ECE 566 PROJECT REPORT: MARKOV DECISION PROCESS AND ITS APPLICATION IN FINANCE

Ranvir Rana, Sakshi Agarwal

ABSTRACT

Markov Decision Processes provide a formal framework for modeling tasks that involve planning under uncertainty and deriving an optimal solutions for them. They provide a framework for decision making in cases where the outcome is partially random and partially under the decision maker's control. They follow the markov property where each state is dependent only on the previous state and the action taken at that step, and not on prior history. We have explored MDPs and outlined the algorithm for determining the optimal policy in the discrete state space and then given ways to implement the same algorithm for continuous state space MDPs. In particular we have considered the consumption investment problem, implemented the algorithm for an example case and compared the optimal policy thus obtained with the expected policy derived theoretically.

1 INTRODUCTION TO MARKOV DECISION PROCESSES

A discrete time Markov Decision Model includes the following 5 elements:

$$(X, A, \{A(x)|x \in X\}, T, r)$$

where

- X and A are borel spaces in $\mathbb{R}$. X is the state space and A is the action space.
- $\{A(x)|x \in X\}$ are the non empty measurable subsets of action space where A(x) denotes the set of permissible actions when the system state is x.
- Define $\mathbb{K}$ to be the set of state action pairs as follows:

  $$\mathbb{K} = \{(x, a)|x \in X, a \in A(x)\}$$

  $T(B|x, a)$ is a stochastic kernel on X given $\mathbb{K}$, i.e. , $T(\cdot|x, a)$ is the probability measure on X for every state action pair, called the transition law.
- r is measurable function $\mathbb{K} \to \mathbb{R}$ denoting one step reward function, assumed to be upper semicontinuous in $a \in A(x)$.

The dynamics of the system work as follows: at time t if a system is in state $x_t = x \in X$, and action applied is $a_t = a \in A(x)$, then two things follow:

- a reward r(x,a) is obtained
- the next state of the system x_{t+1} is as per the transition law T
- all future rewards are discounted by a discount factor $\alpha \in [0, 1]$

A *policy* $\pi = \{\pi_t\}$ is a sequence of stochastic kernels define on A given the history of the process. It is a mapping from states to actions. Π denotes the set of all policies and $\mathbb{F}$ denotes the subset of stationary policies. A stationary policy $f \in \mathbb{F}$ is a measurable function $X \rightarrow A$ such that $f(x) \in A(x)$ for every $x \in X$. For every MDP there exists an optimal policy, which is the best option to follow for every start state.

Total discounted reward starting from state x, and using the policy π with discount factor $\alpha \in [0, 1]$ is given by:

$$V(\pi, x) := E_x^\pi[\sum_{t=0}^{\infty} \alpha^t r(x_t, a_t)]$$

A policy π^* is optimal if for all $x \in X$,

$$V(\pi^*, x) = \max_{\pi \in \Pi} v(\pi, x) = V(x)$$

The function V(x) is called the *optimal value function* for all $x \in X$ and the *optimal control problem* consists of determining the optimal policy.

One way to determine the optimal policy is to run through every policy and find the one that maximizes the value function. This is called policy iteration and is obviously a very naive approach and not a good option. The problem is solved by Value Iteration using dynamic programming approach.

2 Computing the Optimal Value Function using Value Iteration

The value iteration (VI) functions are defined as the maximum possible expected total discounted reward for starting at state x and living for n steps for each $x \in X$ and $n = 1, 2, ..$ with $V_0(\cdot) = 0$.

$$V_n(x) = \max_{a \in A(x)} [r(x, a) + \alpha \int V_{n-1}(y) T(dy|x, a)]]$$

2.1 Assumptions made

- The function u as defined below is continuous and bounded on $\mathbb{K}$ for every continuous bounded function u on X.

$$u'(x, a) := \int u(y) T(dy|x, a)$$

- For each nonnegative measurable function u on X, we define the function Hu for x in X by

$$(Hu)(x) := \sup_{a \in A(x)} \int u(y) T(dy|x, a)$$

  and $H^n u := H(H^{n-1}u)$ for n=1,2,.. with $H^0 u = u$. The assumption is :

$$R(x) := \sum_{t=0}^{\infty} \alpha^t H^t r_0(x) < \infty$$

  for every $x \in X$, where
$$r_0(x) := \sup_{a \in A(x)} |r(x, a)|$$

  For bounded r, the assumption is satisfied.

Under the above assumptions,for every n there exists a stationary policy $f_n \in \mathbb{F}$ such that the supremum in value iterations is attained. Thus for every n and $x \in X$ we have

$$V_n(x) = r(x, f_n(x)) + \alpha \int V_{n-1}(y) T(dy|x, f_n(x))$$

and f_n is called the *nth VI Decision Function*

Also define $\mathcal{R} := \{V : X \to \mathbb{R}, V measurable, |V| < R\}$

2.2 Results:

If the given assumptions hold, then:

1. The optimal value function V(x) is the unique function in $\mathcal{R}$ satisfies the *Optimality Equation*

$$V(x) = \sup_{a \in A(x)} [r(x, a) + \alpha \int V(y) T(dy|x, a)] \quad \forall x \in X$$

2. For every $x \in X, V_n(x) \to V(x)\ as\ n \to \infty$ with V_n given by the VI function.

3 Solving Continuous MDP by Discretization

The simplest way to solve a continuous state MDP is to discretize the state space and then use the value iteration or policy iteration algorithm. For eg, for a 2D state space (s_1, s_2), we can discretize into a grid where each grid cell denotes a discretized state $\bar{x}$ as shown:

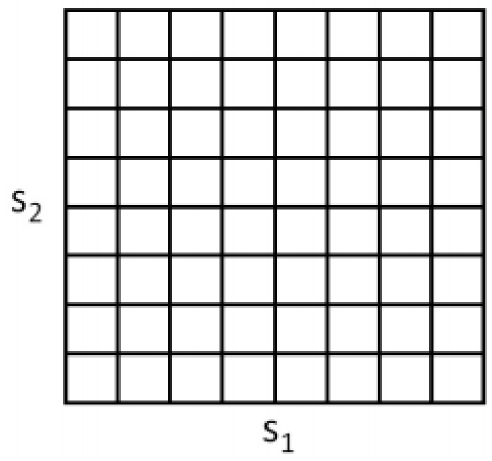

Figure 1: discretization of 2D state space

Thus we approximate the continuous state space MDP by a discrete state one as

$$(\bar{X}, A, \{A(\bar{x}|\bar{x} \in \bar{X}\}, T_{\bar{x}a}, r)$$

where $\bar{X}$ is the set of discrete states, $\{T_{\bar{x}a}\}$ are the state transition probabilities over the discrete states. Then use value iteration or policy iteration to compute $V(\bar{x})$ and $\pi * (\bar{x})$ for the discrete MDP.

When the continuous MDP is in state x and we want to compute the policy to be used, we compute the corresponding discrete state $\bar{x}$ and execute the corresponding policy $\pi * (\bar{x})$

However, discretization has a few limitations:

- It uses a fairly naive representation for V* (and $\pi *$). It assumes that the value function is constant over each discretization interval, ie. , the value function is piecewise constant over the state space grid cells.

  This piecewise constant representation does not work well for many smooth functions. Also, we would need very fine discretization to obtain a good approximation.

- The second limitation is **curse of dimensionality**, ie., suppose we have state space $X = \mathbb{R}^n$, then we discretize each of the n dimensions in to k discrete states, and the total number of discrete states we get are k^n which grows exponentially with the dimension of the continuous state space and does not scale well with large problems.

Another way to solve continuous state space MDPs is Value Function Approximation.

4 Consumption investment problem

The problem consists of an investor who wants to allocate his wealth at each time step between investment and consumption. We find the best investment policy in order to maximize his consumption utility over infinite time horizon. The consumption-investment model is defined as follows.

At each time t, current wealth x_t generates wealth $h(x_t)$,($h : [0, \infty) \to [0, \infty)$ is a production function).

A part of it a_t is consumed and the rest $i_t = h(x_t) - a_t$ is invested. It is assumed that borrowing is not allowed. Thus, both $i_t \; and \; a_t \in [0, h(x_t)]$.

The investment will lead to another wealth at time t+1. The relation between wealth and consumption will be given by:

$$x_{t+1} = \xi_t[h(x_t) - a_t]$$

where $\{\xi_t\}$ is a sequence of i.i.d. random variables taking values in $S = [0, \infty)$ independent of $x_0 \in X$ which is the initial capital stock. Consider ξ as a generic element of $\{\xi_t\}$ with density $\Delta \in C^2([0, \infty))$. It is the return per invested dollar.

The set of feasible actions for each x is given by $A(x) \in [0, h(x)]$ We also have a utility function $U : [0, \infty) \to \mathbb{R}$. The objective is to maximize the investor's consumption utility over all $\pi \in \Pi$ given by :

$$V(\pi, x) = E_x^\pi[\sum_{t=0}^{\infty} \alpha^t U(a_t)]$$

4.1 Assumptions

1. It is assumed that h satisfies the following:
   (a) $h \in C^2([0, \infty))$
   (b) h is concave on X
   (c) $h' > 0$ and h(0)=0

2. The Utility function U satisfies the following:

(a) $U \in C^2([0,\infty),\mathbb{R})$ which is strictly increasing and strictly concave
(b) U' invertible with inverse u
(c) $U'(0) = \infty \; and \; \lim_{x\to\infty} U'(x) = 0$
(d) interchange of integrals and derivatives is valid

5 Main Results about CIP

Value iteration functions are given by:

$$V_n = \max_{a\in[0,h(x)]} \{U(a) + \alpha E[V_{n-1}(\xi(h(x)-a))]\}$$

for $x \in X,\; n \geq 1,\; V_0(x) = 0$. The corresponding dynamic programming equation for CIP is given by:

$$V(x) = \max_{a\in[0,h(x)]} \{U(a) + E[V(\xi(h(x)-a))]\} \quad x \in X$$

for each $n \geq 1$, define $G^n : \hat{\mathbb{K}} \to \mathbb{R}$ as :

$$G^n(x,a) = U(a) + \alpha H^n(x,a)$$

where $H^n : \hat{\mathbb{K}} \to \mathbb{R}$ is defined as:

$$H^n(x,a) = E[V_{n-1}(\xi(h(x)-a))]$$

and,

$$\hat{\mathbb{K}} := \{(x,a) | x > 0, a \in (0,h(x))\}$$

Lemma 5.1. For the value iteration functions there exist their maximizers $\{f_n\}_{n\geq 1}$ and for $n \geq 2,\; f_n(x) \in (0,h(x)) \; with \; x > 0$. Further, $V_n \in C^2((0,\infty),\mathbb{R}) \; and \; f_n \in C^1(0,\infty)$

Lemma 5.2. The value iteration functions satisfy Euler Equation given by:

$$V_n'(x) = \alpha E[V_{n-1}'[\xi(h(x) - U(V_n'(x)/h'(x)))]\xi]h'(x)$$

for all $x \in (0,\infty)$

Lemma 5.3. For each $x > 0$, the optimal policy $f(x) \in (0,h(x))$

Lemma 5.4. For each $x \in (0,\infty)$, the optimal policy f(x) satisfies the Euler Equation given by:

$$U'(f(x)) = \alpha E[h'(\xi(h(x)-f(x)))U'(f(\xi(h(x)-f(x))))\xi]$$

Conversely, if $f \in \mathbb{F}$ is a policy which satisfies the above Euler equation for each $x \in (0,\infty)$ and if

$$\lim_{t\to\infty} \alpha^t E_x^f[h'(x_t)U'(f(x_t))x_t] = 0$$

is satisfied, then f is optimal.

6 Example 1: exponential utility

Suppose,

$$U(a_t) = \frac{b}{\gamma} a_t^\gamma$$

where $b > 0\ and\ \gamma \in (0,1)$. The production function $h(x) = x$ and

$$x_{t+1} = \xi_t(x_t^\gamma - a_t)$$

$a_t \in [0, x_t]$, t=0,1,2... and $x_0 = x \in X := [0, \infty)$ $\{\xi_t\}$ is a sequence of i.i.d. random variables independent of x $\in S := (0,1)$. Let ξ be a generic element of the sequence $\{\xi_t\}$. Suppose $\mu_\gamma := E[\xi^\gamma)] < \infty$ with $0 < \alpha\mu_\gamma < 1$. Define $\delta := (\alpha\mu_\gamma)^{1/(\gamma-1)}$ and $\Delta \in C^2((0,\infty))$ denotes density of ξ.

Lemma 6.1. For each n=1,2,...

$$V_n(x) = (\frac{\delta^{n-1}(1-\delta)}{1-\delta^n})^{\gamma-1} \frac{b}{gamma} x^\gamma$$

$\forall x \in X$

Proof by induction. Fix $x > 0$, then $V_1(x) = \frac{b}{\gamma}x^\gamma$. H^2 is given by:

$$H^2(x,a) = \frac{b}{\gamma}\mu_\gamma(x-a)^\gamma,$$

$a \in A(x). H^2 \in C^2(\hat{\mathbb{K}}, \mathbb{R}),\ H^2_{aa}(x,a) < 0$. Using Lemma 5.2,

$$V_2'(x) = (\frac{\delta(1-\delta)}{1-\delta^2})^{\gamma-1} b x^{\gamma-1}$$

Hence

$$V_2(x) = (\frac{\delta(1-\delta)}{1-\delta^2})^{\gamma-1} \frac{b}{\gamma} x^\gamma$$

since the constant of integration wil be 0 since for n=0,1,2,... $V_n(0) = 0$ for this example.

Suppose that for $n > 2, V_{n-1}$ is of the form given in the lemma, then H^n is:

$$H^n(x,a) = (\frac{\delta^{n-1}(1-\delta)}{1-\delta^n})^{\gamma-1} \frac{b\mu_\gamma}{\gamma}(x-a)^\gamma$$

$a \in A(x). H^n \in C^2(\hat{\mathbb{K}}, \mathbb{R}),\ H^n_{aa}(x,a) < 0$. Using Lemma 5.2,

$$V_n'(x) = (\frac{\delta^{n-1}(1-\delta)}{1-\delta^n})^{\gamma-1} b x^{\gamma-1}$$

Finally, the lemma follows on integrating this equation.

Lemma 6.2. For each n=1,2,..

$$f_n(x) = (\frac{\delta^{n-1}(1-\delta)}{1-\delta^n})x$$

$x \in X$

Observe that for each $x \geq 0, f_n(x) \to \tilde{f}(x)$, where

$$\tilde{f}(x) := (\frac{\delta - 1}{\delta})x$$

and $\tilde{f}(x) \in [0, x]$, thus it is an admissible stationary policy.

Lemma 6.3.

$$V(x) = (\frac{\delta - 1}{\delta})^{\gamma - 1}\frac{b}{\gamma}x^{\gamma}$$

and $\tilde{f}$ is the optimal policy.

7 Example 2: logarithmic utility

Suppose,

$$U(a_t) = \ln(a_t)$$

and production function $h(x) = x^{\gamma}$, with $\gamma \in (0, 1)$, and

$$x_{t+1} = \xi_t(x_t^{\gamma} - a_t)$$

$a_t \in [0, x_t^{\gamma}]$ and $x_0 = x \in X := [0, \infty),\ \{\xi_t\}$ is a sequence of i.i.d. random variables independent of x $\in S := (0, 1)$. Let ξ be a generic element of the sequence $\{\xi_t\}$. Suppose $\mu_{\gamma} := E[ln(\xi^{\gamma})] < \infty$ the density of ξ, Δ will be $\in C^2((0, \infty))$

Lemma 7.1. For each n=1,2,..

$$V_n(x) = \gamma k_n \ln(x) + c_n$$

where $x \in X, c_n \in \mathbb{R}\ and\ k_n = \sum_{t=0}^{n-1}(\alpha\gamma)^t,\ n = 2, 3, ...$

Proof by induction: fix $x > 0, n = 1,\ then\ v_1(x) = \gamma \ln(x)$. In this case:

$$H^2(x, a) = \gamma \ln(x^{\gamma} - a) + \mu_{\gamma}$$

$a \in A(x), H^2 \in C^2((\hat{\mathbb{K}}, \mathbb{R})),\ H^2_{aa}(x, a) < 0$, then using Lemma 5.2,

$$V_2'(x) = \alpha E[\gamma\xi/\xi(x^{\gamma} - \frac{\gamma x^{\gamma-1}}{V_2'(x)})]\gamma x^{\gamma-1}$$

which gives, $V_2'(x) = \frac{\gamma}{x}(1 + \alpha\gamma)$. Thus,

$$V_2(x) = \gamma \ln(x)(1 + \alpha\gamma) + c_2$$

Now for $n > 2$, suppose V_{n-1} has the form given above, then

$$H^n(x, a) = \gamma k_{n-1} \ln(x^{\gamma} - a) + \gamma k_{n-1}\mu_{\gamma} + c_{n-1}$$

$a \in A(x), H^n \in C^2((\hat{\mathbb{K}}, \mathbb{R})),\ H^n_{aa}(x, a) < 0$, then using Lemma 5.2,

$$V_n'(x) = \alpha E[\gamma\xi/\xi(x^{\gamma} - \frac{\gamma x^{\gamma-1}}{V_{n-1}'(x)})]\gamma x^{\gamma-1}$$

which on some calculation gives the following,

$$V_n'(x) = \frac{\gamma}{x}\sum_{t=0}^{n-1}(\alpha\gamma)^t$$

Which integrates to give the above result.

Lemma 7.2. For each n=1,2,..

$$f_n(x) = \frac{x^\gamma}{k_{n+1}},$$

$x \in X \; with \; k_{n+1} = \sum_{t=0}^{n} (\alpha\gamma)^t, n = 0, 1, 2, ..$ where

$$f_n(x) = arg \max_{a \in A(x)} \{\ln(a) + \alpha H^n(x, a)\}$$

Observe that $\forall x \geq 0, f_n(x) \to \tilde{f}(x)$, where

$$\tilde{f}(x) = x^\gamma(1 - \alpha\gamma)$$

$\tilde{f}(x) \in [0, x^\gamma]$, thus it is an admissible stationary policy. This further gives $V(\tilde{f}, x) = K \ln(x) + C \; > -\infty$, where $K, C \in \mathbb{R}$

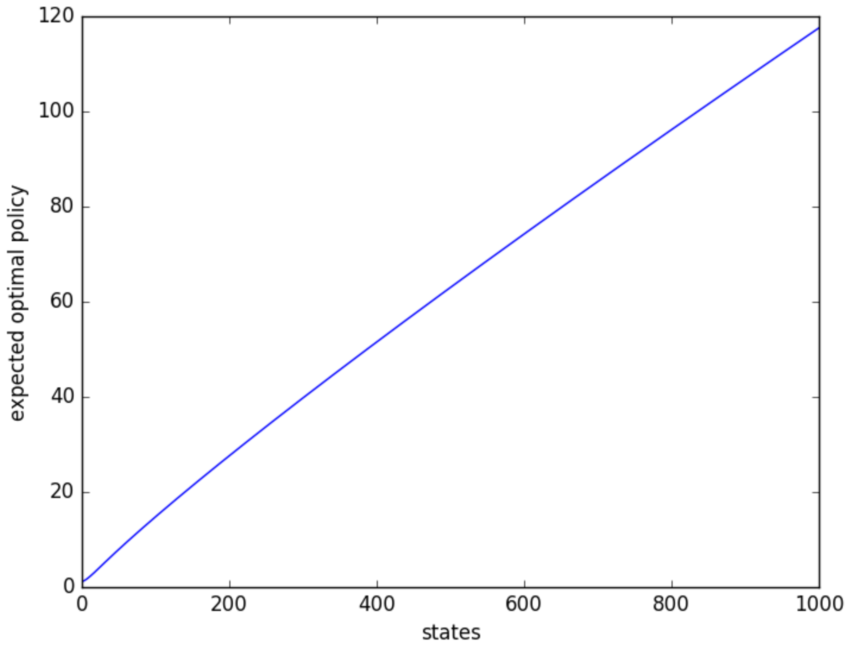

Figure 2: expected Optimal policy function with 1000 states

Lemma 7.3.

$$V(x) = \frac{\gamma}{1 - \alpha\gamma} \ln(x) + C$$

$x > 0$ where

$$C = \frac{1}{1 - \alpha}[\ln(1 - \alpha\gamma) + \frac{\alpha\gamma}{1 - \alpha\gamma}(\mu_\gamma + \ln(\alpha\gamma))]$$

and $\tilde{f}$ is the optimal policy.

8 IMPLEMENTATION OF CIP

The state and action space of the CIP are continuous. Value iteration is usually applied to discrete action and state spaces. This problem can be worked around by discretizing state and action spaces of the CIP.

We run value iteration by taking state and action states in discrete dollars, starting and \$0 and maxing out at \$1000. Thus splitting state and action spaces into 1000 discrete action and state spaces each. We run value iteration for special cases with logarithmic utility and gaussian return per invested dollar. The value iteration algorithm is as shown below.

Algorithm 1 Value iteration algorithm

```
 1: procedure COMPUTE V(S)
 2:     Initialize $V(s)$ arbitrarily
 3:     repeat
 4:         repeat(for each state s):
 5:             repeat(for each action a):
 6:                 $Q(s,a) \leftarrow R(s,a) + \alpha \sum_{s' \in S} T(s,a,s')V(s')$
 7:             until all actions are completed
 8:             $V(s) \leftarrow max_a Q(s,a)$
 9:         until all states are visited
10:     until $V(s)$ has stabilized
```

Here $R(s,a)$ is the reward for taking action a at state s. $T(s,a,s')$ is the transition probability from state s to s' while taking action a and $Q(s,a)$ is the expected long term reward of taking action a at state s.

In our implementation of the CIP we have:

$$U(a) = ln(a)$$

$$h(s) = s^{\gamma} \qquad \gamma = 0.9$$

$$T(s,a,s') \sim N(h(s) - a, (0.1 * h(s) - a)^2)$$

where $N(\mu, \sigma^2)$ represents a gaussian distribution.

The above setup stabilizes at the following value function shown in figure3.

The value function obtained matches with the logarithmic value function derived in Section 7 i.e. $V(x) = \frac{\gamma}{1-\alpha\gamma} \ln(x) + C$

The optimal policy function stabilizes to the function shown in figure4.

The expected optimal policy function is of the form kx^{γ} which will be almost linear for $\gamma = 0.9$ as seen in the plot in figure2. Thus the function the iterations stabilize to close to what we expect.

Since the state and action spaces are 1D the discretization method is expected to work well, however the matrix size required to store $Q(s,a)$ still varies as N^2 where n is the number of discrete states or actions. To work around this, there are various methods like value approximation and Q-learning which work without discretization of state space.

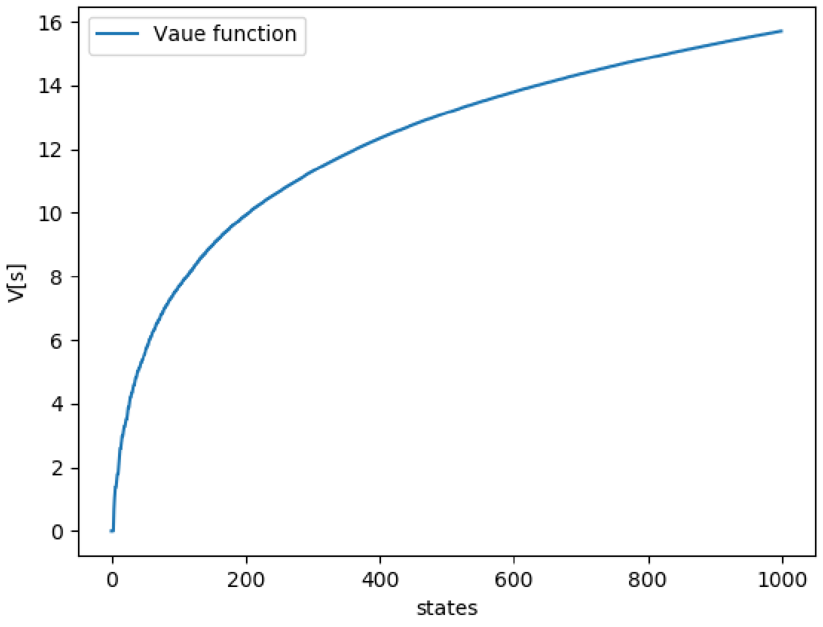

Figure 3: Value function for 1000 states

8.1 Value function approximation

Value function approximated is implemented using fitted value iteration algorithm. The main idea of fitted value iteration is to perform value function update given by the equation:

$$V_n = \max_{a \in [0, h(x)]} \{U(a) + \alpha E[V_{n-1}(\xi(h(x) - a))]\}$$

over a finite sample of states: $s^{(1)}, ..., s^{(m)}$. Supervised learning algorithms are applied to approximate value function as a linear/non-linear function of the states.

$$V(s) = \theta^T \phi(s)$$

For each state s in the sample of m states we compute $y^{(i)}$, which will be the approximation of $R(s) + \alpha max_a E_{s' \sim P_{sa}}[V(s')]$. Then we apply supervised learning to get $V(s)$ close to y.

The fitted value regression uses linear regression mostly to approximate value function but it can be replaced with other approximation methods. We used a neural network shown in figure 5 to approximate the Value function. However, unlike discrete value iteration, fitted value iteration cannot be proved to always converge. In many of our experiments the fitted value iteration lead to a non convergent neural network. We used a neural network to approximate the value function as shown in figure 6.

8.2 Q-learning

We can set the optimal utility problem as a reinforcement learning problem as the system does not know the right output i.e. the problem is not supervised. The structure of MDP is an essential part

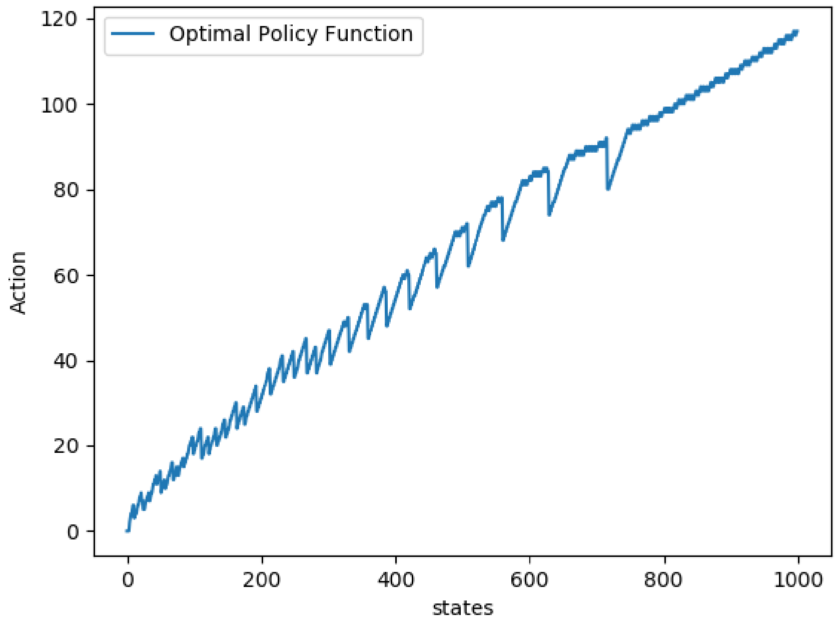

Figure 4: optimal policy function for 1000 states

Algorithm 2 Fitted Value iteration algorithm

```
1: procedure APPROXIMATE V(S)
2:     Sample m states $s^{(1)}, ..., s^{(m)}$ randomly
3:     Initialize $\theta = 0$
4:     repeat
5:         repeat(for each sampled state $s^{(i)}$):
6:             repeat(for each action a):
7:                 Sample states $s'_1, ...s'_k \sim P_{s^{(i)}a}$
8:                 $q(a) \leftarrow \frac{1}{k}\sum_{j=1}^{k} R(s^{(i)}) + \alpha V(s'_{(j)})$
9:             until all actions are completed
10:            $y(i) \leftarrow max_a q(a)$
11:        until all states are visited
12:        $\theta = argmin_\theta \frac{1}{2}\sum_{i=1}^{m}(\theta^T \phi(s^{(i)}) - y(i))^2$
13:    until $V(s)$ has stabilized
```

of a reinforcement learning problem. Here we implement Q-learning specifically Deep Q learning to solve the problem.

In the simplest form Q-learning can be defined by the one-step equation as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \gamma[r_{t+1} + \alpha \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Here we let the agent take actions according to a ϵ-greedy policy, meaning that the agent takes a greedy-step with probability $1 - \epsilon$ and a random permissible action with probability ϵ. The agent

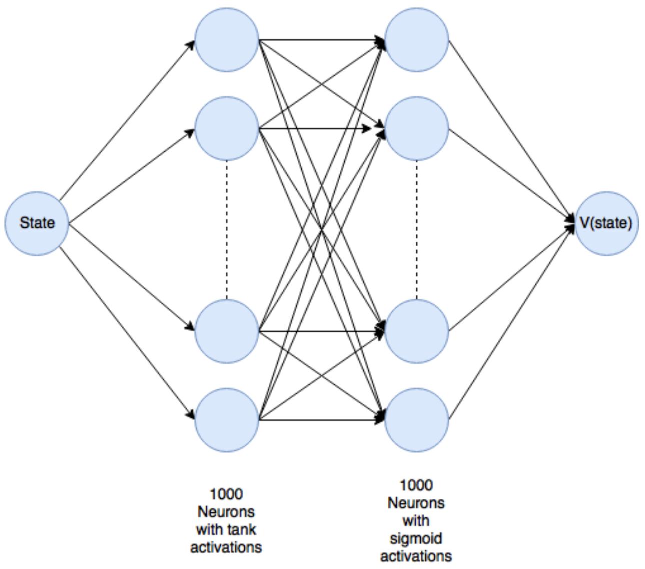
State
1000 Neurons with tank activations
1000 Neurons with sigmoid activations
V(state)

Figure 5: Neural Network trained for Value Function Approximation

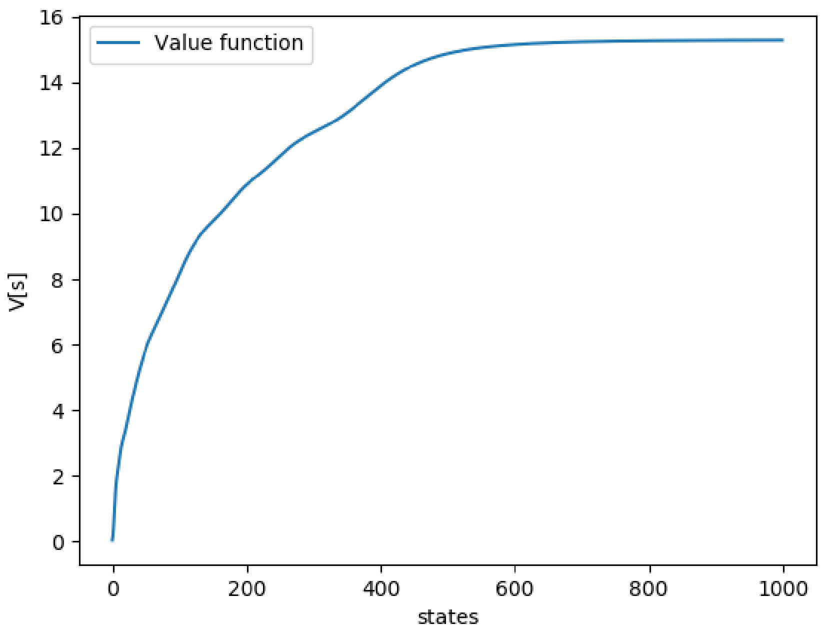

Figure 6: Value Function Approximation for log utility

runs various episodes starting with an initial sum of any value ranging from 0 to 1000 and the episode ends when the agent is bankrupt. The system tries to learn the action-value function and passes on the learned action-value function across various episodes, the learning ends when the $Q(s, a)$ stabilizes.

Algorithm 3 Q-learning algorithm

```
 1: procedure COMPUTE Q(S,A)
 2:     Initialize $Q(s, a)$ arbitrarily
 3:     repeat(for each episode):
 4:         Initialize $s$
 5:         repeat(for each step of episode):
 6:             Choose $a$ form $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
 7:             Take action$a$, observe $r$, $s'$
 8:             $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a') - Q(s, a)]$
 9:         until $s$ is terminal
10:     until No more episodes are available
```

An update to this algorithm is to change the Q(s,a) output as a neural network and training the neural network at each iteration rather than updating the action-value function. This leads to a better approximation of the Q(s,a) function. The Neural network takes takes the state x_t as input and outputs 1000 different vectors corresponding to $Q(x_t, a_k)$ for $k \in \{0, ..., 1000\}$.

Our setup uses a a 3 layer Feed Forward neutral network with first layer made up of 1000 neurons with tanh activation function, second layer with 2000 neurons with ReLU activation function and the last output layer with 1000 neurons with linear activation function.

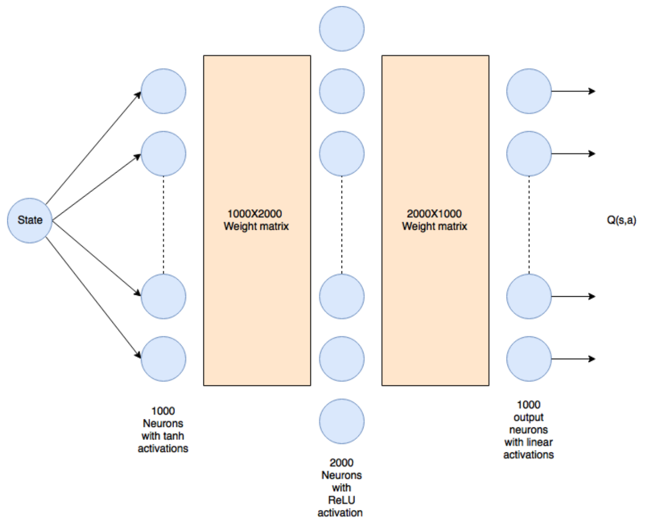
State
1000X2000 Weight matrix
2000X1000 Weight matrix
Q(s,a)
1000 Neurons with tanh activations
2000 Neurons with ReLU activation
1000 output neurons with linear activations

Figure 7: Neural Network for Q-learning

However, in our experiments, we observed that multiple configurations of Q(s,a) did not converge, leading to an incorrect approximation of the value function as shown in figure 8. This maybe because the neural network was unable to learn the constrained action space defined by $(0, h(s))$ thus leading to incorrect Value function. Moreover, Deep Q-learning based networks have proved to work only problems with huge state space and small action space like the game of chess. In our case the action space is of the order of the state space(1000 discrete actions in our example), thus the neural network was unable to learn outputs for so many different actions.

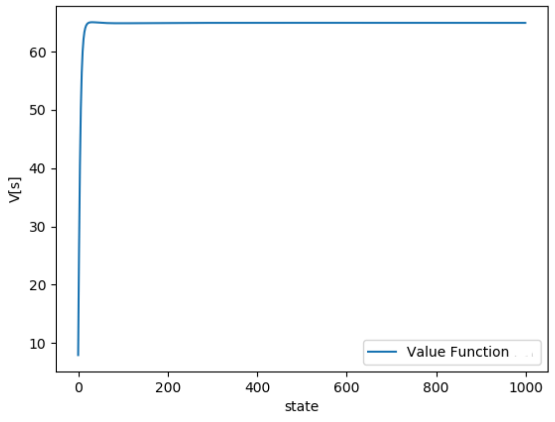

Figure 8: Incorrect Value Function by Q learning

Algorithm 4 Algorithm to train neural network for Q-learning

```
1: procedure TRAINING
2:     Initialize $Q$ network arbitrarily
3:     function EVALUATE($s$)
Require: state $s$
Ensure: vector $t$, where $t[i] = Q(s, i)$, $i \in A$
4:         Run $Q$ network forward with state $s$ as input
5:         Store output of $Q$ network in $t$
6:         return $t$
7:     function TRAIN($s$, $target$)
Require: state $s$, vector $target$
Ensure: Network $Q$
8:         $output \leftarrow$ Evaluate($s$)
9:         Update network $Q$ based on Error($output$, $target$)
10:        return $Q$
11:    repeat(for each episode):
12:        $s \leftarrow$ initial state
13:        $target \leftarrow$ Evaluate($s$)
14:        repeat(for each step of episode):
15:            Choose $a$ as $argmax_i\ target[i]$ following $\epsilon$-greedy policy
16:            Take action $a$, observe reward $r$ and new state $s'$
17:            $t \leftarrow$Evaluate($s'$)
18:            $maxQ \leftarrow argmax_i\ t[i]$
19:            $target[a] \leftarrow r + \gamma \cdot maxQ$ where $\gamma$ is discount factor
20:            Train($s$, $target$)
21:        until $s$ is terminal
22:    until No more episodes are available
```

Note that the action are still discrete, this can be avoided by taking (x_t, a) as input to the neural network and taking one output off the neural network as $Q(s, a)$ however, taking the most optimal policy will lead to evaluating the neural network for infinite different actions to get the optimal action, ultimately that would lead to evaluating the neural network over finite discrete actions and would involve multiple forward passes to the neural network, thus leading to a much slower algorithm.

9 Conclusion

Consumption investment Problem can be modeled as a continuous MDP with 1 dimensional action and state space. Discretization methods work well for this problem as the state and action space have low dimension. Value function approximation work well to for this problem with continuous state space and discrete action space. However, contrary to expectation, Q-learning based algorithms work poorly as there are high number of actions and the neural network fails to learn the action-value function.

10 References

1) An stochastic consumptioninvestment problem with unbounded utility function - *Heliodoro D. CruzSuarez*
http://www.morfismos.cinvestav.mx/Portals/morfismos/SiteDocs/Articulos/Volumen4/No1/Suarez/helio.pdf

2)A consumption-investment problem modelled as a discounted Markov decision process- *Hugo Cruz-Suarez, Raul Montes-de-Oca, Gabriel Zacarias Espinoza*

3) Continuous MDPs
https://www.cs.utah.edu/ piyush/teaching/continuous-mdp.pdf

4) Q-learning with Neural Networks
http://outlace.com/Reinforcement-Learning-Part-3/

5)Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, MIT Press, 1998, http://webdocs.cs.ualberta.ca/ sutton/book/ebook/

6)Demystifying Deep Reinforcement Learning https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

7) Keras-Deep learning library
https://keras.io