# Gaussian Based Non-linear Function Approximation for Reinforcement Learning

SN Computer Science (2021) 2:223
https://doi.org/10.1007/s42979-021-00642-4

ORIGINAL RESEARCH

# Gaussian Based Non-linear Function Approximation for Reinforcement Learning

**Abbas Haider**[1] · **Glenn Hawe**[1] · **Hui Wang**[1] · **Bryan Scotney**[1]

Received: 11 November 2020 / Accepted: 9 April 2021


**Abstract**
Reinforcement learning (RL) problems with continuous states and discrete actions (CSDA) can be found in classic examples such as Cart Pole and Puck World, as well as real world applications such as Market Making. Solutions to CSDA problems typically involve a function approximation (FA) of the mapping from states to actions and can be linear or nonlinear. Linear FAs such as tile-coding (Sutton and Barto in Reinforcement learning, 2nd ed, 2009) suffer from state information loss due to state discretization, whilst non-linear FAs such as DQN (Mnih et al. in Playing atari with deep reinforcement learning, https://arxiv.org/abs/1312.5602, 2013) are practically infeasible in infinitely large state spaces due to their cubic time complexity ($O(n^3)$). In this paper, we propose a novel, general solution to CSDA problems, called Gaussian distribution based non-linear function approximation (GBNLFA). Experimentation on three CSDA RL problems (Cart Pole, Puck World, Market Marking) demonstrates the superiority of GBNLFA over state-of-the-art FAs, namely tile-coding and DQN. In particular, GBNLFA resolves the state information loss problem with linear FAs and provides an asymptotically faster algorithm ($O(n)$) than linear FAs ($O(n^2)$) and neural network based nonlinear FAs ($O(n^3)$).

**Keywords** Function approximation · Reinforcement learning · Gaussian distribution · Probability density function

## Introduction

Function approximation (FA) in reinforcement learning (RL) solves the dimensionality curse problem in continuous state RL tasks. Tabular RL method maintains a table of state-action pairs and the associated action-value or $Q(s, a)$ value, where $s$ and $a$ denotes the state and the action, respectively. In continuous state space, there can be a huge number of state-action pairs. The tabular RL technique is computationally greedy and expensive in continuous state spaces. Hence, using tabular RL method to store the learning experience, becomes practically infeasible in real-life applications. In continuous state space RL problems, FA is the most suitable way of approximating the mapping function ($Q(s, a)$) of state space over the action space. The $Q(s, a)$ function estimates the quality of an action $a$ in state $s$ and the RL agent learns this function during the interaction with the environment.

There are two categories of FAs, namely parametric and non-parametric. Parametric FA involves a function of a fixed number of basis functions or features. Parametric FAs are further categorized into two groups: linear, e.g. tile-codings; non-linear, e.g. Deep Q Network (DQN). The FAs which do not assume the form of underlying function, are known as non-parametric FAs. The number of features in non-parametric FAs can be variable and can not be predicted before hand, these features are derived from the data during training phase. The $Q(s, a)$ function is a linear combination of $n$ number of features ($\phi$) which represents the relevant features of a state. This linear combination of state features, denoted by Eq. (1), is known as parametric linear FA.

$$Q(s, a) = \sum_{i=1}^{n} \phi(s, a)_i * w_i \tag{1}$$

Parametric class of FAs comprises of two main steps: (1) determine the form of RL model prior learning; (2) learn the RL

✉ Abbas Haider
haider-a@ulster.ac.uk

Glenn Hawe
gi.hawe@ulster.ac.uk

Hui Wang
h.wang@ulster.ac.uk

Bryan Scotney
bw.scotney@ulster.ac.uk

[1] School of Computing, Ulster University, Jordanstown, Northern Ireland, UK

Published online: 20 April 2021

model parameters from interactive experience of agent and environment. There are several advantages of parametric methods: (1) they are fast and learn parameters of the RL model quickly; (2) the flexibility to design the model as per the problem complexity; (3) require less experience and can perform well even in a case of imperfect model fit. Few disadvantages of parametric FAs are: (1) they are bounded to a fixed form of model; (2) they may derive a poor fit of the underlying model. Despite the discussed disadvantages, many real world applications have been solved recently using parameteric FAs. Russo et al. [20] and Spooner et al. [22] used tile-codings to determine the efficient elasticity policies for data stream processing systems and to design a RL based market making (MM) agent, respectively. Narvekar and Stone [14] used tile-codings to solve the curriculum Markov Decision Process for continuous state space. Ghiassian et al. [6] reduce learning interference in ReLu gates due to the geometric nature of tile-codings similar to ReLu gates. Li et al. [11] develops a multipath congestion control approach using tile-codings for state aggregation of high dimensional state space. Wang et al. [26] and Oroojlooyjadid et al. [15] both use DQN to direct the scheduling of multi-workflows in multi-agent RL and to design an optimized decision making algorithm for beer game, respectively. Han et al. [8] propose a novel DQN algorithm with dueling architecture for visual object detection problem.

Linear parametric FAs discretize the continuous state space by aggregating the similar states together. These similar states are represented by $n$ number of hand crafted features ($\phi$ in Eq. (1)). Tile-codings, a well-known and widely used practical solution of linear parametric FA, developed by Sutton and Barto [24], belongs to the linear parametric FA class. Tile-codings partitions the continuous state space into a fixed size grid known as tiling. Each tiling contains a fixed number of blocks known as tiles. Similar states are grouped together in the same tile; this way a compact discrete representation of the entire continuous state space is achieved. In non-linear parametric class, a NN is used as the FA method, where state variables are treated as neurons of the input layer. The NN backpropagates the error between the actual and the observed $Q(s, a)$ function values to adjust the weights of input layer neurons.

We propose a novel FA method for continuous state and discrete action (CSDA) RL domain. This proposed method is referred as Gaussian based Non-linear Function Approximation (GBNLFA) throughout the paper. In GBNLFA, each discrete action is represented by a Gaussian distribution with two standard parameters ($\mu$ and $\sum$). The Gaussian distribution fits over *temporal-difference* (TD) error $= \alpha(r + \gamma \max(Q(s', a')) - Q(s, a))$. Each distribution clusters the $Q(s, a)$ values based on the action selected during agent-environment interaction. The purpose of using a Gaussian distribution as a model of learning is the ubiquitousness nature and wide applicability of Gaussians in real-life applications and phenomenon. Moreover, the Gaussian distribution requires only a few hundred samples to fit over the data, hence model generalization from experience to unseen data is fast.

Linear parametric FAs suffer from two problems: (1) they require domain expertise for manual features extraction; (2) they discard lot of crucial state information while state aggregation. Though, the non-linear parametric FAs e.g. fully connected DQN also resolves these two problems, but they suffer from cubic running time complexity ($O(n^3)$, where $n$ is the number of state variables). The main contributions of this paper are: (1) provides a novel FA method based on Gaussian distribution for CSDA RL domain; (2) GBNLFA resolves the state information loss problem; (3) GBNLFA grows linearly $O(n)$ with the dimension of the state-space, and hence is faster than DQN and tile-codings, asymptotically. The empirical results show the significance of GBNLFA method in terms of the scores from three RL problems (Cart Pole, Puck World and MM). Nonetheless, the proposed method uses multiple Gaussian distributions in comparison to the single NN as an approximator.Hence, a group of multiple Gaussians leads to the better generalization (evident via improved scores in empirical analysis) of RL learning experience over unseen states from sampled states.

## Related Work

### Linear Parametric Methods

Basis functions, also termed as features [3], are the inception of linearly parameterized function approximators; Eq. (1) denotes a linear function of basis functions ($\phi$'s) associated with their weights. Lagoudakis and Parr [10] propose the linear combination of state variables directly with a single constant basis function ($\phi = 1$), termed as polynomial basis functions. Another instance of linear FAs involves radial basis functions, where $\phi$ is a normalized Gaussian distribution over the number of states. Sutton [23] proposes TD learning based function approximation method known as TD-FA; then Rummery and Niranjan [19] combined Q learning [28] with TD learning to develop a connectionist Q learning known as TDQ-FA. Watkins and Dayan [29] developed a Q-learning based function approximator known as QL-FA and showed that Q learning converges to optimum action values. Geist et al. [5] developed a parametric FA method termed as Kalman temporal difference (KTD) for deterministic MDPs to derive the value function and a policy.

Discrete action function approximators uses state aggregation to discretize the continuous states by partitioning the state space into disjoint sets. All the similar states are grouped together to form a large aggregate, and hence the total number of aggregates become finite. For each action, the function approximator allocates the same value to all of the similar states in an aggregate. As all subsets are disjoint, at any discrete point in time any one of them is taken

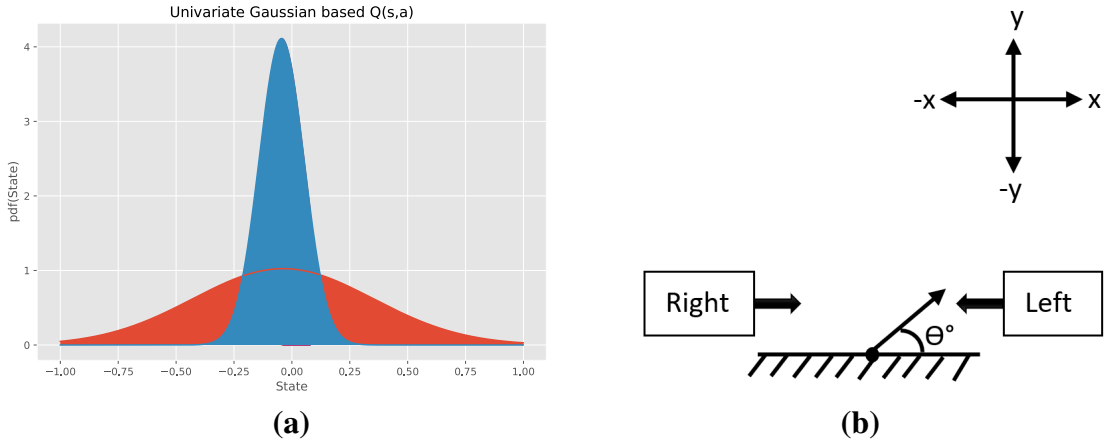


**Fig. 1** **a** Univariate Gaussian based GBNLFA. **b** A Needle attached to a fixed surface can move freely towards both sides (left or right). The angle from the positive x-axis is the state variable. The needle needs to be balanced at 90° by two actions (Left and Right)

into consideration for action, and the others remain inactive. A typical example of a discrete action approximator is tile-codings, developed by Sutton and Barto [24], where state space is partitioned into square-shaped tiles. Davies [4] designed an interpolation algorithm using triangulation of $n$-dimensional state space and Munos and Moore [13] use Kuhn triangulation to generate improved RL policies.

### Non-linear Parametric Methods

Linear parametric FA methods involve linear combination of features or basis functions, which requires domain expertise in selecting potential basis functions. NNs resolve this problem by automatic feature extraction from the input data when state and action spaces are continuous in nature. NNs in RL has been a widely known research area, and the well known RL algorithm, which uses simple neural nets (a shallow network with one hidden layer), is TD backgammon [25]. Riedmiller [17] proposed Neural Fitted Q Iteration for estimating the V function offline through weight updating using the RPROP [18] algorithm. This research route of using NN function approximators in RL is formally known as deep RL, and the most recent ground-breaking applications of deep RL involve ATARI 2600 games.

Researchers of DeepMind developed a technique called Deep Q Networks (DQN) [12], which is an improvement on Neural Fitted Q Iteration. DQN combines deep convolutional neural networks (CNN) with a Q-learning algorithm to play ATARI 2600 games through processing screen pixels data. With the advent of processing technology such as graphical processing units, DQN becomes a popular choice of solving problems using RL and that involve raw input data. Also, DQN is based on large deep CNNs that are responsible for powerful feature extraction and representation. Moreover, DQN uses experience replay to resolve the issues of stable performance due to correlation among states. These features make DQN a powerful solution to real-world problems, and hence is considered as a benchmark non-linear parametric function approximator for comparison. Some of the popular variants of DQN include double DQN [9], dueling DQN [27], prioritized DQN [21] and averaged-DQN [1].

## Proposed Method

Sutton and Barto [24] describe RL as the method of mapping situations to actions by assessing the scalar reward signal. Markov Decision Processes (MDPs) are known as the best way to solve sequential decision making problems including RL problems [16]. From the MDP perspective, an RL framework contains four key components:

- A set of states, $\mathbb{S}$.
- A set of actions, $\mathbb{A}$.
- $P_a(s', s) = P(s'|s, a)$, is the probability of transition from state $s$ to $s'$, when action $a$ is taken.
- $R_a(s', s) = \sum_{i=1}^{n} \gamma^i \cdot R_i$, where $n$ is the total number of time steps, $\gamma$ is the discount factor, and $a$ is the selected action.

### GBNLFA

We propose to use a group of independent Gaussian distributions (Fig. 1a) as FAs for the RL agent. Here, the actions left and right relate to a simple example described below.

These distributions are independent, since each Gaussian represents an action in a discrete action space. The probability density function (pdf) serves as the $Q(s, a)$ of the RL agent. The $Q(s, a)$ estimates the quality of an action $a$ in state $s$ based on the probability density of the corresponding Gaussian distribution. The dimensions of the Gaussian depend on the

dimensionality of the state space. The parameters of the RL model are the parameters of a Gaussian distribution, namely mean ($\mu$) and covariance ($\sum$). These parameters are estimated by the RL agent while interacting with the environment.

To make GBNLFA clearer, we consider a simple CSDA problem, where a needle is fixed to a surface using a pivot, as shown in Fig. 1b. The state space in this problem has one dimension, i.e. angle from the positive $x$-axis. The action space is discrete with two actions, namely left and right. The reward function here is $1/(90-\theta)$, the value of $(90-\theta)$ has to be minimized by maximizing the angle $\theta$. The aim is to maximize the reward function by maximizing angle $\theta$. In Fig. 1a, two Gaussian distributions with different $\mu$ and $\sigma$ are shown, representing two discrete actions. For a particular angle value $\theta$, the two distributions denoting left and right actions have different values of probability density using Eq. (2). As the needle falls towards the right side, a force pushing the needle towards the left is required. Suppose that the Gaussian distribution corresponding to the left action has a higher $Q(s, a)$ (using Eq. (2), as shown in Fig. 1a; the Gaussian corresponding to the left action has a larger area than the Gaussian corresponding to the right action in the range $[\mu, \theta]$. The updated parameters $\mu$ and $\sigma$ represent the GBNLFA based RL model of this simple CSDA problem.

$$Q(s,a) = \frac{1}{\sqrt{2\pi}\sigma_a} \exp\left(\frac{-1}{2\sigma^2}(s-\mu_a)^2\right) \tag{2}$$

The radial basis functions uses Gaussian distribution over the number of states to solve the CSDA RL problem. In more practical and complex CSDA RL problems, the state space is multi-dimensional in nature. A multivariate Gaussian based GBNLFA addresses this concern by directly integrating the entire continuous state space with each action, contrary to the radial basis functions. Since, the number of states could be infinite, hence a Gaussian distribution over the number of states still suffers from state information loss. The $Q(s, a)$ function in multivariate Gaussian is represented by Eq. (3), where $\mu$ and $\sum$ are the parameters to be estimated through sequential agent-environment interaction.

$$\begin{aligned} Q(s,a) &= \frac{1}{(2\pi)^{n/2}|\sum_a|^{1/2}} \\ &\exp(-1/2(s-\mu_a)^T \sum_a^{-1}(s-\mu_a)), \\ &\text{where } s, \mu \in R^n \text{and } \sum_a \in M_{nxn} \end{aligned} \tag{3}$$

The parameters of each Gaussian, representing a particular discrete action, are updated using *TD* RL algorithm, denoted by Eq. (4) [23].

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max(Q(s',a')) - Q(s,a)) \tag{4}$$

The ultimate goal of the GBNLFA-based RL agent is to identify the optimal $Q(s, a)$ denoted by $Q^*$ (Eq. (5)) through repetitive interaction with its intended environment using the *TD* learning method.

$$Q^*(s,a) = argmax \qquad Q(s,a) \tag{5}$$

The RL agent chooses an action corresponding to the Gaussian distribution with maximum pdf value among all the independent Gaussian distributions. The estimated $TD_{\text{update}} = \alpha(r + \gamma \max(Q(s', a')) - Q(s, a))$, from Eq. (4), is used to update the parameters of the corresponding the Gaussian distribution representing the selected action. In this manner, the selected Gaussian distribution receives sequential TD updates throughout the learning experience following $\epsilon$-greedy RL policy.

The pdf of a Gaussian distribution denotes the distance of an $n$-dimensional point $s$ in space from the mean ($\mu$) of the distribution. The area between these two points denotes the probability density of point $s$. Using Eq. (5), we obtain the maximum $Q(s, a)$ value among all independent Gaussians. The maximum pdf represents the maximum distance of point $s$ from $\mu_a$ of a Gaussian distribution corresponding an action $a$. The maximum distance spans the maximum area under the curve (between $s$ and $\mu$), which represents the maximum probability of action $a$ in state point $s$. The inputs to the Algorithm 1 are: (1) state point $s$; (2) action $a$ with maximum pdf; (3) Gaussian associated with action $a$.

**Algorithm 1** GBNLFA

1: **Input:** $s \in \mathbb{S}, a \in \mathbb{A}, G_a$ is Gaussian of action $a$ and $n$ is the sample size.
2: **Output:** $Q(s, a)$ and $G_a$
3: $G_a \sim \mathcal{N}(\mu_a, \sum_a)$
4: $Q_{value} = Q(state, action)$ (Eq. (2) or Eq. (3))
5: Obtain $TD_{update}$ (Eq. (4))
6: Obtain $\mu_a$ and $\sum_a$ (diagonal matrix) from $G_a$
7: $\mu = \mu_a + \frac{(TD_{update} - \mu_a)}{n+1}$ (Corollary 1)
8: $\sum = diag(\frac{n}{n+1}(\sum_a + \frac{(TD_{update} - \mu_a)^2}{(n+1)}))$ (Corollary 2) ($\sum_a$ remains diagonal)
9: if($|\sum| > 0$) then
10: update $\mu_a$ with $\mu$ and $\sum_a$ with $\sum$ (if determinant is positive then diagonal matrix is positive definite)
11: else
12: no update

The algorithm updates the parameters of the Gaussian corresponding to the action $a$ with $TD_{\text{update}}$.

The Gaussian distribution based RL model, denoted by $G_a$ in Algorithm 1, clusters the $Q(s, a)$ in the form of distribution parameters. On observing a new state, an action $a$ with maximum pdf value is identified and used in Algorithm 1 to further update the $\mu_a$ and $\sum_a$ of $G_a$. The matrix, denoted by $\sum_a$, represents the variances of the state variables along the diagonal. The correlation between the state variables is not known, hence the covariance matrix contains individual variances. The determinant of $\sum_a$ is the product of the diagonal elements. Hence, the positive determinant indicates the positive definiteness of the matrix and negative determinant means the matrix is negative definite. Each Gaussian distribution gets updated independently (without affecting the other distributions) during the sequential decision making process of the RL agent.

The proposed FA scheme can be scaled for high dimensional state space RL problems easily as compared to DQN due to the linear order growth. However, the empirical results evidently confirm that the GBNLFA performs better than both benchmark FA methods in the CSDA RL problems of low dimensional state space, considered here. Hence, the shallow NNs containing only 3 layers (input, hidden and output) were employed in DQN replication, with *adam* optimizer and *mean-squared-error* loss function.

## Experiments

### Hardware and Software Libraries

An empirical study is conducted on two classic RL problems, namely Cart Pole balancing and Puck World, and one advanced problem from the financial domain, known as market making (MM). In classical problems, *openAI gym* environment is used to simulate the environment.

In MM, *Boost-c++* and *Yaml-c++* are used for handling utilities including filesystem operations. An open source c++ library for linear algebra, known as *Armadillo*, is used for handling matrix operations. Pycharm is used for python, and Microsoft visual studio for c++ programming. The hardware infrastructure used is described below:

- processor: Intel64 Family 6 Model 142 Stepping 10 GenuineIntel 1600 Mhz
- system type: x64-based PC
- os: Microsoft Windows 10 Pro
- system model: Dell Latitude 5590

### Cart Pole

Cart pole balancing is a well known classic problem in the CSDA RL domain. A pole is fixed using a passive pivot joint with a moving cart on a fixed platform. The cart moves either left or right in order to maintain the balance of the pole. The *left* and *right* actions of the cart prevent the pole from falling off. The cart has a continuous state space containing four real-valued variables, namely *position*, *velocity*, *angle* and *angular-velocity*. The reward function returns a scalar value for every action taken within an episode. The goal of the agent is to get maximum reward value in a minimum number of episodes. An episode terminates in two cases: (1) if the angle of the pole is more than 12° on either side from the vertical axis; or (2) if the position of the cart is more than 2.5 cms on either side from the centre. The cart pole system is considered balanced whilst these two conditions are satisfied, otherwise the system becomes unbalanced. The hyperparameter configuration used in empirical analysis is shown in Table 1.

#### Results

The score in Fig. 2a denotes the cumulative time in milliseconds for which the pole maintains a vertical balance until the termination step (pole falls off) starting from initial state. The GBNLFA method maintains a higher cumulative score than the DQN and linear FA method over 100 consecutive episodes (Fig. 2a). Moreover, the GBNLFA based RL agent successfully converges to an optimal behaviour (Fig. 2b) encoded in the reward function, similar to the benchmarks. The curves shown represent the cumulative score value (score summed up at the end of each episode), and therefore the uppermost line receives the highest cumulative score during the agent-environment interaction phase.

**Table 1** Hyperparameters of cart pole and puck world

| Serial no. | Hyperparameter | Value | Range |
|---|---|---|---|
| 1 | No. of state variables for cart pole | 4 | |
| 2 | No. of actions for cart pole | 2 | |
| 3 | No. of state variables for puck world | 6 | |
| 4 | No. of actions for puck world | 5 | |
| 5 | RL algorithm | Sarsa | |
| 6 | RL policy | $\epsilon$-greedy | |
| 7 | Number of tiles | 8 | |
| 8 | Initial Gaussian | $\mathcal{N}(0, 1)$ | |
| 9 | Discount rate | 0.95 | 0–1 |
| 10 | Exploration rate | 0.5 | 0–1 |
| 11 | Learning rate | 0.001 | 0–0.1 |
| 12 | Activation function | ReLu | |
| 13 | Optimizer function | Adam | |

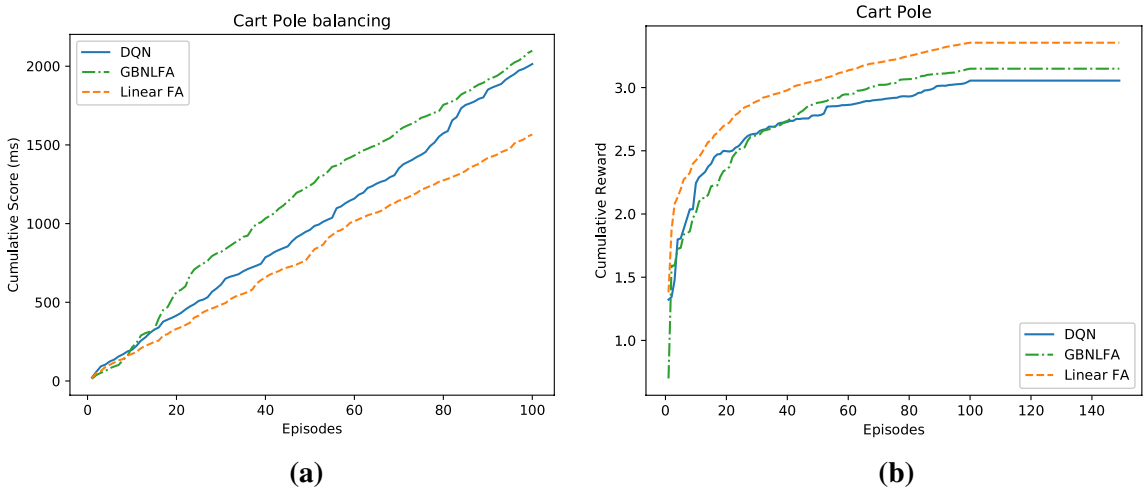


**Fig. 2** **a** Cumulative vertical standing time (in milliseconds) of the pole (score) vs episodes. **b** GBNLFA convergence in cart pole

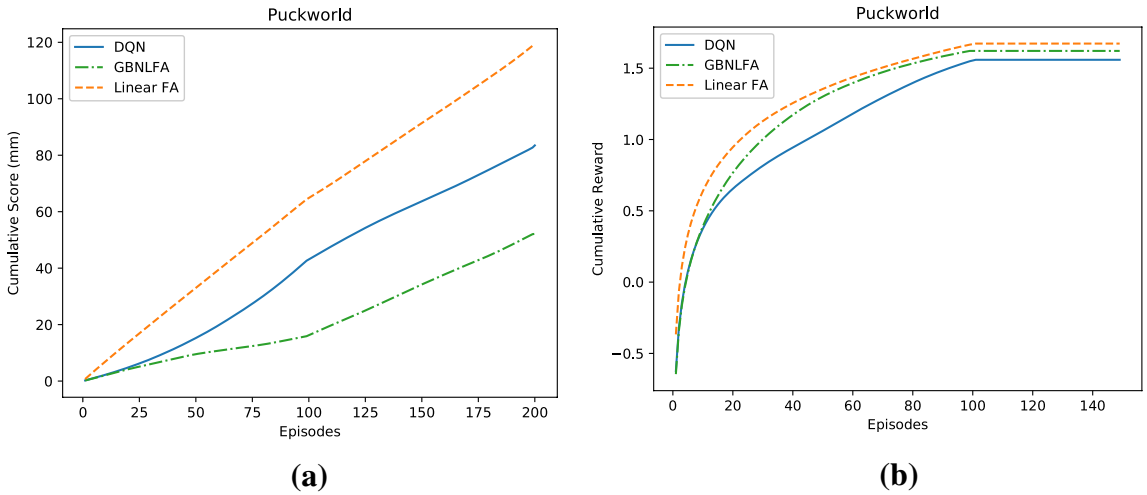


**Fig. 3** **a** Cumulative distance (in millimeters) of the puck agent from the target (score) vs episodes. **b** GBNLFA convergence in puck world

## Puck World

The puck world game contains a random moving target within a grid of fixed size. Puck agent (RL agent) follows the green target slowly, based on the state space vector containing six variables. This problem belongs to the CSDA RL domain, and hence is suitable for evaluating the GBNLFA method. The state space vector contains *agentX*, *agentY*, *agentVelocityX*, *agentVelocityY*, *targetX*, *targetY*, whereas the action space has 5 discrete actions, namely *left*, *right*, *up*, *down*, *standStill*. The RL based puck agent receives the reward based on its distance from the target. The lower the distance, the higher the reward. The goal of the puck agent is to minimize its distance from the randomly moving target in a minimum number of episodes. The hyperparameter configuration used in empirical analysis is shown in Table 1.

### Results

The score in Fig. 3a denotes the distance, the lower the score the better the behaviour of the agent. The reward function is the inverse of the distance between puck agent and the target, and the learned behaviour of the GBNLFA based puck agent is shown in Fig. 3b. The convergence of GBNLFA is evident from the reward behaviour similar to the benchmark FAs. Figure 3a depicts the comparative performance between GBNLFA, DQN and the tile-codings.

## Market Making

MM is a well known strategy of high frequency trading. In stock markets, market makers are responsible for providing and enhancing the market liquidity by ensuring smoothness in order arrival and execution [7].

MM preserves the trading interest of the participants by maintaining sufficient liquidity of the stock market. Market makers earn money from the difference between their quoted ask (sell) price and quoted bid (buy) price, known as quoted spread (QS). We study this problem from an RL perspective, and a recent work [22] uses a linear method of function approximation, i.e. tile-codings to design a RL based MM agent. We consider this work as a benchmark and refer to it as *RMM-Spooner*. Moreover, we also use a traditional MM model developed by Avellaneda and Stoikov [2] as another benchmark for this problem. We call this traditional MM model the *AS-model*. From an RL viewpoint, MM is taken as a CSDA problem. The market maker or RL based MM agent uses a limit order book (LOB) as the trading environment. The state space of a MM agent is a tuple of 7 real-valued variables, namely *volatility*, *relative strength index*, *book imbalance*, *inventory*, *ask level* and *bid level*. The action space constitutes a fixed number of discrete actions, and each action specifies the ask and bid level in the LOB. The action space contains 8 actions, namely *quote(1, 1)*, *quote(2, 2)*, *quote(3, 3)*, *quote(1, 0)*, *quote(0, 1)*, *quote(2, 0)*, *quote(0, 2)* and *clear inventory*. The reward is a function of the investment return (profit/loss) of the MM agent and the inventory holding (Eq. (6)). The RL based MM agent reinforces to maximize the return ($\phi$) and minimize the product of $\lambda$ (inventory controling parameter) times the inventory position (inv) and price of an equity ($\xi$).

$$\text{Reward} = \phi - (\lambda \times \max((\text{inv} \times \xi), 0)) \tag{6}$$

### Data and Hyperparameter Settings

We collect level 2 order book data from an open source data provide, namely namely Chicago Boards of Options Exchange (CBOE). CBOE provides level 2 Trade and Quote (TAQ) data between market opening and closing hours. The technique used in data gathering is *Web Scraping* using *beautiful-soup (a python library)*. The Exchange Traded Funds (ETFs) are well-known and a popular choice of investment among traders these days. Therefore, we gathered TAQ data for 5 ETFs and 5 Options for empirical analysis of GBNLFA, tile-codings and DQN in MM domain. These 10 equities are randomly selected from top 100 in the list on Yahoo finance. The data collected through scraping requires preprocessing such as discarding redundant rows, discarding rows containing nulls, discarding extra columns in rows. We write our own scripts (github link[1]) for data preprocessing. The preprocessed and cleaned data is splitted in training, validation and test sets with 80%, 20% of training data and 20%. The RL agent draws a sample of size 3 (Markovian property states that the current state of RL agent depends on the immediate past and the immediate future state, therefore three states are drawn) from training, validation and test sets at every discrete time step in each episode.

[1] https://github.com/Haider93/mm_data/blob/main/preprocess_data.py.

**Table 2** MM empirical settings

| Serial no. | Hyperparameter | Value | Range |
|---|---|---|---|
| 1 | No. of state variables | 7 | |
| 2 | No. of actions | 8 | |
| 3 | RL algorithm | Sarsa | |
| 4 | RL policy | $\epsilon$-greedy | |
| 5 | Training data (split %) | 80 | |
| 6 | Validation data (split %) | 20 | |
| 6 | Testing data (split %) | 20 | |
| 7 | Backtesting episodes | 20 | |
| 8 | No. of tillings in tile-codings | 8 | |
| 9 | Initial Gaussian | $\mathcal{N}(0, 1)$ | |
| 10 | Inventory range | [− 100, 100] | |
| 11 | Order size | 10 | |
| 12 | Discount rate | 0.95 | 0–1 |
| 13 | Exploration rate | 0.5 | 0–1 |
| 14 | Learning rate | 0.001 | 0–0.1 |

Some hyperparameters, shown in Table 2, are same across all CSDA RL problems considered (Cart Pole, Puck World and MM) such as discount rate, exploration rate, learning rate, RL algorithm, RL policy, number of tiles, and the recommended values from Sutton and Barto [24] are used. The hyperparameters specific to RL based MM, namely number of state variables, number of actions, inventory range, order size, are adopted from the benchmark [22].

GBNLFA based MM agent is trained for 100 episodes. Whereas, the *RMM-Spooner* benchmark is trained for 2000 episodes on all datasets. The out-of-sample backtesting, shown in Table 3, is conducted on 20% of the total amount of preprocessed data for 20 episodes both for GBNLFA and *RMM-Spooner*.

### Results and Statistical significance

RL based MM agents with two variants of FAs, GBNLFA and *RMM-Spooner*, are compared on two parameters, namely the investment return of the MM agents and the QS as an indicator of market liquidity provided. The higher the investment returns and lower the quoted spread, the higher is the market liquidity provided by the MM agent. GBNLFA has shown consistent performance and outperforms *RMM-Spooner* in terms of both parameters. From the analysis of Table 3, the proposed method yields a profit in 9 out of 10 equities whereas RMM-Spooner obtains profit in 3 out of 10 cases. Moreover, the average quoted spread provided by GBNLFA is lower in 9 out of 10 equites compared with

**Table 3** Out-of-sample backtesting comparison between GBNLFA, tile-codings and traditional MM model

| Criteria | GBNLFA | | RMM-spooner | | AS-model | |
|---|---|---|---|---|---|---|
| | Return | QS | Return | QS | Return | QS |
| SPY | 134.1685 | 0.049 | 40.9792 | 0.1515 | − 3.60791 | 0.10000 |
| DIA | 234.601 | 0.1035 | 119.4805 | 0.136 | −4.47948 | 0.03818 |
| XLF | 36.5150 | 0.025 | −0.4012 | 0.05 | −4.59474 | 0.00909 |
| GDX | 27.7250 | 0.025 | −5.8076 | 0.0471 | −4.80450 | 0.00909 |
| IEUR | 37.2099 | 0.0155 | −3.1850 | 0.06185 | −4.73971 | 0.01818 |
| VOD | 20.4435 | 0.0578 | 9.7375 | 0.0568 | −4.79472 | 0.00909 |
| CVX | 25.6002 | 0.067 | −14.2416 | 0.1895 | −4.69314 | 0.00909 |
| UPS | 49.5496 | 0.08 | −1.775 | 0.1175 | −4.55836 | 0.02636 |
| TXN | 2.5601 | 0.0675 | −15.7651 | 0.147 | −4.76562 | 0.02455 |
| GSK | −1.006 | 0.08303 | −5.3405 | 0.1263 | −4.45971 | 0.05455 |

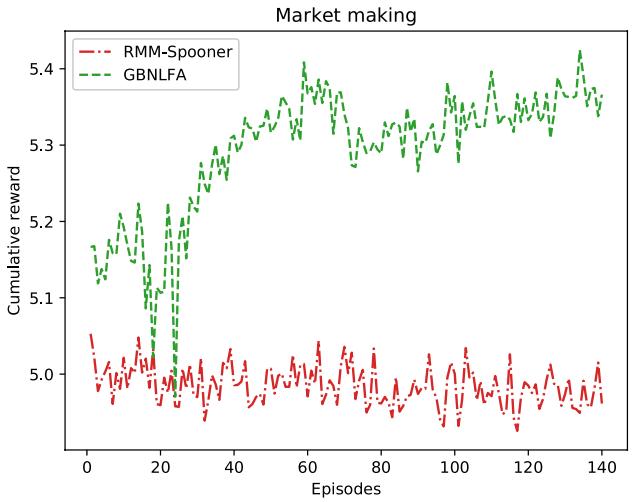


**Fig. 4** GBNLFA convergence in market making

*RMM-Spooner*. Hence, lower quoted spread means higher market liquidity. GBNLFA outperforms the traditional *AS-model* in terms of the investment returns in all equities. However, the traditional MM method is seen to provide lower spread than the other two methods. The convergence of GBNLFA, as shown in Fig. 4, proves that the proposed algorithm successfully converges to an optimal MM behaviour. These out-of-sample backtesting performances show that GBNLFA is a better function approximation method for designing an RL based MM agent.

We also conduct a *t-test* and *ANOVA* test to determine the significance of the differences between the results among the 3 groups, namely GBNLFA, *RMM-Spooner* and *AS-model*. ***Null-hypothesis 1***: There is a significant difference between the returns of GBNLFA and *RMM-Spooner*.

***Null-hypothesis 2***: The returns of GBNLFA and *AS-model* are equivalent.

***Significance level***: 0.05 (default value)

The *t-test* between the 10 observations of GBNLFA and *RMM-Spooner* generates a $p$-value = 0.115. The *t-test* between GBNLFA and *AS-model* gives a $p$-value = 0.026. In case of ANOVA, the p-value between GBNLFA and *RMM-Spooner* is 0.110, whereas between GBNLFA and *AS-model* is 0.016. Conclusively, the ***Null-hypothesis 1*** cannot be rejected (the associated $p$-value is higher than 0.05) and ***Null-hypothesis 2*** cannot be accepted (the associated $p$-value is lesser than 0.05).

## Description of Experiments

The Cart Pole and Puck World are two well-known classical CSDA RL benchmarks. To compare GBNLFA with tile-codings and DQN, we use openAI-gym library in python to simulate Cart Pole[2] and Puck World[3] environments. These three algorithms provides three variants of a RL agent for both Cart Pole and Puck World, as shown in empirical results. OpenAI-gym provides built-in libraries for Cart Pole and Puck World, the RL agent instantiates these environments to train itself for a fixed number of episodes. The agent observes a sequence *state-action-reward-state'-action'* [24] in a discrete time step in each episode. For each state-action pair, the environment generates a reward signal which is a feedback for the agent. The agent uses this feedback signal to learn and fit the model provided by three algorithms (GBNLFA, tile-codings and DQN). In each episode, the cumulative scores (pole vertical standing time in ms for Cart Pole and distance between Puck agent and target in Puck World) and the cumulative reward values are collected. These scores and reward, as shown in Figs. 2a, b, 3a, b, are the assessment metrics or statistics for both classical problems. The average running times of GBNLFA, tile-codings and DQN for Cart Pole and Puck World are 0.36s, 0.38s, 0.38s per episode, respectively.

In MM[4] problem, the Trade and Quote data is gathered and preprocessed to simulate a level 2 order book. The order

[2] https://github.com/Haider93/cart-pole.

[3] https://github.com/Haider93/puck-world.

[4] https://github.com/Haider93/market-making.

book acts as the environment for the RL based MM agent. The MM agent is responsible for placing ask and bid quotes in order book, simultaneously. We use historical Quotes data for simulating the actual order book and executing the trades occured in past to generate the investment returns (PnL in USD) using hyperparameters shown in Table 2. In each episode, the MM agent simulates a level 2 order book from Quotes data and updates the order book at a frequency of 5 seconds. Then, the agent places an ask and bid order on both sides and at same or different levels of the order book. The book is updated with the historical trades, the ask and bid orders of the MM agent also get executed during updation. If both ask and bid orders get executed, then the bid-ask spread of these orders are accumulated to reckon the final return value. The average return in USD and *Quoted-Spread(QS)*, as shown in Table 3, are the assessment metrics or statistics in MM problem. The average running time of an episode for GBNLFA, *RMM-Spooner* and *AS-model* are 0.3s, 0.5s and 0.4s, respectively.

### Description of Algorithms

We compare three algorithms, namely GBNLFA, tile-codings and DQN both in both classical RL problems (Cart Pole and Puck World) and an advance MM problem from finance domain. In GBNLFA, a group of Gaussian distributions represents a model for the RL agent. Each Gaussian distribution is responsible for storing the learning experience i.e. state-action mapping as mentioned in "GBNLFA" section. The *temporal difference error* (refer Eq. (4)) is used to fit the parameters of each Gaussian distribution in the group. The formulas used for updating the Gaussian parameters are depicted at step 7 and 8 in Algorithm 1 along with their mathematical proofs. The RL agent learns from the collection of sequences *state-action-reward-state'-action'* known as learning experience. Each such sequence generates the *temporal difference error* which is further used to update the distribution parameters.

In tile-codings, the continuous state space is discretized using a fixed number of tiles or groups. The states are aggregated based on their similarity with each other. Each tile represents a discrete state, which is a combination of multiple similar states. The similarity among states is measured using the distance similarity metric e.g. Euclidean distance. Then, these tiles are used to derive a state-action mapping or a RL policy of action selection. In DQN, a fully connected feed-forward NN represents the learning model or more specifically the RL policy. The NN based model takes state space and an action as input and estimates the Q value as the output. Each neuron has an activation function and the NN uses a learning algorithm which is responsible for learning the weights of input layer.

The hyperparameters shown in Tables 1 and 2 is the best combination used by GBNLFA, tile-codings and DQN. These algorithms use different FA methods e.g GBNLFA uses a group of Gaussian distributions, tile-codings is itself a FA method and DQN uses NN. However, the algorithms are based on RL, hence the recommended RL settings, suggested by Sutton and Barto [24], are used by all of them e.g. discount rate, learning rate, exploration rate, RL algorithm, RL policy, number of tiles. Some hyperparameters are problem specific, namely inventory range, order size of MM agent, and are adopted from benchmark. In DQN FA method, the *ReLu* activation and *Adam* optimizer are well-known and widely used methods of training a NN.

## Time Complexity Analysis

In this section, we calculate the worst case time complexity of GBNLFA and compare it with the linear and non-linear FAs. The asymptotic complexity analysis provides a fair comparison in terms of a numerical function *T(n)*, where $n$ is the number of inputs. Some operations such as array element access, initialization and assignment are amortized time operations, i.e. they take constant time ($O(1)$ or C).

### Linear Function: Tile-Codings

Tile-coding belongs to the linear FA method and is widely used to solve CSDA RL problems. In this algorithm, the sum of the number of state variables and the number of action variables is mapped on a fixed number of tiles. The Cartesian product of $n + n_a$ (number of state variables + number of action variables) with $n_t$ (number of tiles) generates all possible pairs of mapping between input variables and the tiles.

Number of state variables: $n$
Number of action variables: $n_a$
Number of tiles: $n_t$
$T(n) = (n + n_a) \cdot n_t$
$T(n) = (n + n_a) \cdot n_t$ (assuming $n_t = n$ and $n = n_a$)
$T(n) = 2n \cdot n$
$T(n) = 2n^2$
$T(n) \approx O(n^2)$
Quadratic order growth.

### NN Based FA:DQN

DQN is a popular FA method algorithm which solves CSDA RL problems, like tile-codings. The total number of input variables is the sum of number of state and action variables. Moreover, the number of hidden layers and the number of neurons in each hidden layer also contribute to the qsmptotic performance of the algorithm. The $T(n)$ function, denotes the growth of the algorithm with the size of input, and is equal to the product of total number of input variables, number of hidden layers and number of neurons in each hidden layer.

Number of state variables: $n$
Number of action variables: $n_a$
Total number of output neurons: 1
Number of hidden layers: $n_h$
Number of neurons in each hidden layer: $k$
Type of neural network: fully connected
$T(n) = (n + n_a) \cdot n_h \cdot k \cdot 1$
$T(n) = 2n \cdot n_h \cdot k$ (assuming $n_h = k = n$ and $n = n_a$)
$T(n) = 2n \cdot n \cdot n$
$T(n) = 2n^3$
$T(n) \approx \mathrm{O}(n^3)$
Cubic order growth.

### Gaussian Distribution: GBNLFA

GBNLFA is a Gaussian based FA method algorithm also solves CSDA RL problem similar to the tile-codings and DQN. We compute the $T(n)$ function at each step of Algorithm 1. In this algorithm, the state variables are mapped on a group of Gaussian distributions (each Gaussian represents a particular action). All the primitive mathematical operations (addition, subtraction, multiplication, Square root, assignment) are assumed to take amortized time (denoted by $C$) depending on the hardware infrastructure. For instance, in step 4 $T(n)$ is the sum of 5 primitive operations, refer Eq. (3), and is equal to $5n$. Each primitive operation in step 4 can grow linearly ($O(n)$) with the number of state variables $n$. In this manner, $T(n)$ function is computed at every step of Algorithm 1, and 18 primitive operations with linear order growth ($O(n)$) are computed in step 10.

Number of state variables: $n$
Number of action variables: $n_a$
Step 3: Assignment operations takes amortized time.
$T(n) = C, C$ is constant time.

Step 4: $T(n) = T(n) + 5 \cdot n$(= 1 subtraction +1 multiplicative inverse +2 multiplications +1 square root)

Step 5: $T(n) = 5 \cdot n$ (for $Q(s', a')$ from step 4) $+5 \cdot n_s$ for (Q(s, a))$+6 \cdot C$ (= 2 multiplications +2 additions +1 subtraction +1 C from step3)

$T(n) = 10 \cdot n + 6 \cdot C$

Step 7: $T(n) = T(n) + 4 \cdot C$ (= 1 addition +1*subtraction* +1*division* +1*assignment*)

$T(n) = 10 \cdot n + 10 \cdot C$

Step 8: $T(n) = T(n) + n$ subtractions

$T(n) = 11 \cdot n + 10 \cdot C$

Step 8: $T(n) = T(n) + n$ subtractions $+n$ self multiplications

$T(n) = 13 \cdot n + 10 \cdot C$

$T(n) = T(n) + 6 \cdot C$ (= 3 additions +1*multiplication* +1*division* +1 assignment)

$T(n) = 13 \cdot n + 16 \cdot C$

Step 9: $T(n) = T(n) + (n - 1)$ multiplications for determinant of the matrix and +1 comparison against zero

$T(n) = 14 \cdot n + 16 \cdot C$

Step 10: $T(n) = 14 \cdot n + 16 \cdot C + 2\dot{n}$ assignments to update parameters of Gaussian

$T(n) = 16 \cdot n + 16 \cdot C + 2n_a$ number of parameters of Gaussian grows linearly with the number of actions

$T(n) \approx 18 \cdot n$ (assuming $n_a = n$)

$T(n) \approx n$

$T(n) \approx O(n)$

Linear order growth.

GBNLFA: $O(n)$ < Tile-codings: $O(n^2)$ < DQN: $O(n^3)$.

## Conclusion

This paper introduced a novel parametric non-linear FA method based on Gaussian distribution for the CSDA RL domain, known as GBNLFA. The pdf of the Gaussian distribution serves as the action-value function ($Q(s, a)$) of the RL agent. This novel method resolves the state information loss problem in tile-codings due to state discretization and provides an asymptotically faster ($O(n)$) FA algorithm among the linear ($O(n^2)$) and non-linear ($O(n^3)$) parametric class of FAs in CSDA problems. Empirical scores on three CSDA problems (Cart Pole, Puck World and MM) show that GBNLFA outperforms the benchmark FA methods in both linear and non-linear parametric class.

MM is an advanced problem where the market-maker is responsible for providing the market liquidity. The out-of-sample backtesting results (see Table 2) show that GBN-LFA based MM agent yields higher returns than the MM benchmarks, namely *RMM-Spooner* and *AS-model*.

## Appendix

**Corollary 1** *A sample S of size n with mean $\mu$. A new observation X is added, then the updated mean $\mu'$ is defined as* $\mu' = \mu + \frac{X-\mu}{n+1}$

***Proof*** Let $x_1, x_2, \ldots, x_n$ are the observations in a sample S with mean $\mu$. Let X be the new observation.

$$\text{Then, } \mu = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$n\mu = \sum_{i=1}^{n} x_i \ldots (a)$$

Then,

$$\mu' = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i$$

$$(n+1)\mu' = \sum_{i=1}^{n+1} x_i \ldots (b)$$

Put $(a)$ in $(b)$,

$$(n+1)\mu' = n\mu + X$$

Simplify,

$$\mu' = \frac{n\mu + X}{n+1} \ldots (c)$$

$$\mu' = \frac{n}{n+1}\mu + \frac{X}{n+1}$$

$$\mu' = \frac{n}{n+1}\mu + \frac{X-\mu+\mu}{n+1}$$

$$\mu' = \frac{n}{n+1}\mu + \frac{X-\mu}{n+1} + \frac{\mu}{n+1}$$

$$\mu' = \mu + \frac{X-\mu}{n+1} \ldots (d)$$

□

**Corollary 2** *A sample S of size n with mean $\mu$ and variance $\sigma^2$. A new observation X is added, then the updated variance $\sigma'^2$ is defined as* $\sigma'^2 = \frac{n}{n+1}(\sigma^2 + \frac{(X-\mu)^2}{n+1})$

***Proof*** Let $x_1, x_2, \ldots, x_n$ are the observations in a sample S with mean $\mu$ and variance $\sigma^2$. Let X be the new observation.

By definition,

$$\sigma^2(X) = E(X^2) - [E(X)]^2$$

$$\text{Then, } \sigma^2 = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) - \mu^2$$

$$\sum_{i=1}^{n} x_i^2 = n(\sigma^2 + \mu^2) \ldots (e)$$

Then,

$$\sigma'^2 = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i^2 - \mu'^2 \ldots (f)$$

Put $(e)$ *in* $(f)$,

$$\sigma'^2 = \frac{1}{n+1}(n(\sigma^2 + \mu^2) + X^2) - \mu'^2 \ldots (g)$$

Put $(c)$ in $(g)$,

$$\sigma'^2 = \frac{1}{n+1}(n(\sigma^2 + \mu^2) + X^2) - \left(\frac{n\mu + X}{n+1}\right)^2$$

$$\sigma'^2 = \frac{n\sigma^2}{n+1} + \frac{n\mu^2}{n+1} + \frac{X^2}{n+1} -$$

$$= \frac{X^2}{(n+1)^2} - \frac{n^2\mu^2}{(n+1)^2} - \frac{2nX\mu}{(n+1)^2}$$

$$\sigma'^2 = \frac{n\sigma^2}{n+1} + \frac{n(n+1)\mu^2}{(n+1)^2} + \frac{nX^2}{(n+1)^2} -$$

$$= \frac{n^2\mu^2}{(n+1)^2} - \frac{2nX\mu}{(n+1)^2}$$

$$\sigma'^2 = \frac{n\sigma^2}{n+1} + \frac{n}{(n+1)^2}(\mu^2 + X^2 - 2X\mu)$$

$$\sigma'^2 = \frac{n}{n+1}\left(\sigma^2 + \frac{(X-\mu)^2}{n+1}\right) \ldots (h)$$

**Funding** This study is a part of a PhD project funded by VCRS (Grant number E4G3EN3G39F0A4).

**Availability of data and materials** Data source CBOE.

**Code availability** The project is on-going, hence code will be available after completion.

## Declarations

**Conflict of interest** On behalf of all authors, the corresponding author states that there is no conflict of interest.

## References


1. Anschel O, Baram N, Shimkin N. Averaged-dqn: variance reduction and stabilization for deep reinforcement learning. In: Proceedings of the 34th international conference on machine learning, PMLR, vol. 70. 2017. p. 176–85.
2. Avellaneda M, Stoikov S. High-frequency trading in a limit order book. Quant Finance. 2008;8(3):217–24.
3. Bertsekas DP, Tsitsiklis JN. Neuro-dynamic programming. Nashua: Athena Scientific; 1996.
4. Davies S. Multidimensional triangulation and interpolation for reinforcement learning. https://scottdavies.net/nips96.pdf. 1997.
5. Geist M, Pietquin O, Fricout G. Kalman temporal differences: the deterministic case. In: 2009 IEEE symposium on adaptive dynamic programming and reinforcement learning. 2009.
6. Ghiassian S, Yu H, Rafiee B, Sutton RS. Two geometric input transformation methods for fast online reinforcement learning with neural nets. 2018.
7. Haider A, Wang H, Scotney B, Hawe G. Effect of market spread over reinforcement learning based market maker. In: Machine learning, optimization, and data science, vol. 11943. Cham: Springer; 2019. p. 143–53.
8. Han X, Liu H, Sun F XZ. Active object detection with multistep action prediction using deep q-network. IEEE Trans Ind Inform. 2019;15(6:3723–31.
9. van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. In: Thirtieth AAAI conference on artificial intelligence. 2016.
10. Lagoudakis MG, Parr R. Least-squares policy iteration. J Mach Learn Res. 2003;4:1107–49.
11. Li W, Zhang H, Gao S, Xue C, Wang X, Lu S. Smartcc: a reinforcement learning approach for multipath tcp congestion control in heterogeneous networks. IEEE J Select Areas Commun. 2019;37(11):2621–33.
12. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602 2013.
13. Munos R, Moore A. Variable resolution discretization in optimal control. Springer Mach Learn. 2002;49:291–323.
14. Narvekar S, Stone P. Learning curriculum policies for reinforcement learning. In: Proc. of the 18th international conference on autonomous agents and multiagent systems (AAMAS 2019). 2019.
15. Oroojlooyjadid A, Nazari MR, Snyder LV, Takac M. A deep q-network for the beer game: using machine learning to solve inventory optimization problems. 2017.
16. Otterlo M, Wiering M. Adaptation, learning, and optimization, vol. 12, chap Reinforcement Learning and Markov Decision Processes. Berlin: Springer; 2012.
17. Riedmiller M. Neural fitted q iteration—first experiences with a data efficient neural reinforcement learning method. In: ECML:Lecture notes in Computer Science, vol. 3720. Berlin: Springer; 2005. p. 317–28.
18. Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In: IEEE international conference on neural networks, IEEE. 1993.
19. Rummery GA, Niranjan M. Online q learning using connectionist systems. http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf. 1994.
20. Russo G, Cardellini V, Presti FL. Reinforcement learning based policies for elastic stream processing on heterogeneous resources. In: DEBS '19: proceedings of the 13th ACM international conference on distributed and event-based systems, 2019. p. 31–42.
21. Schaul T, Quan J, Antonoglou I, Silver D. Prioritized experience replay. https://arxiv.org/abs/1511.05952. 2016.
22. Spooner T, Fearnley J, Savani R, Koukorinis A. Market making via reinforcement learning. In: AAMAS2018 conference proceedings, 2018. p 434–42.
23. Sutton RS. Learning to predict by the methods of temporal differences. Springer Mach Learn. 1988;3:9–44.
24. Sutton RS, Barto AG. Reinforcement learning. 2nd ed. Cambridge: The MIT Press; 2018.
25. Tesauro G. Applications of neural networks, Springer, Boston, MA, chap TD-Gammon: a self-teaching backgammon program. 1995.
26. Wang Y, Liu H, Zheng W, Xia Y, Li Y, Chen P, Guo K, Xie H. Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE Access. 2019;7:39974–82.
27. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N. Dueling network architectures for deep reinforcement learning. In: Proceedings of the 33rd international conference on machine learning, PMLR. 2016.
28. Watkins C. Learning from delayed rewards. PhD thesis, Computer Science. 1989.
29. Watkins CJ, Dayan P. Technical note: Q-learning. Springer Mach Learn. 1992;8:279–92.