Final Assignment (Project) Report

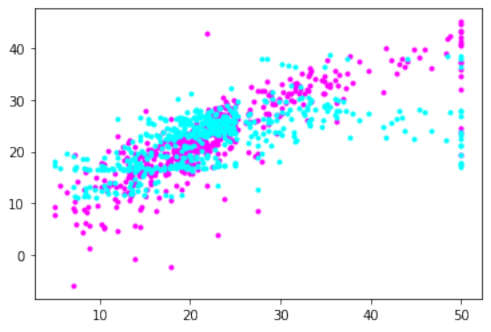
YANG LI, 31319653, yl13y19@soton.ac.uk

June 12, 2020

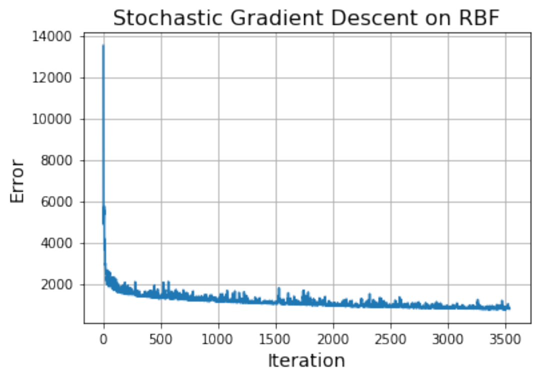
1 Radial Basis Functions

1.1 Solve the house problem by gradient descent

The result of append code show in Fig. 1(a). The loss of linear regression of pseudo-inverse is 110 and the loss of RBF is 168. The loss of RBF using stochastic gradient descent is 786.



(a)



(b)

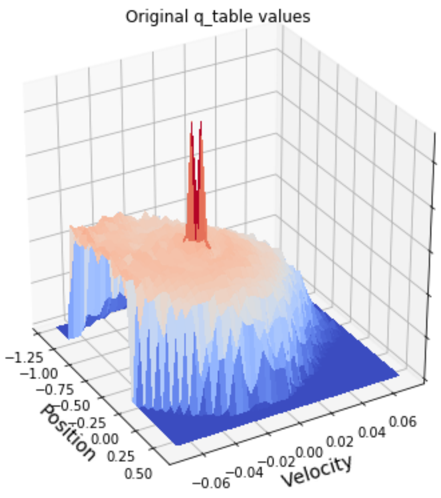
Figure 1: The comparison of linear regression and RBF is shown in Figure 1(a). The RBF convergence curve using stochastic gradient descent is shown in Fig. 1(b)

1.2 Mountain Car

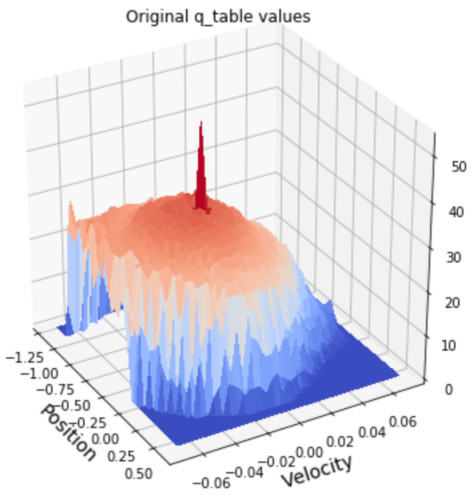
1.2.1 Value function learned by the tabular

According to the document of gym, rewards of most situations are -1. When I draw the value of 3D picture , I set the value of Z to be negative. The value of original table values are shown in Fig. 2.

I used three sets of RBF to simulate the values of the three actions in Qtable, and using RBF can make the car reach the goal. In order to measure the accuracy, I use Qtable as the criterion for each action selection (according to Qtable to select the next execution action). Record the number of times the Qtable selection action is consistent with the RBF model selection action and the total number of selections to calculate the accuracy. The accuracy of the RBF model is shown in Figure 3. It can be seen that the accuracy rate is positively correlated with the number of functions.



(a)



(b)

Figure 2: After 10 times to reach the goal, the original Qtable values is shown in Fig. 2(a). After 40 times to reach the goal, the original Qtable values is shown in Fig. 2(b)

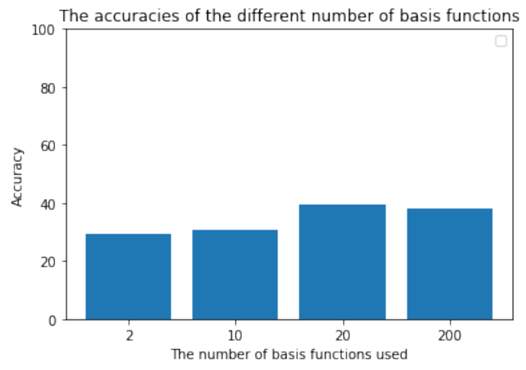
I select the maximum value of every situation (potion and velocity) as the value of RBF. The value function is show in Fig. 4. The greater the number of RBF functions, the more similar the simulated results.

1.2.2 RBF approximation on-line

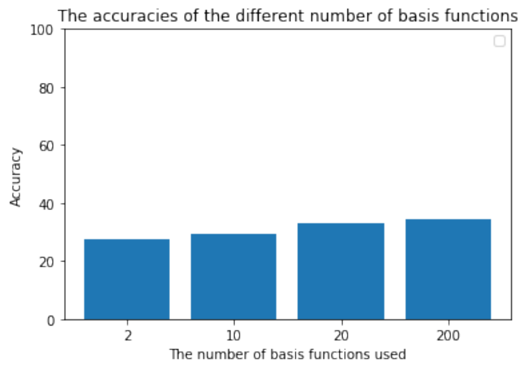
Updating W directly online instead of using Tabular can make the value function smoother which avoid the problem of discontinuous function values. The equation used is shown below:

$$\omega_{t+1} = \omega_t + \alpha(\upsilon_\pi(S_t) - \upsilon(S_t, \omega_t)) \bigtriangledown \upsilon(S_t, \omega_t)$$

The result of the RBF approximation on-line is shown in Fig. 5. Compare the RBF value function of on-line and it based on tabular, I find that the on-line result is better than the tabular. The change of value is more obvious. I think the reason is that on-line training avoid the Qtable value discontinuities. However, the on-line RBF value function generation speed is slower than the tabular one.



(a)



(b)

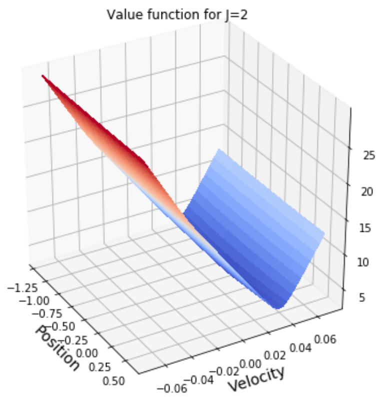
Figure 3: After 10 times to reach the goal, the accuracies with different numbers of basis functions are shown in Fig. 3(a). After 40 times to reach the goal, the accuracies with different numbers of basis functions are shown in Fig. 3(b).

1.3 Resource Allocation Network (RAN)

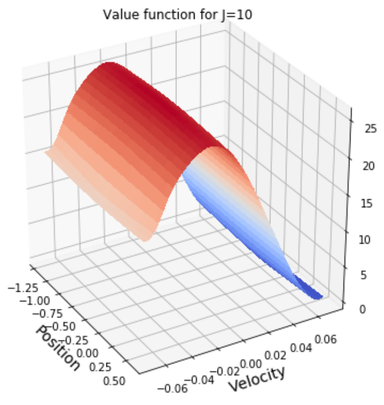
In order to allocate the number of RBF functions according to the complexity of the problem, RAN is used to automatically adjust the center of RBF and the number of functions. In the problem of mountain car, it can use RAN instead of K-means to generate the center of RBF.

The pseudocode show in below:

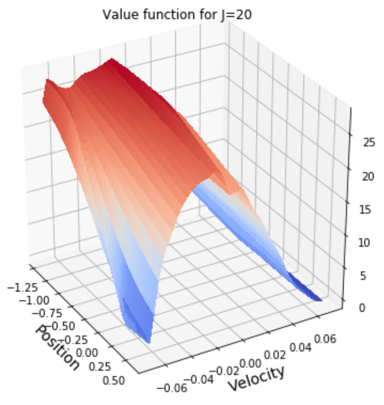
```
$\delta = \delta_{max}$
Randomly generate centers (c)
$\gamma = T_0$ (I is the input of position and velocity, T is the output of network of RAN)
loop over presentations of input-output pairs(I,T){
    evaluate of network $y = \sum_j h_j x_j(I) + \gamma$
    computer error $E = T - y$
    find distance to nearest center $d = min_j \|c_j - I\|$
    if $\|E\| > \epsilon$ and $d > \delta$ then
        allocate new unit, $c_{new} = I, h_{new} = E$
        if this is the first unit to be allocated then
            width of new unit $= k\delta$
        else
            width of new unit $= kd$
        use new center $c_{new}$ to generate RBF of Q-learning
    else
        perform gradient descent on $\delta$ , $h_j$ , $c_{jk}$
    if $\delta > \delta_{min}$
        $\delta = \delta \times exp(-1/\tau)$
```



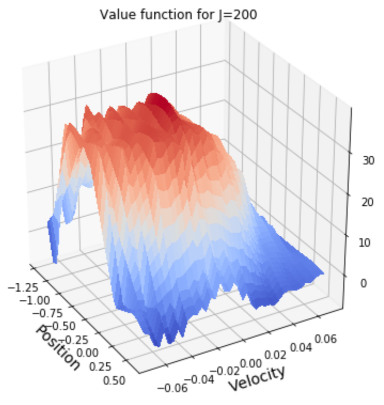
(a)



(b)

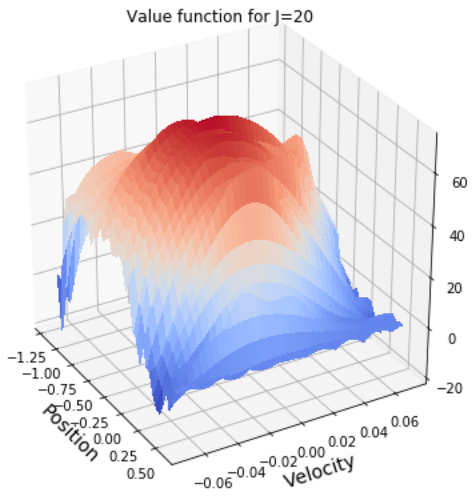


(c)



(d)

Figure 4: RBF value functions of different number of functions. The function number J is 2 shown in a, J is 10 shown in b, J is 20 shown in c and J is 200 shown in d



(a)

Figure 5: RBF value functions of J is 20 for on-line

2 Summarising a recent scientific study (Human-level control through deep reinforcement learning)

2.1 Introduction

The test domain is classic Atari 2600 games. Agents which use reinforcement learning are confronted with a difficult task that use high-dimensional sensory inputs to generate past experience to new situations. This paper uses convolutional neural networks to process image data and combines with Q-network to realize the processing of high-dimensional data and the training of agents.

For Atari 2600 games, inputs are pixels of game screen and game score, which is suitable for reinforcement learning. The state can be obtained through pixels, the reward can be obtained through the game score, and the action is determined by the game itself.

2.2 States, rewards, state transition and policy models

2.2.1 States and State transition

The raw data are $210 * 160$ pixel images with a 128-colour palette. When preprocessing, writer take the maximum value for each pixel colour value over the frame and the previous frame which removes flickering. Then, according to luminance, writer rescale the image to $84 * 84$. Last, stack 4 most recent frames. This three steps is called ϕ in this paper. The original image is x_1, $s_1 = \{x_1\}$, and the state is s (sequence) after processing ($\phi(s)$).

2.2.2 Rewards

Writer clipped all positive rewards at 1, all negative rewards at -1 and unchanged rewards at 0 because of different score ranges in different games. So that the same learning rate can be used in different games.

2.2.3 Policy models

The agent in this paper selects and executes actions according to ε-greedy policy based on Q which is off-policy.

2.3 Learning paradigm

The difficulty of model learning is that when nonlinear fitting is used, the problem of divergence and non-convergence will occur. There are two ways to solve in this paper. First, use experience replay to select some random data which reduce the correlation of frames. Second, increasing the update cycle of Q does not update every frame, which is shown in algorithm 1 in Fig. 6.

In each episode, according to the probability ε, select a random action or the largest Q value action. Use the above actions to get the corresponding reward and the next image, then the next state will process 4 frames of image to get a new network input. Store the data (previous state, action, reward, next state) which is processed by ϕ to replay memory D. Randomly select a stored transformation data from replay memory D to train the network. If the next state game is over, then the action-value is the reward; if the game is not over,

```
**Algorithm 1: deep Q-learning with experience replay.**
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t$ = 1,T do
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Figure 6: Deep Q-learning with experience replay.

then enter the next processed state into the network to get the target. According to the loss function, the parameters in network are updated through stochastic gradient descent (SGD). The loss function is shown in Fig. 7.

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a,r}\left[\left(\mathrm{E}_{s'}[y|s,a] - Q(s,a;\theta_i)\right)^2\right] \\ &= \mathbb{E}_{s,a,r,s'}\left[\left(y - Q(s,a;\theta_i)\right)^2\right] + \mathrm{E}_{s,a,r}\left[\mathrm{V}_{s'}[y]\right]: \end{aligned}$$

Figure 7: The loss function.

2.4 Conclusion

Without prior experience,Deep Q-network (DQN) performs better than linear learners in most games, what's more, DQN outperforms humans in 49 games. However, during training, average score per episode fluctuate greatly.

Use the t-SNE algorithm to visualize high-dimensional data and display the last hidden layer of DQN. Taking Space Invaders as an example, it shows that similar states will be placed close to each other and sometimes the state may not be similar, but the expected rewards are similar.

DQN successfully applied high-dimensional data (image data) in different environments (game in Atari), indicating that DQN has good universality. Adopt the end-to-end training method, use CNN instead of handcrafted features, so as to achieve no need to manually extract features. DQN solves the problem of correlation and non-static distribution by establishing an experience pool, storing experience, and randomly taking samples for training during training.

I think this paper provides a reference for the agent to process high-dimensional data for the state. High-dimensional data is extended to other forms such as sound and position, or different forms of data are combined to form states.