall x \in S, \; \forall u \in \mathcal{U}. \tag{10}$$

Observation 2: Note that $V(x) = \max_u Q(x,u)$ which implies that Q-values satisfies their own dynamic programming equation:

$$Q(x,u) = r(x,u) + \gamma \sum_y p(y \mid x,u) \max_v Q(x,v), \; \forall x \in S, \; \forall u \in \mathcal{U}. \tag{11}$$

Q-value iteration

Again, the Banach fixed-point theorem gives us our first algorithm:

$$Q_{n+1}(x,u) = r(x,u) + \gamma \sum_y p(y \mid x,u) \max_v Q_n(y,v),\ \forall x \in S,\ \forall u \in \mathcal{U}.$$

To obtain the optimal policy you simple need to take π_{n+1} as follows:

$$\pi_{n+1}(x) = \arg\max_u [Q_{n+1}(x,u)],\ \forall x \in S. \tag{12}$$

when n goes to infinity, π_{n+1} will converge to the optimal policy.
Remark: What we have gained at the expense of increased dimensionality is that the nonlinearity is now inside the conditional expectation w.r.t. the transition probability function.

Stochastic approximation

A stochastic approximation scheme has usually the following form:

$$w_{k+1} = \Gamma\left[w_k + \alpha_k(h(w_k) + M_{k+1})\right],\ w_0 = w^0.$$

The classical ingredients are:

- The previous guess;
- a small-update for which its amplitude is controlled by the step-size. The update is composed of:
  1. A deterministic function +
  2. A noise with zero mean.
- $\Gamma[\cdot]$ is a projection function to ensure a good behavior of the iterative scheme.

Convergence theorem: The O.D.E approach

Theorem

Under the right assumptions, the behavior of $\lim_{k\to+\infty} w_k$, where

$$\boxed{w_{k+1} = w_k + \alpha_k(h(w_k) + M_{k+1}),\ w_0 = w^0,}$$

is the same as the behavior of $\lim_{t\to+\infty} w(t)$, where $w(t)$ is the solution of the ordinary differential equations:

$$\boxed{\dot{w}(t) = h(w),\ w(0) = w^0.}$$

Why is it true? An intuition

Recall that the standard 'Euler scheme' for numerically approximation a trajectory of the o.d.e

$$\dot{x}(t) = h(x(t))$$

would be

$$x_{n+1} = x_n + ah(x_n),$$

where $a > 0$ is a small step.
The stochastic approximation iteration differs from this in two aspects:

1. Possible replacement of the constant time step a by a time-varying α_k (for instance $\frac{1}{k+1}$).
2. The presence of the noise M_{k+1}.

This is why the stochastic approximation scheme is nothing more than a noisy discretization of the o.d.e.

Why this theorem/ the O.D.E. approach

- There are two approaches to the theoretical analysis of such algorithms:
  1. Probabilistic approach, popular among statisticians,
  2. O.D.E. approach, more popular among engineers.
- The O.D.E. approach can serve as useful recipe for concocting new algorithms: any convergent o.d.e. is a potential source of a stochastic approximation algorithm that converge to the desired one.

Q-learning ODE

A possible O.D.E associated to the Q-learning algorithm would be:

$$\begin{aligned}\dot{Q}(x,u) &= r(x,u) + \gamma \sum_y p(y \mid x,u) \max_v Q(y,v) - Q(x,u) \\ &= r(x,u) + \mathbb{E}_{\xi(x,u)\sim p(\cdot|x,u)}[\max_v Q_n(\xi(x,u),v)] - Q(x,u)\end{aligned}$$

If this O.D.E is converging, to which point is it converging?

Estimator

- Note that

$$\sum_y p(y \mid x, u) \max_v Q_n(y, v) = \mathbb{E}_{\xi(x,u) \sim p(\cdot|x,u)}[\max_v Q_n(\xi(x, u), v)]$$

- From the previous observation, can you deduce a be a good estimator of $p(y \mid x, u) \max_v Q_n(y, v)$?
- **Answer:**

Estimator

- Note that

$$\sum_y p(y \mid x, u) \max_v Q_n(y, v) = \mathbb{E}_{\xi(x,u)\sim p(\cdot \mid x,u)}[\max_v Q_n(\xi(x, u), v)]$$

- From the previous observation, can you deduce a be a good estimator of $p(y \mid x, u) \max_v Q_n(y, v)$?
- **Answer:**

$$\max_v Q_n(\xi(x, u), v), \text{ with } \xi(x, u) \sim p(\cdot \mid x, u).$$

Q-learning

From the previous observations, we can deduce **Q-learning** algorithm:

Q-learning

- **Input:** Initial value $Q_0(x,u)$ and $\alpha \in (0,1)$.
- **For** $t = 0, 2, \ldots,$ **do:**
  - **For** (x,u), **sample** $\xi(x,u) \sim p(\cdot \mid x,u)$ and **update**:

$$
\begin{aligned}
Q_{n+1}(x,u) \quad &= \quad Q_n(x,u)(1-\alpha) \\
&\quad\; +\alpha\left[r(x,u) + \max_v Q_n(\xi(x,u),v)\right]
\end{aligned}
$$

Exercice: Can you prove that this algorithm is a stochastic approximation (basically you need to find $h(w_n)$ and M_{n+1})?

Online Q-learning

Online Q-learning

- **Input:** Initial value $Q_0(x,u)$, $\alpha \in (0,1)$ and a policy π.
- **For** $t = 0, 2, \ldots,$ **do:**
  1. **Play** $U_n = \pi(X_n)$,
  2. **Sample** $\xi(X_{n+1}) \sim p(\cdot \mid X_n, U_n)$
  3. **Update**:

$$
\begin{aligned}
Q_{n+1}(x,u) \quad = \quad & Q_n(x,u) \\
& + I\{X_n = x, U_n = u\}\alpha[r(x,u) \\
& + \max_v Q_n(X_{n+1}, v) - Q_n(x,u)]
\end{aligned}
$$

Assumption for Convergence

Frequent update/sufficient exploration: For this algorithm to converge, we need that π is such that:

$$\liminf_{n \to +\infty} \frac{1}{n} \sum_{m=0}^{n-1} I\{X_m = x,\ U_m = u\} > 0 \; a.s \text{ a.s. } \forall x, u$$

Why do we need this assumption?

We need to ensure that every states/actions are visited.

Epsilon-greedy

The *epsilon-greedy* policy is a simple method for balancing exploration and exploitation in reinforcement learning. The policy is defined as follows:

- With probability ϵ, select an action uniformly at random (exploration).
- With probability $1 - \epsilon$, select the action that has the highest estimated current Q-value (exploitation).

Deep Q-learning

- *Observation:* Q-learning scheme inherits the 'curse of dimensionality' of MDPs
- **Solution:** Replace Q by a parametrized family $(x, u, \theta) \to Q(x, u; \theta)$ (a neural network).
- **Challenge**: What could be the objective to learn the 'optimal' approximation $Q(\,\cdot\,,\,\cdot\,; \theta^*)$?

Empirical Bellman Error

Empirical Bellman Error

One natural *performance measure* is:

$$E(\theta) := \mathbb{E}_{(X,U,X')}[(r(X,U) + \gamma \max_v Q(X',v;\theta) - Q(X,U;\theta))^2].$$

Remark: The expectation is with respect to the stationary law of (X_n, U_n).

Optimisation algorithm

- How to optimize the Empirical Bellman Error?

Optimisation algorithm

- How to optimize the Empirical Bellman Error?
- Idea 1: Use a stochastic gradient algorithm:

$$\begin{aligned}\theta_{n+1} &= \theta_n - \alpha \nabla_\theta \left[r(X_n, U_n) \right. \\ &\quad \left. + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2 \right]\end{aligned}$$

Optimisation algorithm

- How to optimize the Empirical Bellman Error?
- Idea 1: Use a stochastic gradient algorithm:

$$\begin{aligned}\theta_{n+1} &= \theta_n - \alpha \nabla_\theta \Big[r(X_n, U_n) \\ &\quad + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2 \Big]\end{aligned}$$

- Due to the max operator it is hard compute the gradient in θ of:

$$(r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2$$

Optimisation algorithm

- How to optimize the Empirical Bellman Error?
- Idea 1: Use a stochastic gradient algorithm:

$$
\begin{aligned}
\theta_{n+1} &= \theta_n - \alpha \nabla_\theta \Big[r(X_n, U_n) \\
&\quad + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2 \Big]
\end{aligned}
$$

- Due to the max operator it is hard compute the gradient in θ of:

$$
(r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2
$$

- Idea 2: Approximate the function above by using a semi-gradient:

Optimisation algorithm

- How to optimize the Empirical Bellman Error?
- Idea 1: Use a stochastic gradient algorithm:

$$\begin{aligned}\theta_{n+1} &= \theta_n - \alpha \nabla_\theta \left[r(X_n, U_n) \right. \\ &\qquad \left. + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2 \right]\end{aligned}$$

- Due to the max operator it is hard compute the gradient in θ of:

$$(r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))^2$$

- Idea 2: Approximate the function above by using a semi-gradient:

$$(r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n)) \nabla_\theta Q(X_n, U_n; \theta_n).$$

Deep Q-learning

Online Deep Q-learning

- **Input:** Initial value $Q_0(x, u)$, $\alpha \in (0, 1)$ and a policy π.
- **For** $t = 0, 2, \ldots,$ **do:**
  1. **Play** $U_n = \pi(X_n)$,
  2. **Sample** $\xi(X_{n+1}) \sim p(\cdot \mid X_n, U_n)$
  3. **Update**:

$$
\begin{aligned}
\theta_{n+1} \quad = \quad & \theta_n - \alpha \, (r(X_n, U_n) \\
& + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n)\Big) \\
& \times \nabla_\theta Q(X_n, U_n; \theta_n)
\end{aligned}
$$

Experience replay

Remark: The previous implementation of DQN tends to overfit on the current data. One solution is to store the transitions from past perform the following update:

Online Deep Q-learning with experience replay

$$
\begin{aligned}
\theta_{n+1} \quad = \quad & \theta_n - \frac{\alpha}{M} \sum_{m=0}^{M} \Big(r(X_n^{(m)}, U_n^{(m)}) \\
& + \gamma \max_v Q(X_{n+1}^{(m)}, v; \theta_n) - Q(X_n^{(m)}, U_n^{(m)}; \theta_n) \Big) \\
& \times \nabla_\theta Q(X_n^{(m)}, U_n^{(m)}; \theta_n),
\end{aligned}
$$

where $(X_n^{(m)}, U_n^{(m)})$, are samples from past.

Bibliography

- A simple way to play with a realistic set-up for optimization of configuration of a network: https://bnn.upc.edu/challenge/gnnet2022/. Especially the following link: https://github.com/BNN-UPC/GNNetworkingChallenge/blob/2022_DataCentricAI/quickstart.ipynb
- DOTE: Rethinking (Predictive) WAN Traffic Engineering : https://www.usenix.org/conference/nsdi23/presentation/perry

Bibliography

- François-Lavet, Vincent, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. "An introduction to deep reinforcement learning." Foundations and Trends® in Machine Learning 11, no. 3-4 (2018): 219-354.
- Avrachenkov, Konstantin E., Vivek S. Borkar, Hars P. Dolhare, and Kishor Patil. "Full gradient DQN reinforcement learning: A provably convergent scheme." In Modern Trends in Controlled Stochastic Processes: Theory and Applications, V. III, pp. 192-220. Cham: Springer International Publishing, 2021.