Payments System Design Using Reinforcement Learning: A Progress Report

A. Desai[1] H. Du[1] R. Garratt[2] F. Rivadeneyra[1]

[1]Bank of Canada
[2]University of California Santa Barbara

16th Payment and Settlement System Simulation Seminar
31 August 2018

Motivation

How should new HVPS be designed?

- Performing counterfactual exercises with realistic and reliable estimates of a structural model

How are new HVPS designed?

- Empirical evaluation of different design options using historical data
- But historical data were generated under different rules and behaviour of participants would likely change
- How important are the behavioural changes?

Objective

- First: bring machine learning (ML) techniques to study payments systems
  - Approximate the current behavioural rules of participants in the Canadian LVTS
- Future: help design new system by investigating the implied tradeoffs (delay and liquidity) of alternative designs

Concepts: Why Machine Learning?

- Simulation: process of replication of known outcomes given input data, environment and rules of agents (ABM)

- ML: estimation of rules given input data and environment
  - Supervised: when output and inputs are known
  - Reinforcement learning (RL): agents learn by observing the results of their interaction with the environment

Task: estimate (learn) the policy rules of agents

Concepts: Why Machine Learning?

- Unsupervised ML: when only the inputs are available, used to find or interpret the structure or topology of inputs. Not our interest today
- Deep Neural Network (DNN): technique for non-linear estimation
- Deep learning: estimation of policy rules using DNN

Reinforcement Learning

Payments systems are well suited for RL methods because rules of environment are fixed and known

- Outcomes can change with the actions (inputs) of agents
- Agents interact with the environment and learn by observing the results of those actions

What we need:

1. Reward function: provide agents with signals of the value of actions taken
2. A value function approximator (like DNN)
3. Environment: RTGS simulator

Literature

Payments systems theory:

- Bech & Garratt (2003) liquidity management game and equilibria

Agent-based methods:

- Arciero et al. (2009) explore responses of agents to shocks in RTGS
- Galbiati & Soramäki (2011) model agents choosing initial liquidity to satisfy payment demands

Reinforcement learning:

- Bellman (1957) dynamic programming is a direct precursor of RL
- Bertsekas & Tsitsiklis (1996) techniques for approximate DP
- Sutton & Barto (1998) early techniques of reinforcement learning
- Efron & Hastie (2016) estimation of DNN
- Lots of open source work: OpenAI, Deep Mind, etc.

Plan of the talk

1. Model: liquidity management problem as a dynamic programming problem
2. Implementation of RL algorithm
   - 2.1 The reward function
   - 2.2 Q learning algorithm
   - 2.3 Deep neural network
   - 2.4 Computational architecture
3. Future work

Model: individual liquidity management problem

- Day divided into subperiods: $t = 0, 1, .., T$
- ℓ_t: liquidity at t
- $d > 0$: cost of delay per dollar per subperiod t
- $\{p_t\}$: set of new payments to be processed **in** subperiod t
- $\{s_t\}$: set of payments sent **in** subperiod t
- $\{p_t^{-1}\}$: set of payments queued (internally) **up to** subperiod t
- $\{r_t\}$: set of payments received in subperiod t

Model

Objective: minimize cost of delay s.t. liquidity constraint and all payments sent by the end of the day

$$\begin{aligned} V(\{s_t\};\ell_t) &= \max_{\{s_t\}} \left[-\sum \left\{ p_{t+1}^{-1} \right\} \times d + \beta V(\{s_{t+1}\};\ell_{t+1}) \right] \\ \text{s.t.} \quad & \ell_t = \ell_{t-1} - \sum\{s_t\} + \sum\{r_t\} \geq 0 \\ & \left\{ p_t^{-1} \right\} \subseteq \{p_{t-1}\} \cup \left\{ p_{t-1}^{-1} \right\} \setminus \{s_{t-1}\} \end{aligned}$$

and

$$\begin{aligned} & \ell_T = \ell_{T-1} - \sum \left\{ p_T^{-1} \right\} - \sum\{p_T\} + \sum\{r_T\} \geq 0 \\ & s_T \subseteq \{p_T\} \cup \left\{ p_T^{-1} \right\} \end{aligned}$$

Model

Remarks:

- Stochastic version can be similarly formulated
- In this formulation payments are indivisible and non-interchangeable
- Integer problem and curse of dimensionality make this a hard problem to solve by guess-and-verify, envelope theorem or backward induction
- Solution: approximate Dynamic Programming (Bertsekas & Tsitsiklis, 1996) using Deep Neural Networks

Model: variants

Basic problem can be extended

- Liquidity management problem with choice of initial liquidity (ℓ_0)
- Collateral management problem choosing subsequent collateral apportioning (c_t)
- Liquidity management game (Bech & Garratt, 2003): simultaneously solve for the policy function of each participant

RL Implementation

RL agent objective: learn how much payment value, P, to send at any point in time (given current liquidity, payments queued, expected demands, average sent payments, expected and received incoming payments)

To implement our RL agent we need:

1. Reward function: the normative statement of the value of actions taken by the agent
2. Q learning: method to record value of action-state pairs
3. DNN: method to approximate the Q function
4. Simulator: calculates the outcomes of the environment given certain actions providing the reward

Reward Function

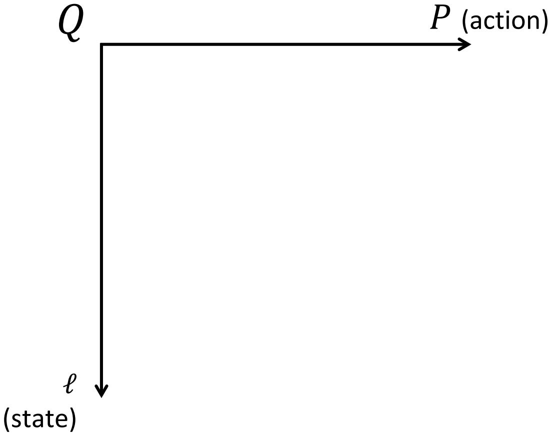
- Reward function provides the returns from taking certain actions given the state

$$\rho_t = -\sum \left\{ p_t^{-1} \right\} \times d - c_t$$

- Choice is P which determines the amount of delay $\left\{ p_t^{-1} \right\}$
- c_t is the cost of collateral (if needed)
- Variant: add received payments, $\{r_t\}$

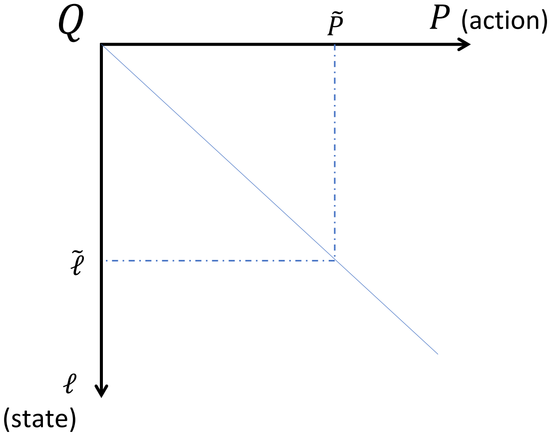
Q learning algorithm

- Q function: logs attained rewards of action P given state ℓ
- Q learning: the procedure to estimate this function



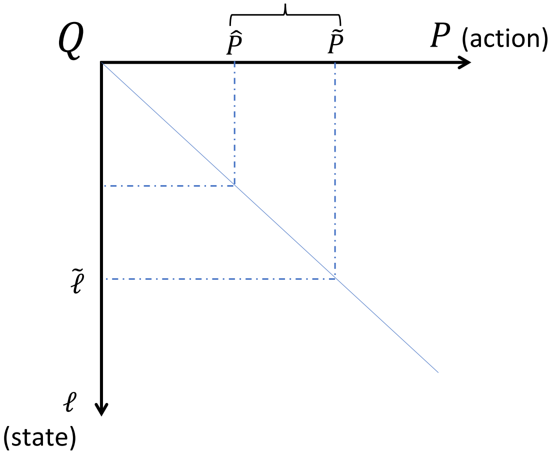
Q learning algorithm

- Q function: logs attained rewards of action P given state ℓ
- Q learning: the procedure to estimate this function



Q learning algorithm

- Q function: logs attained rewards of action P given state ℓ
- Q learning: the procedure to estimate this function



Q learning algorithm

1. Initialize Q-function with zero value entries
2. $\ell_t \leftarrow$ initial state ℓ_0 where ℓ_t defined as liquidity at time t
3. **While** not converged **do**
4. $\hat{P}_t =$ $\begin{cases} \text{sample randomly } P_t \in \text{FIFO}(\{s_t\}) \text{ with probability } \epsilon \\ \text{argmax}_{P_t \in \text{FIFO}(\{s_t\})} Q(\ell_t, P_t) \text{ with probability } 1-\epsilon \end{cases}$
5. Perform action $(\{s_t\}) = \text{FIFO}^{-1}(\hat{P}_t)$
6. $\rho_t \leftarrow \rho_{t+1}$ new reward from environment
7. $\ell_t \leftarrow \ell_{t+1}$ new state from environment
8. $Q(\ell_t, P_t) = R_t + \gamma Q(\ell_{t+1}, P_{t+1})$

FIFO algorithm

- FIFO algorithm is a solution to the integer problem
- Agent chooses the **value** P and the FIFO algorithm returns the set $\{s_t\}$ such that $\sum\{s_t\} \leq P$
- Other algorithms (FIFO bypass, sort, etc) are interesting and could help learning (future work)

Deep Neural Network

A feed forward neural network is a nonlinear system with

- one input layer of x_j as predictors (liquidity, payment demands, ...)
- one or more hidden layers of "neurons" a_ℓ
- and the output layer o

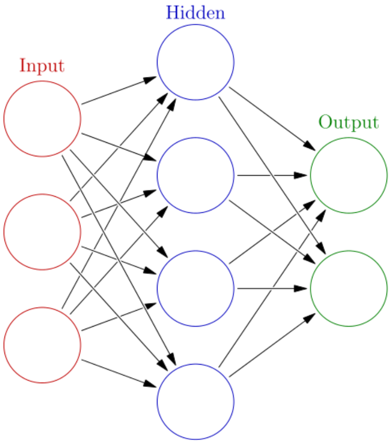
Neurons use inputs in the following way:

$$a_\ell = g\left(w_{\ell 0}^{(1)} + \sum_{j=1}^{K_1} w_{\ell j}^{(1)} x_j\right)$$

and feed into the output

$$P = h\left(w_{\ell 0}^{(2)} + \sum_{\ell=1}^{K_2} w_{\ell}^{(2)} a_\ell\right)$$

Deep Neural Network

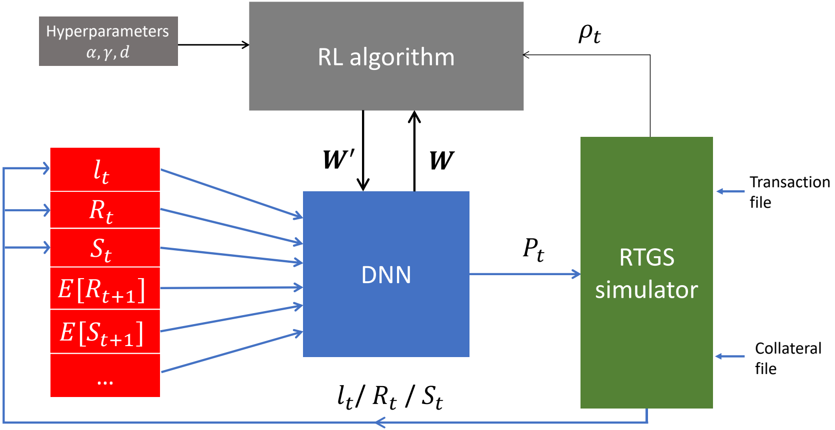


The inputs x_j feed into the neurons with some weights $w_{\ell j}^{(k)}$ and to the output value P

Objective is to estimate the vector $\mathbf{W}$ usually via gradient descent

Deep refers when the number of hidden layers, k, is "large"

RL Computational Infrastructure



Each day is an "epoch." The DNN is re-estimated until some convergence criteria of the Q function is attained

Intuition and Expected Results

Agent uses and estimates:

- Current liquidity, ℓ, expected demands, $E[p_{t+1}]$, payments queued, p_t^{-1}, expected and received incoming payments, $E[r_{t+1}]$ and r_t, and past rewards ρ,

To maximize cumulative rewards by:

- Choosing the value of payments to be sent in that period (action P_t)
- DNN is updated ($\mathbf{W}'$) using the variation in rewards

Expected Results

- Policy function should be a buffer: $\ell - P > 0$
- Policy function conditions on the deviation of typical payment demands and received payments

Learning and Challenges

Learning:

- When choosing P, solution is a choice of a liquidity buffer. How to deal with payment priorities?
- Idea: two-step process. First choose P and then optimize within $\{s\}$

Challenges:

- Estimation of the DNN (e.g. # of layers), choice of hyperparameters (e.g. ε) and convergence criteria
- How to handle the impact of an agent's choices on the ability of others to send payments at the time as observed in the data: add collateral

Future work

- Finish estimation of individual liquidity management problem
- Game version: simultaneously estimate individual policy rules of multiple agents
- Optimize over hyperparameters and initial liquidity
- Alternatives to FIFO to learn from the structure of payments

Future work

How to use in payments system design:

- With estimated rules do transfer learning: use the trained DNN and rules under a new environment and re-estimate outcomes
- For example set simulator (environment) to:

  i) reduce reward to induce throughput guidelines (vary the delay cost)

  ii) introduce new LSMs

Advertisement

5th BoC - PayCan Fall Payments Workshop
Ottawa ON, September 20-21, 2018

Contact:
Francisco Rivadeneyra
riva@bankofcanada.ca