Deep Learning for Solving Linear Integral Equations Associated with Markov Chains

Yanlin Qu
Columbia Business School
qu.yanlin@columbia.edu

Jose Blanchet
Stanford University
jose.blanchet@stanford.edu

Peter Glynn
Stanford University
glynn@stanford.edu

Abstract

Linear integral equations are central to the analysis of general state-space Markov chains; solving them leads to Lyapunov functions (drift equation), the central limit theorem (Poisson's equation), and stationary distributions (global balance equation). This paper develops a simple, simulator-based procedure that solves such equations by training a neural network to minimize a squared residual estimated via an unbiased "double-sample" loss of one-step transitions. The method does not require access to stationary distributions or long trajectories, and it extends to non-compact state spaces through a first-return decomposition that localizes training. A queueing case study demonstrates accuracy and robustness relative to Monte Carlo baselines.

1 Introduction

In operations research and stochastic modeling, *linear integral equations* are central to the analysis of general state-space Markov chains; solving them leads to Lyapunov functions (drift equation), the central limit theorem (Poisson's equation), and stationary distributions (global balance equation). In this paper, we show how deep learning can be used to numerically solve such equations.

Given a Markov chain $X = (X_n : n \geq 0)$ on $\mathcal{S} \subseteq \mathbb{R}^d$ with one-step transition kernel $P = (P(x, dy) : x, y \in \mathcal{S})$, we leverage deep learning to globally compute expectation functionals of the form

$$u^*(x) = \mathbb{E}_x \left[\sum_{k=0}^{T} \exp\left(-\sum_{j=0}^{k-1} \beta(X_j) \right) r(X_k) \right], \tag{1}$$

for $r \geq 0$ and $x \in C$, where $T = \inf\{n \geq 0 : X_n \in C^c\}$. Expected hitting times, exit probabilities, and infinite-horizon expected discounted rewards (obtained by making C^c the empty set) are all special cases of (1). By conditioning on the first transition of the Markov chain (i.e., using *first-transition analysis* (FTA)), these functionals are characterized by linear integral equations, and our algorithm trains a neural network to minimize the associated residual.

As a canonical special case, this viewpoint recovers the Lyapunov framework commonly used for stability. The typical Lyapunov construction involves a nonnegative function $v = (v(x) : x \in \mathcal{S})$ satisfying

$$(Pv)(x) \leq v(x) - 1, \tag{2}$$

for $x \in A^c$ and a suitable set A, where $(Ph)(x) = \int_{\mathcal{S}} h(y)P(x, dy)$ for a generic function h; see Meyn and Tweedie (2012). The function v then provides an upper bound on $\mathbb{E}_x T_A$, where $\mathbb{E}_x$ denotes expectation on the path-space of X conditioned on $X_0 = x$, and $T_A = \inf\{n \geq 0 : X_n \in A\}$. In fact, our deep learning algorithm computes the minimal nonnegative function satisfying (2), namely $v^*(x) = \mathbb{E}_x T_A$, which satisfies (2) with equality.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MLxOR: Mathematical Foundations and Operational Integration of Machine Learning for Uncertainty-Aware Decision-Making.

We train neural networks to satisfy these equations by minimizing the integrated squared residual error with respect to the network parameters using stochastic gradient descent (SGD). To obtain an unbiased gradient estimator, we use the double-sampling trick that utilizes two independent and identically distributed (iid) copies of the first transition. We have used this idea in related work of ours on computing rates of convergence to equilibrium for Markov chains; see Qu et al. (2024). This idea has also been used to compute splitting probabilities and to predict rare events without simulating long sample paths, thereby achieving great efficiency; see Strahan et al. (2023); Cheng and Weare (2024) and the references therein. In the context of reinforcement learning, the same idea can be used to perform policy evaluation, leading to the *residual gradient algorithm* (RGA); see Baird (1995); Baird and Moore (1998); Sutton and Barto (2018).

Our main contribution in this paper is to expose the use of deep learning to the applied probability community, as a means of solving the linear integral equations that arise from Markov chain models. In the course of explaining these ideas, we describe how deep learning can be applied to compute global approximations to the solutions of FTA equations arising within the general setting of (1). In particular, as explained above, our proposed approach therefore provides a vehicle for deep-learning-based numerical construction of Lyapunov functions. Hence, deep learning can potentially play a role in settling the types of stability questions that often arise in Markov chain modeling applications.

In addition to solving the linear integral equations that arise within the setting of FTA expectations and probabilities, this paper also contributes suitable deep-learning-based algorithms for solving Poisson's equation and estimating stationary distributions. Moreover, we show that the deep learning method can be applied to Markov chains on non-compact state spaces, despite the fact that the theory and practice of deep-learning-based function approximation require compact domains. In summary, with the deep learning method, we can construct Lyapunov functions, solve Poisson's equation, and estimate stationary distributions, for Markov chains on compact or non-compact state spaces. Qu et al. (2025) is an extended version of this paper that discusses, for example, related sample complexity issues.

2 First-transition analysis via deep learning

We start by observing that the expectation $u^* = (u^*(x) : x \in C)$ defined by (1) solves

$$u = g + Hu, \tag{3}$$

where

$$H(x, dy) = e^{-\beta(x)} P(x, dy)$$

for $x, y \in C$ and

$$g(x) = r(x) + \int_{C^c} e^{-\beta(x)} P(x, dy) r(y)$$

for $x \in C$.

Neural networks are flexible function approximators that can be trained to satisfy functional equations such as (3). Given a neural network $\{u_\theta : \theta \in \Theta\}$ (a parametrized family of functions), to choose θ such that u_θ approximately satisfies (3) on C, we minimize the integrated squared residual error

$$l(\theta) = \int_C \left(u_\theta(x) - g(x) - (Hu_\theta)(x)\right)^2 \nu(dx),$$

where ν is a probability measure on C (e.g., the uniform distribution). Let $X_0 \sim \nu$ and $X_1 \sim P(X_0, \cdot)$, where we write $Y \sim \eta$ to denote $P(Y \in \cdot) = \eta(\cdot)$. Then $l(\theta)$ can be written as

$$\mathbb{E}\left[\left(\mathbb{E}\left[u_\theta(X_0) - \Gamma(X_0, X_1) - e^{-\beta(X_0)} u_\theta(X_1) I(X_1 \in C)) \Big| X_0\right]\right)^2\right],$$

where $\Gamma(X_0, X_1) = r(X_0) + e^{-\beta(X_0)} r(X_1) I(X_1 \in C^c)$. Stochastic gradient descent (SGD) $\theta_{t+1} = \theta_t - \alpha_t \hat{l}'(\theta_t)$ can be used to minimize $l(\theta)$, where α_t is the step size and $\hat{l}'(\theta_t)$ is an unbiased estimator of the gradient of $l(\theta)$ at θ_t, i.e., $\mathbb{E}\hat{l}'(\theta_t) = l'(\theta_t)$. To construct such an unbiased estimator, we use the double-sampling trick. Given X_0, let X_1 and X_{-1} be independent samples from $P(X_0, \cdot)$. Then

$$\begin{aligned}\hat{l}'(\theta) = 2 &\left[u_\theta(X_0) - \Gamma(X_0, X_1) - e^{-\beta(X_0)} u_\theta(X_1) I(X_1 \in C)\right] \\ &\cdot \left[u'_\theta(X_0) - e^{-\beta(X_0)} u'_\theta(X_{-1}) I(X_{-1} \in C)\right]\end{aligned}$$

is an unbiased estimator of the gradient of

$$l(\theta) = \mathbb{E}\left[u_\theta(X_0) - \Gamma(X_0, X_1) - e^{-\beta(X_0)}u_\theta(X_1)I(X_1 \in C)\right] \\ \cdot \left[u_\theta(X_0) - \Gamma(X_0, X_{-1}) - e^{-\beta(X_0)}u_\theta(X_{-1})I(X_{-1} \in C)\right].$$

Here, u'_θ is the gradient of the neural network with respect to its parameters, computed via backpropagation. (We note that the expression for $\hat{l}'(\theta)$ is not symmetric in X_1 and X_{-1}. We do not recommend the symmetrized estimator for $\hat{l}'(\theta)$ because it contains both $u'_\theta(X_1)$ and $u'_\theta(X_{-1})$, consuming too much computational effort relative to the typical variance reduction.) With $\hat{l}'(\theta)$ at hand, we can solve (3) by minimizing $l(\theta)$ via SGD (Algorithm 1).

Algorithm 1 FTA-RGA

```
Require: probability measure $\nu$, neural network $\{u_\theta : \theta \in \Theta\}$, initialization $\theta_0$, step size $\alpha$, number
  of iterations $\bar{T}$
  for $t \in \{0, ..., \bar{T} - 1\}$ do
      sample $X_0 \sim \nu$ and $X_1, X_{-1} \overset{\text{iid}}{\sim} P(X_0, \cdot)$
      Compute $\hat{l}'(\theta_t)$ as
          $2\left[u_{\theta_t}(X_0) - \Gamma(X_0, X_1) - e^{-\beta(X_0)}u_{\theta_t}(X_1)I(X_1 \in C)\right]$
          $\cdot \left[u'_{\theta_t}(X_0) - e^{-\beta(X_0)}u'_{\theta_t}(X_{-1})I(X_{-1} \in C)\right]$
      $\theta_{t+1} = \theta_t - \alpha\hat{l}'(\theta_t)$
  end for
  return $u_{\theta_{\bar{T}}}$
```

3 Dealing with non-compact state spaces

When C is non-compact, we can not expect an algorithm that terminates in finite time to globally approximate u^* on C (because to encode the approximation then consumes infinite memory). Even the issue of whether the algorithm can faithfully approximate u^* over a compact set K is not immediately obvious, because u^* on the compact set is influenced by the behavior of the Markov chain over the entire infinite state space.

To approximate u^* on K, instead of (3), we can solve

$$u^*(x) = \mathbb{E}_x\left[\sum_{k=0}^{T\wedge\tau-1} \exp\left(-\sum_{j=0}^{k-1}\beta(X_j)\right) r(X_k)\right] \\ + \mathbb{E}_x\left[I(T > \tau)\cdot \exp\left(-\sum_{j=0}^{\tau-1}\beta(X_j)\right) u^*(X_\tau)\right],$$

where $x \in K$, $a \wedge b = \min(a, b)$, and $\tau = \inf\{n \geq 1 : X_n \in K\}$. Again, SGD can be used to minimize the corresponding $l(\theta)$, and the double-sampling trick can be used to construct the corresponding $\hat{l}(\theta)$. Instead of two iid first transitions, we need two iid sample paths returning to K.

As an example on a non-compact state space, we consider a G/G/2 queue. It is a single waiting line served by two parallel servers, with iid $U[0, 2/0.6]$ interarrival times (arrival rate 0.6) and iid $U[0, 2/0.5]$ service times (service rate 0.5). Customers are served in the order in which they arrive, by the first server to become available. The Kiefer-Wolfowitz workload vector $W = (W^{\min}, W^{\max})$ (Kiefer and Wolfowitz, 1956) records the remaining workload at each server immediately after each arrival, ordered from smallest to largest, providing a Markovian description of the system. To be specific,

$$W_{n+1}^{\min} = \min((W_n^{\min} - A_{n+1})_+ + S_{n+1}, (W_n^{\max} - A_{n+1})_+),$$
$$W_{n+1}^{\max} = \max((W_n^{\min} - A_{n+1})_+ + S_{n+1}, (W_n^{\max} - A_{n+1})_+),$$

where A_{n+1} is the interarrival time while S_{n+1} is the service time.

Since the state space $\mathcal{S} = \{(x_1, x_2) : 0 \le x_1 \le x_2\}$ is non-compact, we compute $u^*(x) = \mathbb{E}_x T_A$ on K where $A = \{(x_1, x_2) : 0 \le x_1 \le x_2 \le 3\}$ and $K = \{(x_1, x_2) : 3 \le x_1 \le x_2 \le 9\}$. We train a single-layer neural network with width $m = 1000$ and sigmoid activation $\sigma(z) = 1/(1 + e^{-z})$. We run $\bar{T} = 10^6$ SGD steps with step size $\alpha_t \equiv 10^{-3}$. To visualize the result, we evaluate the trained neural network $u_{\theta_{\bar{T}}}$ on a mesh grid with spacing 0.2 in K (left plot of Figure 1). To validate the result, we independently estimate $\mathbb{E}_x T_A$ for each x on the same grid using 10000 iid simulation runs (right plot of Figure 1). From Figure 1, we observe that the "minimal" Lyapunov function $u^*(x_1, x_2)$ remains nearly flat when x_1 is small and then smoothly transitions to linear growth as x_1 increases. This may explain why quadratic-then-linear (Huberized) Lyapunov functions work well in the stability analysis of queueing systems; see, e.g., Blanchet and Chen (2020).

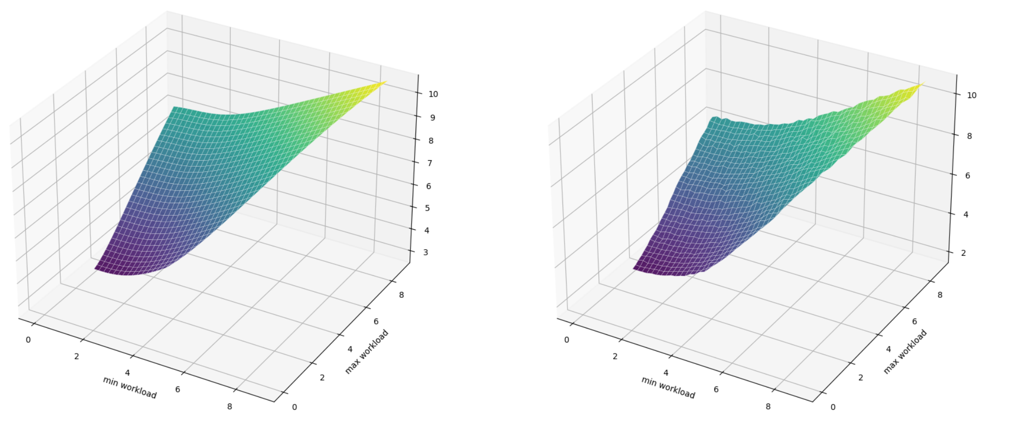

Figure 1: Left: The learned solution of $Pu = u - 1$. Right: The estimated $u^*(x) = \mathbb{E}_x T_A$ on a grid.

4 Solving Poisson's equation

Let X be a Markov chain with stationary distribution π, and suppose r is π-integrable. To solve Poisson's equation

$$u - Pu = r - \pi r,$$

we can eliminate the centering constant $\pi r = \int_{\mathcal{S}} r(y)\pi(dy)$ by applying $I - P$ to both sides, yielding

$$u - 2Pu + P^2 u = r - Pr,$$

where $(P^2 u)(x) = \mathbb{E}_x u(X_2)$. To minimize the corresponding integrated squared residual error, the double-sampling trick requires iid (X_1, X_2) and (X_{-1}, X_{-2}).

5 Estimating stationary distributions

Suppose both P and π have continuous density. Then we can estimate π by solving

$$\pi(y) = \int_{\mathcal{S}} \pi(x)\eta(dx)p(x, y),$$

for $y \in \mathcal{S}$, where η is a reference probability measure. This time the double-sampling trick requires $Z_1, Z_{-1} \overset{\text{iid}}{\sim} \eta$ (and $Y_0 \sim \nu$) to minimize

$$\tilde{l}(\theta) = \mathbb{E}\left[[\pi_\theta(Y_0) - \pi_\theta(Z_1)p(Z_1, Y_0)] [\pi_\theta(Y_0) - \pi_\theta(Z_{-1})p(Z_{-1}, Y_0)] \right].$$

References

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, 1995.

Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems*, 11:968–974, 1998.

Jose Blanchet and Xinyun Chen. Rates of convergence to stationarity for reflected Brownian motion. *Mathematics of Operations Research*, 45(2):660–681, 2020.

Xiaoou Cheng and Jonathan Weare. The surprising efficiency of temporal difference learning for rare event prediction. *Advances in Neural Information Processing Systems*, 37:81257–81286, 2024.

J Kiefer and Jacob Wolfowitz. On the characteristics of the general queueing process, with applications to random walk. *The Annals of Mathematical Statistics*, pages 147–161, 1956.

Sean P Meyn and Richard L Tweedie. *Markov Chains and Stochastic Stability*. Springer Science & Business Media, London, 2012.

Yanlin Qu, Jose Blanchet, and Peter Glynn. Deep learning for computing convergence rates of Markov chains. *Advances in Neural Information Processing Systems*, 37:84777–84798, 2024.

Yanlin Qu, Jose Blanchet, and Peter Glynn. Deep learning for Markov chains: Lyapunov functions, Poisson's equation, and stationary distributions. *arXiv preprint arXiv:2508.16737*, 2025.

John Strahan, Justin Finkel, Aaron R Dinner, and Jonathan Weare. Predicting rare events using neural networks and short-trajectory data. *Journal of Computational Physics*, 488:112152, 2023.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.