Action Value Estimation with Compressed Bellman Error Based Feature Generation

Zhizun Wang[1,2], **Doina Precup**[1,2,3]
[1] McGill University, Montreal, Canada, [2] Mila, [3] DeepMind
zhizun.wang@mail.mcgill.ca, dprecup@cs.mcgill.ca

Abstract

In this paper, we consider the task of action-value function approximation in reinforcement learning (RL). General problems of feature extraction and dimension reduction are introduced at the beginning. The theoretical concepts, including Bellman errors and random projections, are discussed in the following parts. Then we propose two useful algorithms, which are able to compress large sparse feature spaces into low-dimensional spaces without losing the information for exact reconstruction. A series of experiments are conducted using predefined RL environments. The performance of our algorithms is compared with classical RL methods. An empirical analysis is presented, and future directions of research are indicated.

Introduction

An important problem in reinforcement learning (RL) is how to handle the tasks in large or infinite state spaces. In such domains, it may not be feasible to represent the value function explicitly. An effective strategy is to employ function approximation and to represent the value function using some parametric function approximators, such as tile coding, neural networks, or radial basis functions. These function approximators, also known as feature constructors for linear methods, are the basic tools for feature selection. Choosing appropriate features for the task is an important way to add prior domain-specific knowledge to RL systems. If we are valuing the states of a pendulum, for instance, we may want to have features for its velocity, torque, and position. The features used in the function approximation can determine the accuracy of parametrized policy evaluation which is computing the expected return of a given policy. An approach that offers sound theoretical guarantees is generating features in the direction of the Bellman errors of the current value estimates. However, it could be computationally expensive while being applied in high-dimensional state spaces. Attention should therefore be paid to the design and adaptation of Bellman error related methods that are computationally efficient.

To achieve computational improvement, we try to design the action-value estimation algorithm from the perspective of random projection. The principle is to iteratively project the original features into exponentially lower-dimensional spaces. The application of random projection has been carefully studied in the fields of signal processing and machine learning, motivated by the need to sample and store large data sets as efficiently as possible. It turns out that integrating the technique of random projection into the generation of Bellman error based features provides great assistance for policy evaluation, which is what we are trying to implement. In this paper, we set out to answer several research questions pertaining to the topic we discussed:

- What theory can support the application of random projections in the exploration on RL?
- What kind of feature space is suitable for the algorithm with random projection?
- Which role does linear regression play in the compressed space?

Background

Modern algorithms try to employ a number of techniques, including dimensionality reduction, in order to make sure that solving problems posed do not become intractable.

Dimensionality Issues

Dimensionality is closely related to the computability of a problem. If the problem space is low-dimensional, it is inexpensive to compute. In order to shape our algorithms to be feasibly computable, we may want to reduce the dimensionality of the problem space.

Formally speaking, given points $P = \{m \in \Re^d\}$, describe P in fewer dimensions $n << m, d$. That is, define a function f such that $\forall p_i, p_j \in P$

$$||f(p_i) - f(p_j)||_2 \approx ||p_i - p_j||_2 \tag{1}$$

This goal of dimension reduction is important in a number of different research areas, such as reinforcement learning and latent semantic indexing. Obviously, it requires the support of some theoretical bases.

Johnson-Lindenstrauss Theorem

A significant result in high dimensional geometry is the Johnson-Lindenstrauss theorem.

Theorem 1 For any $0 < \epsilon < 1$ and any integer n, let k be a positive integer such that

$$k \geq \frac{24}{3\epsilon^2 - 2\epsilon^3} \ln n \tag{2}$$

Then for any set V of n points $\in \Re^d$, there exists a map $f : \Re^d \to \Re^k$ such that for all $u, v \in V$

$$(1-\epsilon)||u-v||^2 \leq ||f(u)-f(v)||^2 \leq (1+\epsilon)||u-v||^2 \tag{3}$$

Furthermore, this map can be found in randomized polynomial time. Repeating this projection O(n) times can boost the probability of success to any desired constant, giving us a randomized polynomial time algorithm.(Dasgupta and Gupta 2003)

The theorem is a result concerning low-distortion embedding of points from high-dimensional into low-dimensional Euclidean space. It states that a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved. The map used for the embedding is at least Lipschitz, and can even be taken to be an orthogonal projection. This theorem has been applied to reducing the problems with large dimensionality or an enormous amount of data points to those with a much smaller one. Many of such problems use low rank approximations of matrices justified by the theorem.

Bellman Operator and Bellman Error

A Markov Decision Process (MDP) can be represented as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where $\mathcal{S}$ is a set of states; $\mathcal{A}$ is a set of actions; R is a reward function; and $\gamma \in [0, 1)$ is a discount factor. It is an extension of the standard Markov model (Ballard 1999). Each state has a collection of actions. The MDP's state transition can be described by the transition kernel P. $P(.|s, a)$ defines the probabilistic distribution, given that the action a is chosen in the current state s. During each time step, the agent chooses an action and receives a reward. Then the environment changes into a new state according to the transition kernel. We always assume discounted reward Markov Decision Processes.

The value of a state s under a policy π, denoted $V^\pi(s)$, is the expected value of the discounted sum of rewards, $\sum_t \gamma^t r_t$. Let $R(s, \pi(s))$ be the expected reward at state s for a policy π. The value function is:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \int V^\pi(s')P(ds'|s, \pi(s)) \tag{4}$$

We can then define an operator $\mathcal{T}$ as

$$\mathcal{T}(V(s)) = R(s, \pi(s)) + \gamma \int V(s')P(ds'|s, \pi(s)) \tag{5}$$

which is the Bellman operator (Fard et al. 2013). The Bellman error is defined as

$$e_V(.) = \mathcal{T}(V(.)) - V(.) \tag{6}$$

There are a number of feature selection methods which generate features in the direction of Bellman error.

To make the implementation simpler, consider using the temporal difference error (TD error) to estimate Bellman error. Let $S_n = ((x_t, r_t)_{t=1}^n)$ be a random sample of size n, collected on a Markov Decision Process under a fixed policy. Given an estimate, est_V, of the value function, TD error is defined to be

$$\delta_t = r_t + \gamma \cdot est_V(x_{t+1}) - est_V(x_t) \tag{7}$$

It can be proved that the expectation of the temporal difference at x_t equals the Bellman error at that point (Sutton and Barto 2018). Therefore, TD errors are proxies to approximating Bellman errors.

Random Projections

A useful method for dimension reduction is the random projection. The idea of random projection is to compress an $n \times m$ feature matrix U to an $n \times d$ matrix V where $d << m$, by multiplying the feature matrix U with an $m \times d$ matrix R:

$$V = UR \tag{8}$$

It is capable of preserving pairwise distances between data points with strong theoretical guarantees, as introduced in chapter 2.

Sparse Random Projection

Proposed by Achlioptas (Achlioptas 2003), sparse random projection is an efficient approach which reduces the dimensionality by projecting the original input space using a sparse random matrix. The elements of the random matrix, R, are sampled from

$$R_{ij} = \begin{cases} -\sqrt{\frac{s}{n_{components}}}, & \text{with probability } \frac{1}{2s} \\ 0, & \text{with probability } \frac{s-1}{s} \\ +\sqrt{\frac{s}{n_{components}}}, & \text{with probability } \frac{1}{2s} \end{cases} \tag{9}$$

where $s = 1/density$, and $n_{components}$ is the dimensionality of the target projection space. By default the density of non-zero elements is set to the minimum density, as recommended by Ping Li et al. (Li, Hastie, and Church 2006): $\frac{1}{\sqrt{D}}$, where D is the dimensionality of the original vector space.

Gaussian Random Projection

Gaussian random projection is one of the most frequently studied algorithms for dimension reduction. It projects the original input space on a randomly generated matrix, R, where components are drawn from the following distribution: $R_{ij} \sim N(0, n_{components})$. Because R is a dense $m \times d$ matrix, the time complexity of Gaussian random projection is $O(nmd)$. In the experiments, we will only apply Gaussian random projection.

An important benefit of the random projection stems from the associative property of matrix multiplication. In some circumstances, the feature matrix U can be a high-order matrix, e.g. $U = M^k$. Computation of the random projection $V = M^kR$ can be done iteratively as $V = \underbrace{(M \ldots (M(M}_{\text{k}} R)))$. This reduces time complexity from $O(n^3kd)$ to $O(mkd)$.

Compressed Bellman Error Based Feature Generation

Parr et al. (Parr et al. 2007) analyzed a general family of approaches that iteratively add basis functions to a linear approximation architecture in a manner where each new basis function is derived from the Bellman error of the previous set of basis functions. These are called Bellman Error Basis Functions (BEBFs), and can be particularly helpful for value-function approximation. By adding new features to the hypothesis space, iterative BEBF generation slowly increases its complexity. As long as the error in the generation does not cancel out the contraction effect of the Bellman operator, iterative BEBF generation will not diverge. Some theoretically correct and interesting algorithms related to the BEBF have then been proposed. Nevertheless, they are relatively difficult to be applied in extremely large state spaces. The problem lies in the large estimation error when predicting BEBFs in state spaces with high dimensionality. Fard et al. proposed a solution to update the state values, leveraging the use of simple random projections to alleviate this problem (Fard et al. 2013). Based on their contribution, we designed the compressed Bellman error based feature generation algorithm for action-value update.

State Value Function Approximation

Computer scientists are trying to design and adapt Bellman error based methods to be more scalable and computationally efficient. Fard et al. (Fard et al. 2013) studied the effect of generating the BEBFs using linear regression in a relatively small vector space induced by random projections. Because Bellman error estimation in high-dimensional sparse spaces may lead to large prediction error, random projections can be applied to exponentially reduce the dimension, only at the cost of a controlled constant bias.

In reinforcement learning, linear function approximation is frequently used to estimate the value of a given state. Let V_n be an estimated state-value function in a linear space defined by a feature set $\{\phi_i\}_{i=1}^{n}$. Parr et al. demonstrated that if a new BEBF $\phi_{n+1} = e_{V_n}$ is added to the feature set, with mild assumption the approximation error on the new linear space spanned by $\{\phi_i\}_{i=1}^{n+1}$ shrinks by a factor of γ. Moreover, if we can estimate the Bellman error within a constant angular error, $\cos^{-1}\gamma$, the error will still shrink.

The facts above are useful to Fard et al. in proposing the method called Compressed Bellman Error Based Feature Generation (CBEBF). In CBEBF, the state features are first projected into a much smaller space by random projections. Then a hyperplane is regressed to the temporal difference errors. This procedure is called compressed linear regression, which leads to the construction of new features. The new features are added to the set of BEBFs, and used with a policy evaluation algorithm to update the estimate of the state value function.

Action Value Estimation with CBEBF Algorithm

By virtue of the studies mentioned above, we present here the Compressed Bellman Error Based Feature Generation for Action Value Estimation.

Algorithm 1 CBEBF for Action Value Estimation

```
Input: Sample of transitions $D_n = \{(S_t, A_t, R_t, S_t^{'})\}_{t=1}^{n}$
Input: Number of BEBFs: $m$
Input: Projection size schedule: $d_1, d_2, ..., d_m$
Input: Base feature function $\varphi : \mathcal{S} \times \mathcal{A} \rightarrow R^D$
Output: Q: estimate of action-value function
1: procedure CBEBF FOR ACTION VALUE ESTIMATION($D_n, \varphi, m, d_{1:m}$)
2:     Initialize $Q_1(S, A)$ to be 0 $\forall$ $S \in \mathcal{S}, A \in \mathcal{A}$.
3:     Initialize the set of CBEBF features $\Psi \leftarrow \emptyset$.
4:     Set $\gamma$ to be a constant.
5:     for $i \leftarrow$ 1 to $m$ do
6:         Generate Gaussian random projection $\Phi^{d_i \times D}$.
7:         Calculate the TD-errors $\delta_t$, where
8:         $\delta_t = r_{t+1} + \gamma Q_i(S_t^{'}, A_{t+1}) - Q_i(S_t, A_t)$
9:         Apply the compressed ordinary least squares
10:        regression using $\Phi^{d_i \times D}$ :
11:            Let $u^{d_i \times 1}$ be the parameter of COLS
12:            regression in the compressed space, using
13:            $\Phi\varphi(S_t, A_t)$ as the inputs and $\delta_t$ as outputs.
14:        Define $\hat{e}_i(S) = u^T \Phi\varphi(S, A)$.
15:        Add $\hat{e}_i(S)$ to the set of CBEBF features, $\Psi$.
16:        Apply policy evaluation on the feature set $\Psi$ to
17:        obtain $Q_{i+1}$.
18:    end for
19:    return $Q_m$
20: end procedure
```

In fact, CBEBF for Action Value Estimation is an on-policy algorithm. The way that it obtains the transitions is by acting according to a fixed policy π, such as ϵ-greedy. The action selection is done by following π, and the sample transitions are collected. The base feature vector is $\varphi(S, A)$, generated by the feature function φ. There are two feasible design choices of φ. First, we can add an extra "action bit" to the vector representing the state S. In this way, it is simple and intuitive to produce a feature vector for the state-action pair. Alternatively, consider constructing the value function that has an estimator for each action. If the action space is defined as $\mathcal{A} = \{A_i\}_{t=1}^{n}$, then m corresponding estimators, $\{\theta_i\}_{t=1}^{n}$, should be built. An estimator θ_i produces the action value specifically for A_i, and only gets updated when A_i is selected by the agent.

Note that the projection size schedule $d_{1:m}$ is a collection of the dimensions into which the original space is projected. We perform the random projection for m times, and every time we can send the feature vector into a compressed space in a different dimension. Therefore the d_i's can all be different values.

Since the CBEBF for Action Value Estimation keeps increasing the complexity of the hypothesis space by adding new features to the set of BEBFs in each episode, it might suffer from over-fitting if it is not stopped at some point. We can use a validation set and compare the estimates with the empirical returns. When the validation error starts to boost, the algorithm should stop generating new features.

Action Value Estimation with Simplified CBEBF Algorithm

In CBEBF for Action Value Estimation, a feature set of BEBFs named Ψ is constructed. At the end of each episode, Ψ is also applied to do the policy evaluation. This means that multiple levels of regression have to be performed. To decrease the number of regressions and reduce the computational complexity, we propose a simplified version of the original algorithm. It is called the Simplified Compressed Bellman Error Based Feature Generation for Action Value Estimation (SCBEBF for Action Value Estimation). Instead of storing new BEBFs in the feature set during each episode, here we only need to add them to the value function approximator, with a constant weight. For each iteration, the action-value estimate can be expressed as

$$Q_{i+1}(S,A) = \sum_{\psi \in \Psi} \psi(S,A) \tag{10}$$

It can be advantageous to use SCBEBF for Action Value Estimation, because it avoids an extra level of regression on the features, and has lower computation complexity. It simply keeps an record of the coefficients of the action-value estimator by summing up all the generated parameters $\Phi^T u$.

Algorithm 2 SCBEBF for Action Value Estimation

Input: Sample of transitions $D_n = \{(S_t, A_t, R_t, S_t^{'})\}_{t=1}^{n}$
Input: Base feature function $\varphi : \mathcal{S} \times \mathcal{A} \to R^D$
Input: Number of BEBFs: m
Input: Projection size schedule: $d_1, d_2, ..., d_m$
Output: Q: estimate of action-value function

```
1:  procedure SIMPLIFIED CBEBF ($D_n, \varphi, m, d_{1:m}$)
2:      Initialize $v^{D\times 1}$, the coefficients of value estimator,
3:      to be 0.
4:      Set $\gamma$ to be a constant.
5:      for $i \leftarrow 1$ to $m$ do
6:          Generate Gaussian random projection $\Phi^{d_i \times D}$.
7:          Calculate the TD-errors $\delta_t$, where
8:          $\delta_t = r_{t+1} + \gamma v^T \varphi(S_t^{'}, A_{t+1}) - v^T \varphi(S_t, A_t)$
9:          Apply the compressed ordinary least squares
10:         regression using $\Phi^{d_i \times D}$ :
11:             Let $u^{d_i \times 1}$ be the parameter of COLS
12:             regression in the compressed space, using
13:             $\Phi\varphi(S_t, A_t)$ as the inputs and $\delta_t$ as outputs.
14:         Update $v \leftarrow v + \Phi^T u$.
15:     end for
16:     return $Q(S,A) = v^T \varphi(S,A)$
17: end procedure
```

Ordinary Least Squares Regression

It is worth mentioning that for both of the algorithms, ordinary least squares (OLS) regression is applied in the compressed space. To make it intuitive, assume the OLS regression is in one-dimensional space. The projection of a single state S is represented by a point. Together the points form the trajectory of the states. At time $t = 0$, we have the estimate of the value of the first state-action pair in the regression

$$\hat{y}_t = \hat{w}^T \Phi \varphi(S_t, A_t) \tag{11}$$

where $\hat{w}$ is a vector of coefficients, Φ is the random projection matrix, and φ is the base feature function. Because it is an estimate, $\hat{y}_t$ has some height from the trajectory. The next step is to compute a new estimate using the reward obtained from the last action

$$\hat{y}_{t+1} = r_t + \hat{w}^T \Phi \varphi(S_{t+1}, A_{t+1}) \tag{12}$$

This process is repeated for all the states. The output of OLS regression, δ_t, is the squared error between the current estimate and the last estimate

$$\delta_t = ||\hat{y}_{t+1} - \hat{y}_t||^2 \tag{13}$$

In fact, we can pretend that we get a true target $y \approx \hat{y}_{t+1}$. First we can fix $\hat{w}$, compute y, then perform OLS regression to obtain a new $\hat{w}$ and a new target y. To fix $\hat{w}$, we may initialize it to **0**. The initial estimate of the target is thus equal to the reward.

We need to be careful when applying the policy evaluation with OLS regression in CBEBF for Action Value Estimation algorithm. The OLS parameters should be re-calculated in each step, instead of being updated each time. For instance, at one time step we have m features in the set of BEBFs Ψ, and we compute their weights; at the next time step, we add n new features to Ψ. Then, we must refit the weights of all $m + n$ features using OLS regression in policy evaluation.

The bias-variance analysis of the ordinary least squares predictor is made by Fard et al.(Fard et al. 2012). They also derived an upper bound for the worst-case error of the OLS predictor in a compressed space. This serves as the theoretical justification for using compressed OLS regression in both algorithms.

Theorem 2 Let $\Phi^{D\times d}$ be a Gaussian random projection and $w_{ols}^{(\Phi)}$ be the OLS solution in the compressed space induced by the projection. Let $\mathcal{X}$ be a D-dimensional k-sparse space, and f be a linear function of $x \in \mathcal{X}$. Assume we are given a training set of n input-output pairs, consisting of a full-rank input matrix $\mathbf{X}^{n\times D}$. Assume an additive bias in the original space bounded by some $\epsilon_f > 0$ and i.i.d noise with variance σ_η^2. Choose any $0 < \delta_{prj} < 1$ and $0 < \delta_{var} \leq \sqrt{2/e\pi}$. Then, with probability no less than $1 - \delta_{prj}$, we have for any $x \in \mathcal{X}$ with probability no less than $1 - \delta_{var}$:

$$\begin{aligned} &|f(x) - x^T \Phi w_{ols}^{(\Phi)}| \leq \\ &||x^T\Phi|| ||(\mathbf{X}\Phi)^{\surd}|| \left((\epsilon_f + \epsilon_{prj})\sqrt{n} + \sigma_\eta \sqrt{\log(2/\pi\delta_{var}^2)}\right) \\ &+ \epsilon_f + \epsilon_{prj}, \end{aligned} \tag{14}$$

where

$$\epsilon_{prj} = c\sqrt{\frac{k \log d \log(12eD/k\delta_{prj})}{d}} \tag{15}$$

and the symbol $|.|$ denotes the size of a set, $||.||$ denotes the L^2 norm for vectors and the operator norm for matrices. Also, we denote the Moore-Penrose pseudo-inverse of a matrix $\mathbf{A}$ with $\mathbf{A}^{\surd}$.

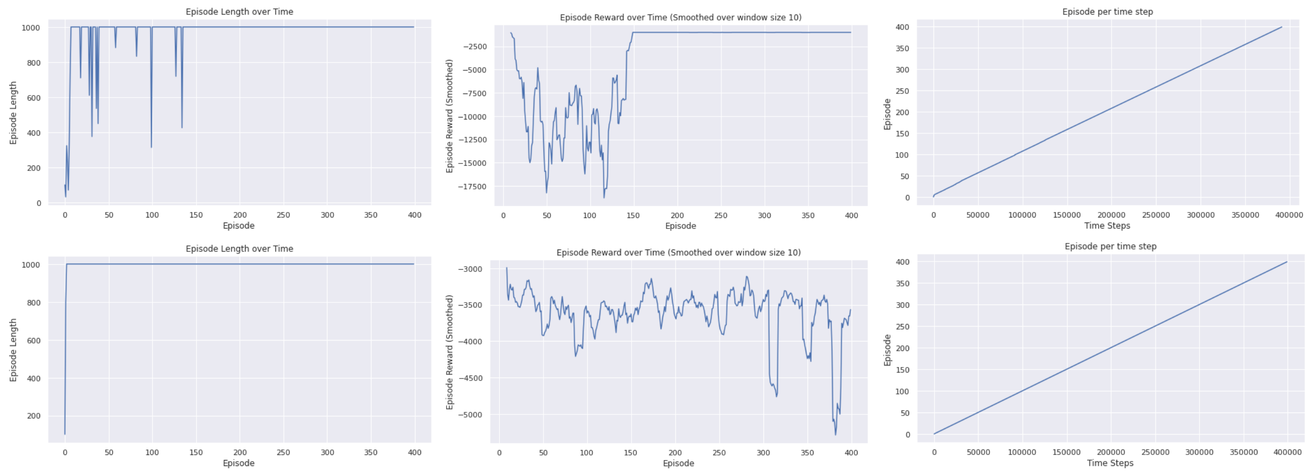

Figure 1: **Top:** Cliff-walking with CBEBF for Action Value Estimation. **Bottom:** Cliff-walking with SCBEBF for Action Value Estimation. **Left:** Episode length over time. **Middle:** Episode reward over time. **Right:** Episode time steps.

Results and Related Studies

We explore the conditions under which the algorithm for action value estimation is useful by conducting a series of experiments. We also try to elucidate why the random projection is unhelpful in some environments. Three groups of experiments are done. The first group is guiding the agent through a board to the goal and trying to avoid falling down the cliff between the start point and the goal. The second one is driving a car in the valley up the hill on its right side. The third group is moving a pole attached by an un-actuated joint to a cart, so that it does not fall over.

Experiments

Because baseline Q-learning and Sarsa are two classical algorithms for exploration in reinforcement learning, we will compare their performance with the performance of our algorithms in each set of experiments. We start by using the cliff-walking environment in Gym for our studies.

It is clear that both Sarsa and Q-learning obtain very good results in the cliff-walking problem. The length of an episode decreases over time, while the reward gradually increases and then stays at a high level. On the other hand, our algorithms do not perform well in the task. The reward given by CBEBF for Action Value Estimation remains stable after 200 episodes, but the reward of SCBEBF for Action Value Estimation keeps fluctuating all the time. The episode lengths of the two algorithms are abnormal, because the maximum length is set to 1000 and when the agent takes steps during an episode, it frequently reaches the limit of time steps. The phenomenon implies that the agent is always unable to find the goal, and is likely to plunge down the cliff repeatedly, being forced to return to the start point.

The cliff-walking environment proves not to be the right choice for the application of our algorithms. The most important reason is that random projections assume a very large feature space. In this environment, there is only a low-dimensional feature space, where we have no need to perform the dimension reduction. Therefore, if we directly use our algorithms which involve random projections, the results will be inferior to those generated by baseline Q-learning and Sarsa. We may consider certain settings with continuous or high-dimensional state spaces. Alternatively, we can use function approximators to construct new feature vectors with high dimensionality.

The next step is to conduct new experiments in the mountain car problem. For this problem, there are several improvements to the implementation of the algorithms. Firstly, we convert the state space into a featurized representation using the radial basis function kernel (RBF kernel) (Vert, Tsuda, and Schölkopf 2004). The constructed feature space is of size 10000. An appropriate setting to apply random projections has been made therefore. Secondly, we normalize the observation samples, so that they have a zero mean and unit variance. Lastly, to visualize the cost to-go for this problem, we plot a function similar to a mountain. The results of the four algorithms, averaged over 5 runs, are displayed in the figures below. The number of episodes, m, was initially set to 400 for all experiments.

In the tests of CBEBF for Action Value Estimation, the problem of over-fitting occurred, as the reward dived after 320 episodes. Resetting m to 250 not only solves the problem, but also yields the most stable episode length over time, among all experiments. The reward given by SCBEBF for Action Value Estimation shows a more steady increase over time than that of Q-learning. The performance of baseline Sarsa is relatively poor, since its reward slowly decreases after reaching the peak. We could affirm that our algorithms outperform Sarsa in the mountain car environment, and have the same level of performance as Q-learning.

Finally, we do research into the cart-pole problem. Again the results are averaged over 5 runs and the default number of episodes, m, is 400. As indicated by the figures, none of the four algorithms delivered a fine performance. Their episode lengths experienced considerable fluctuations, and no prediction could be made. The reward of each algorithm varies substantially over time. It is therefore safe to claim

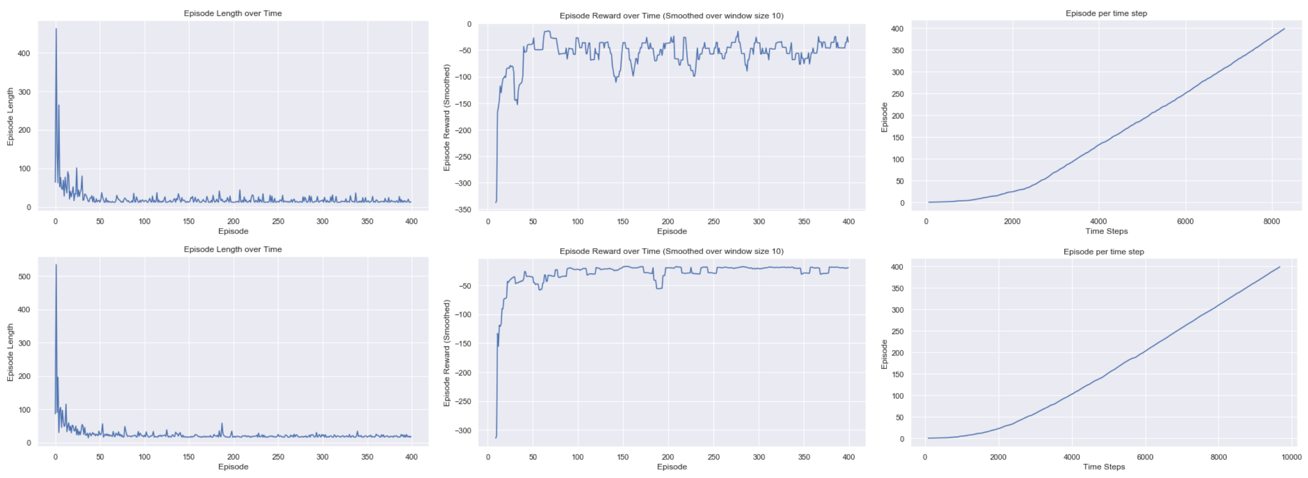

Figure 2: **Top:** Performance of Q-learning in cliff-walking. **Bottom:** Performance of Sarsa in cliff-walking. **Left:** Episode length over time. **Middle:** Episode reward over time. **Right:** Episode time steps.

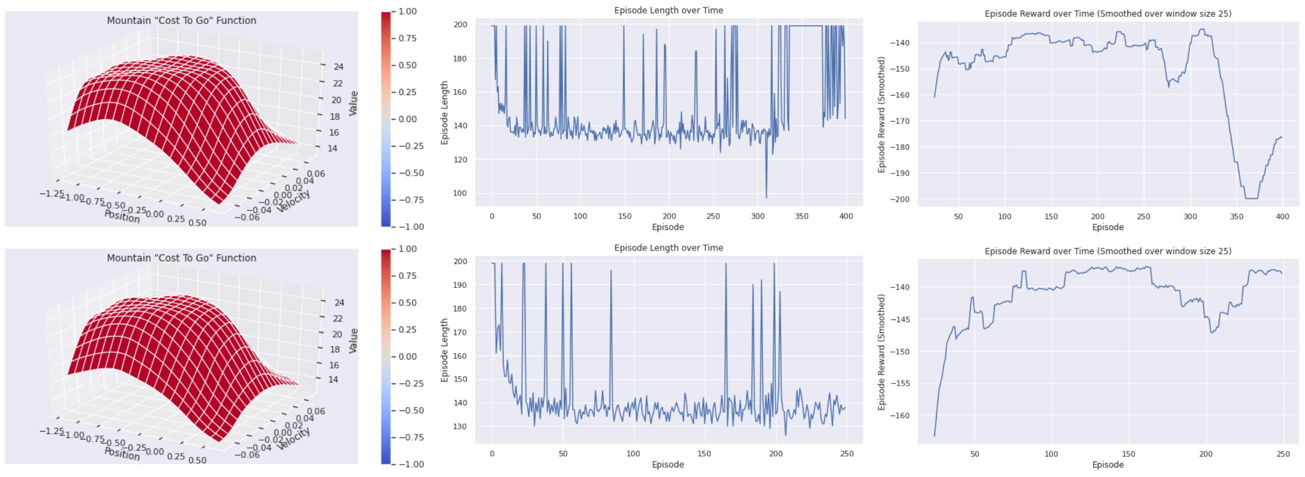

Figure 3: **Top:** CBEBF for Action Value Estimation with $m = 400$. **Bottom:** CBEBF for Action Value Estimation with $m = 250$. **Left:** Cost to-go. **Middle:** Episode length over time. **Right:** Episode reward over time.

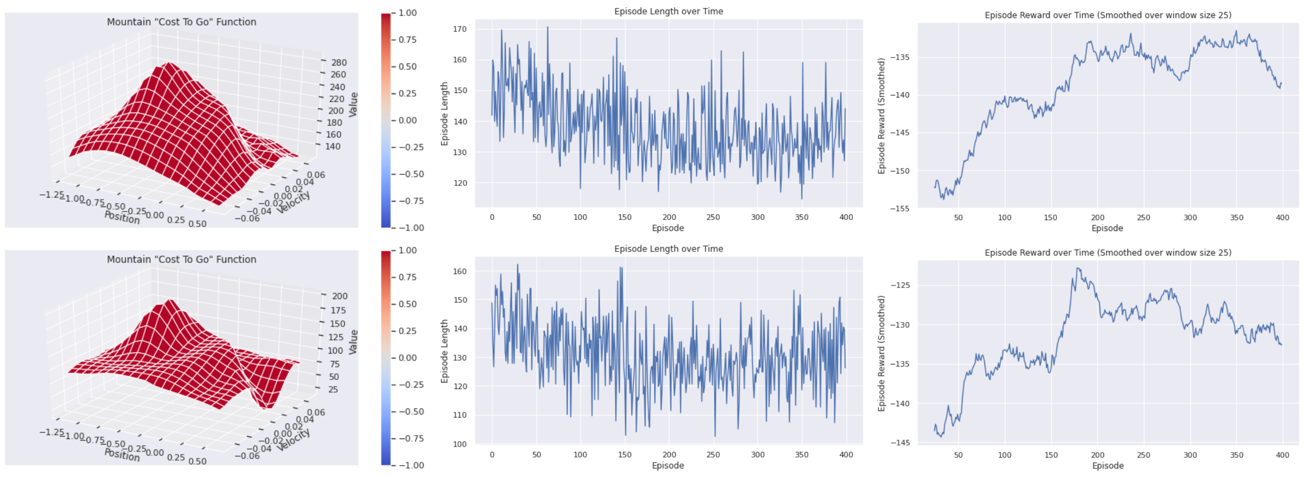

Figure 4: **Top:** Mountain Car with Q-learning, $m = 400$. **Bottom:** Mountain Car with Sarsa, $m = 400$. **Left:** Cost to-go. **Middle:** Episode length over time. **Right:** Episode reward over time.

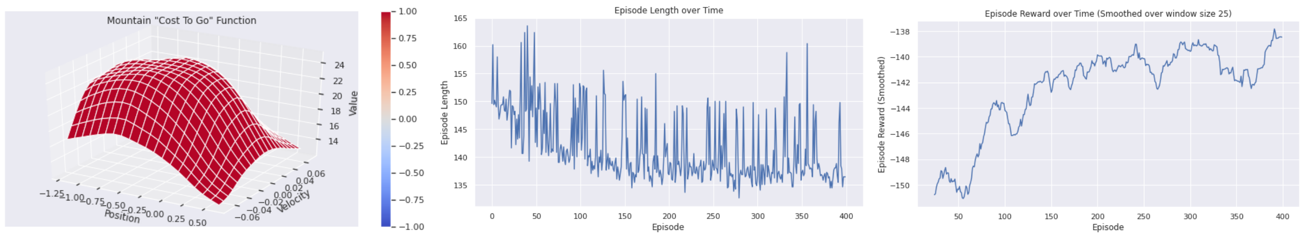

Figure 5: Mountain Car with SCBEBF for Action Value Estimation, $m = 400$. **Left:** Cost to-go. **Middle:** Episode length over time. **Right:** Episode reward over time.

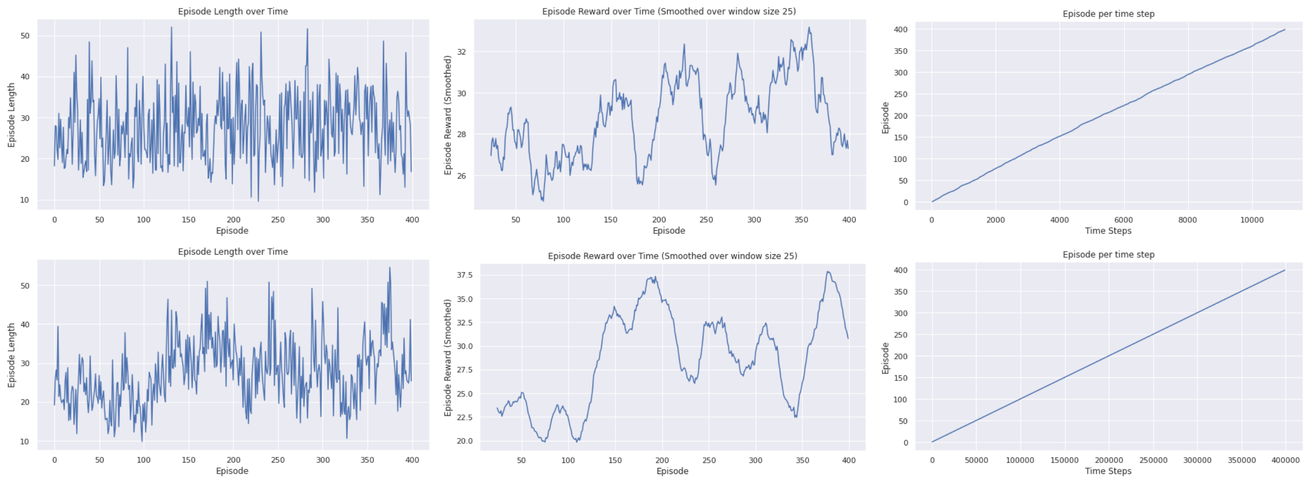

Figure 6: **Top:** Cart-pole CBEBF for Action Value Estimation. **Bottom:** Cart-pole with SCBEBF for Action Value Estimation. **Left:** Episode length over time. **Middle:** Episode reward over time. **Right:** Episode time steps.

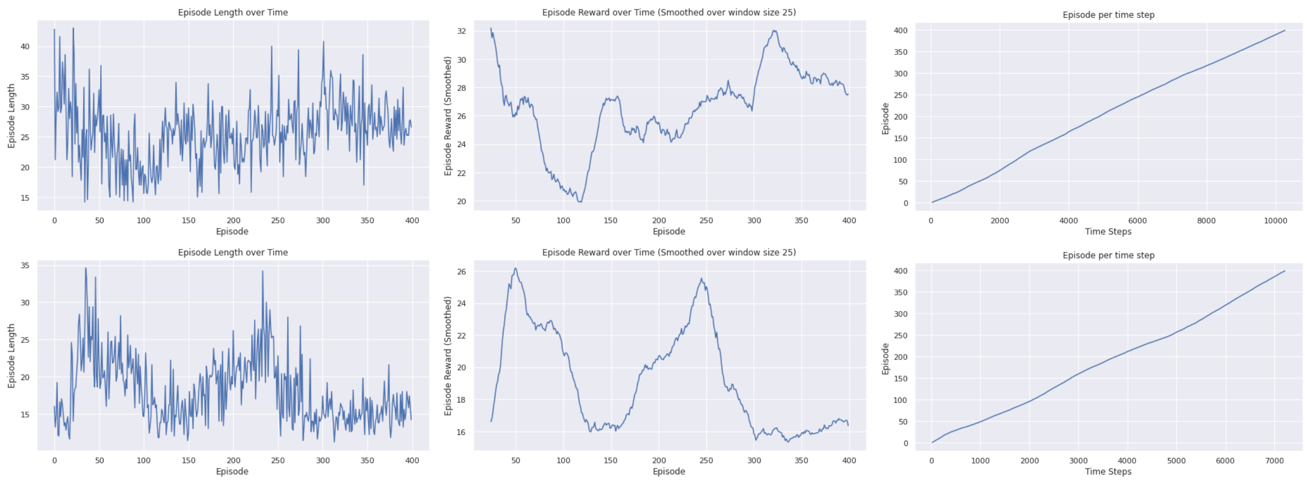

Figure 7: **Top:** Cart-pole with Q-learning. **Bottom:** Cart-pole with Sarsa. **Left:** Episode length over time. **Middle:** Episode reward over time. **Right:** Episode time steps.

that our algorithms are as competitive as baseline Sarsa and Q-learning in this environment. The underlying causes of the problem can be the choice of parameters and the featurization of the state space. It is worth being examined in future studies.

Related Work

Apart from our study on the Bellman error based algorithms, there has been some in-depth work regarding dimensionality reduction and function approximation in reinforcement learning. Keller, Mannor and Precup (Keller, Mannor, and Precup 2006) addressed the problem of automatically constructing basis functions for linear approximation of the value function of an MDP. They discussed a dimensionality reduction technique (Goldberger et al. 2005) in order to map a high-dimensional state space to a low-dimensional space, based on the temporal difference (TD) error. The basic function placed in the low-dimensional state space is added as new features for the linear function approximator. Menache et al. (Menache, Mannor, and Shimkin 2005) examined linear approximation architectures, and presented two algorithms for adapting the basis function during the learning process in the context of evaluating the value function under a fixed control policy. The first involves a gradient-based approach and the second uses the Cross Entropy method. Vapnik et al. (Vapnik, Golowich, and Smola 1997) reported the results of applying the Support Vector method to function approximation and regression estimation.

Random projection and least-squares regression are powerful techniques for feature selection, especially in sparse spaces. They have been analyzed systematically in multiple areas of research. The neuron-friendly random projection was introduced by Arriaga et al. (Arriaga and Vempala 1999) to offer the perspective on cognitive learning that facilitates efficient algorithms which need small samples only. Bingham et al. (Bingham and Mannila 2001) focused on the processing of both noisy and noiseless images, and information retrieval in text documents, using random projections. Their results were comparable to conventional methods such as principal component analysis (PCA), but the latter was computationally significantly more expensive. The numerical complexity of Compressed Least Squares Regression (CLSR) were explored by Maillard et al. (Maillard and Munos 2009), and they provided bounds on the excess risk of the estimate computed in the compressed domain. Ruppert et al. (Ruppert and Wand 1994) investigated the asymptotic properties of multivariate local least-squares regression estimators discussed by Cleveland, Devlin, and Stone (Cleveland and Devlin 1988)(Stone 1980).

Discussion and Conclusion

So far, we have presented the algorithms of action value estimation with CBEBF and SCBEBF, and investigated the effects in different state spaces. Based on the experiments we have done, we conclude that compared to typical reinforcement learning (RL) algorithms such as Q-learning and Sarsa, our algorithms can be as competitive as them in certain environments, with reduced computation complexity and time complexity. But owing to the premise that random projection assumes a state space of high dimensionality, the performance of our algorithms is disappointing in low-dimensional feature spaces, where we should instead use Q-learning or Sarsa. It is also discovered that in some environments all of the RL algorithms produced poor results, which remains intriguing subjects of future work.

The parameters are manually tuned in the experiments, but the best combination is still not found. This can lead to sub-optimal results. How to obtain the optimal choice for the discount factor, the number of episodes and the projection size schedule remains to be solved. Grid search and random search may be used to test different combinations of parameters. On the other hand, changing the parametric function approximator may also improve the performance of the agents. Radial basis functions are utilized in the process of state-action representation, but Fourier basis functions, coarse coding, tile coding and neural networks can be examined in future studies.

The research carried out by Fard et al. (Fard et al. 2013) suggests that Bellman error based algorithms can win against viable alternatives in very large and sparse feature spaces. They have indicated two new directions for further exploration. One is to experiment with our algorithms on larger state spaces; the other is to implement more complex methods, such as L1-LSTD and L2-LSTD, for testing and comparing.

References

Achlioptas, D. 2003. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences* 66(4): 671–687.

Arriaga, R. I.; and Vempala, S. 1999. An algorithmic theory of learning: Robust concepts and random projection. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, 616–623. IEEE.

Ballard, D. H. 1999. *An introduction to natural computation*. MIT press.

Bingham, E.; and Mannila, H. 2001. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 245–250.

Cleveland, W. S.; and Devlin, S. J. 1988. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association* 83(403): 596–610.

Dasgupta, S.; and Gupta, A. 2003. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms* 22(1): 60–65.

Fard, M. M.; Grinberg, Y.; Farahmand, A.-m.; Pineau, J.; and Precup, D. 2013. Bellman error based feature generation using random projections on sparse spaces. In *Advances in Neural Information Processing Systems*, 3030–3038.

Fard, M. M.; Grinberg, Y.; Pineau, J.; and Precup, D. 2012. Compressed least-squares regression on sparse spaces. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Goldberger, J.; Hinton, G. E.; Roweis, S. T.; and Salakhutdinov, R. R. 2005. Neighbourhood components analysis. In *Advances in neural information processing systems*, 513–520.

Keller, P. W.; Mannor, S.; and Precup, D. 2006. Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, 449–456. New York, NY, USA: Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143901. URL https://doi.org/10.1145/1143844.1143901.

Li, P.; Hastie, T. J.; and Church, K. W. 2006. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 287–296.

Maillard, O.; and Munos, R. 2009. Compressed least-squares regression. In *Advances in neural information processing systems*, 1213–1221.

Menache, I.; Mannor, S.; and Shimkin, N. 2005. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research* 134(1): 215–238.

Parr, R.; Painter-Wakefield, C.; Li, L.; and Littman, M. 2007. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, 737–744.

Ruppert, D.; and Wand, M. P. 1994. Multivariate locally weighted least squares regression. *The annals of statistics* 1346–1370.

Stone, C. J. 1980. Optimal rates of convergence for nonparametric estimators. *The annals of Statistics* 1348–1360.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Vapnik, V.; Golowich, S. E.; and Smola, A. J. 1997. Support vector method for function approximation, regression estimation and signal processing. In *Advances in neural information processing systems*, 281–287.

Vert, J.; Tsuda, K.; and Schölkopf, B. 2004. *A Primer on Kernel Methods*, 35–70. Cambridge, MA, USA: MIT Press.