Reinforcement Learning With Deeping Learning in Pacman

Shuhui Qu, Tian Tan, and Zhihao Zheng
shuhuiq@stanford.edu, tiantan@stanford.edu, zhihaoz@stanford.edu

Abstract— A new method to approximate the true value in reinforcement learning by using deep neural network is proposed. We simulated the Pacman by using this method.

Keywords—reinforcement learning; deep learning; Q-learning;

I. INTRODUCTION:

Nowadays, the machine learning has become a hot research area with a lot of different algorithms. Reinforcement learning, different from most machine learning methods, requires learning agent to acquire its own input data that repeatedly choose an action to the dynamic environment, and get some rewards and information about the action feedbacks. Due to the limitation of human experience and current technology, collected data usually cannot cover all highly related features, therefore, the learning agent should be able to explore data features and possible hidden features by itself and gradually improve its behavior by considering the consequences of its past actions.

Currently, there are several methods for reinforcement learning, including model-based Monte Carlo method, SARSA (bootstrapping methods), Q-learning, as well as Q-learning with function approximation. Linear regression, SVM and other methods have been applied to function approximation for Q learning. However, in real world, some data such as spacial and time series data may not have Markov properties. Considering the fact that in deep learning, creating architecture of hierarchical feature representations that become more and more abstract for each additional layer of the architecture, thus, learns features, hidden features could be generated during the process.

This paper proposes an algorithm that applies deep neural network and deep learning method for function approximation so that hidden features as well as weights for each feature could be derived to achieve optimal Q-function. First of all, the relations between supervised learning and reinforcement learning will be derived. Secondly, deep learning will be introduced to find optimal Q-functions. Next, a simulation of Pacman will be applied to compare the method with other reinforcement learning algorithms. Finally, conclusions and discussions of potential future work will be provided.

II. LITERATURE REVIEW:

J. Schmidhuber [1]studied the applicability of deep learning for feature learning in combination with reinforcement learning. He developed a new learning approach using stacked autoencoders as feature learning method. S. Dini and M. Serrano [2] implemented Q-learning by using a q-table to store data, with increasing complexity in the environment and the agent, which will cause fail to scale in this approach. In order to avoid this problem, they investigated an alternative implementation in which use an artificial neural network as a function approximator and eliminate the need for an explicit table.

J. Telecom [3] approach to the problem of autonomous mobile robot obstacle avoidance using reinforcement learning neural network. In the project, the author integrated these two methods aiming to ensure that autonomous robot behavior in complicated unpredictable environment. The result shows that the combination of reinforcement learning and neural network can improve learning ability and also make robot finish tasks under more complicated environment.

III. THEORY:

A. Relation Between Supervised Learning and Reinforcement Learning

1) Supervised Learning:

Given a training set, try to learn a function h: x-> y, so that h(x) could predict y values corresponding to new input of x.

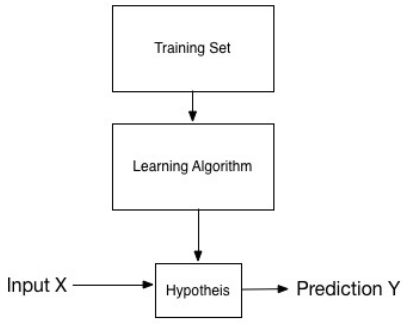

fig 1 supervise learning model

In order to make values of y to be as close to the hypothesis of h(x) as possible, a cost function is defined as:

$$J = \Sigma \frac{1}{2} \|h_\theta(x) - \varphi\|_2^2$$

where φ is value of y for a given example in the training set. Considering the fact that in some cases it is not clear to find the hypothesis function exactly we want, we add on regularization term to make the algorithm less sensitive to outliers.

$$J = \Sigma \frac{1}{2} \|h_\theta(x) - \varphi\|_2^2 + \frac{1}{2\eta} \|\theta\|_2^2$$

In order to minimize the value of cost function, we take derivation of J as a function of θ, and set the value to be zero.

$$\nabla(J(\theta) + \frac{1}{2\eta}\|\theta\|_2^2) = 0$$

By deriving it, we could have the value of k+1th iteration of θ (gradient descent):

$$\theta^{(k+1)} = \theta^{(k)} - \eta\nabla_\theta f$$

2) Reinforcement Learning:

In reinforcement learning, each state is determined by previous state, action that it takes and some random noise. The process could be represented by:

$$X_0 \xrightarrow{a_0} X_1 \xrightarrow{a_1} X_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} X_4$$

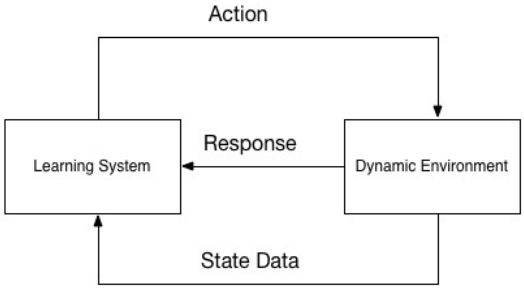

fig 2 reinforcement learning model

$$X_{t+1} = f(x_t, a_t, w_t)$$

Suppose that everything happens with reason and logic, each state could be mapped to next state with the consideration of action and random noise. Where X is the state, a is action that being taken under current state, w is random noise factor.

$$Q^\pi(X, a) = r + \gamma \sum_{x' \in X} T_{x\pi(x)}(X, a, X')[V^\pi(X')]$$

We also define the optimal value function as:

$$V^*(X) = \max_\pi x\ Q^\pi(X)$$

Therefore, we could define the update function of V as:

$$V = TV$$

Q function could be updated in the following format:

$$Q(x, a) = r(x, a) + \gamma \sum_{x' \in X} T(x'|x, a)V^\pi(x')$$

$$Q(x, a) = r(x, a) + \gamma \sum_{x' \in X} T(x'|x, a)\min Q(x', a')$$

$$V(x) = \{r(x, a) + \gamma \Sigma T(x'|x, a)V(x')\}$$

Optimal value function update

$$V^* = T^*V^*$$

Initiate:

$$V^{(0)} = 0 \Leftrightarrow Q^{(0)} = 0$$

in each iteration, we could have:

$$V^{(k+1)} = T^*V^{(k)} \Leftrightarrow Q^{(k+1)} = F * Q^{(k)}$$

Q-function update

$$Q^{(0)} = 0$$

$$\Delta Q^{(k)} = Q^{(k+1)} - Q^{(k)} = FQ^{(k)} - Q^{(k)}$$

$$Q^{(k+1)} = Q^{(k)} + \eta\Delta Q^{(k)}$$

$$Q_{opt}(x, a) = Q_{opt}(x, a) - \eta[Q_{opt}(x, a) - (r(x, a) + \gamma V(x'))]$$

where η is the transition probability.

Therefore, we can have:

$$err = Q_{opt}(x, a) - (r(x, a) + \gamma V(x'))$$

where Qopt(x,a) is the prediction, while r(x,a) + $\gamma V(x')$ is our target, which is denoted by x'. Here, we could find that it is similar to supervise learning.

$$Q_{opt}(x, a, \rho) = \rho(DNN(x, a))$$

Consequently, the cost function could be

$$min_\rho \sum_{(x,a,r,x\prime)} (Q_{opt}(x, a, \rho) - (r(x, a) + \gamma V(x')))^2$$

$$\rho = \rho - \eta(Q_{opt}(x, a, \rho) - (r(x, a) + \gamma V(x')))\nabla_\rho Q_{opt}(x, a, \rho)$$

Where ρ is the set of parameters in hypothesis learning function. Since we will use deep neural networks (DNN) in our model. ρ refers to a set of all parameters in DNN.

B. Q-Learning with Deep Learning

1) Reinforcement learning with deep neural network architecture

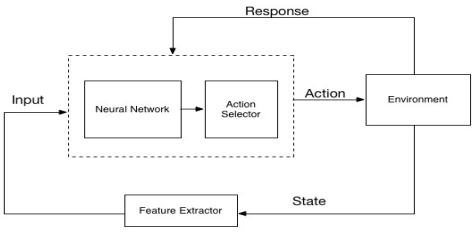

fig 3 Architecture of Q-learning with DL

Figure 3 shows the structure of the learning system with deep neural network. Initially, the network estimates the data roughly. After interacting with the environment and getting rewards, the estimation of value would converge to true value and noise of the network could be minimized.

2) Deep Neural Network

The methods this paper discussed above (Q-learning, Q-learning with linear function approximation, etc.) have poor performance when collected data do not have Markov property or have many hidden features that could not be computed by superficial data. Neural network sometimes could give relative good prediction as it could also use one hidden layer. However, neural network is shallow network, features (the hidden layer of activation a) are computed by using only one layer. Deep neural network has multiple hidden layers. This allows us to compute much more complex features of superficial input. Because each hidden layer computes a non-linear transformation of the previous layer, a deep network can have significantly greater representational

power than others. By using a deep network, it could contour and detect complex features.

3) Learning method [4]

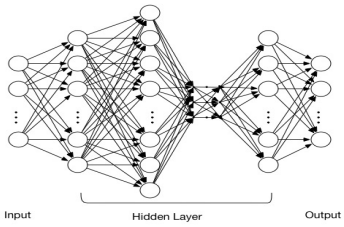

fig 4 Deep Neural Network

In our deep neural network model, we have the input layer l as L_l, and use layer nl as output layer (nl is the number of total layers). Parameters for the network (W, b) = ($W^{(1)}$, $b^{(1)}$, $W^{(2)}$, $b^{(2)}$, …, $W^{(nl-1)}$, $b^{(nl-1)}$), where $W_{ij}^{(l)}$ denotes the weight for the connection between unit j in layer l and unit i in layer l+1.

The Activation (output value) of unit i in layer l is represented by $a_i^{(l)}$. Computation of activations is given by:

$$a_i^{(l)} = f\left(\sum_{j=1}^{n} W_{ij}^{l-1} x_j + b_i^{l-1}\right)$$

Where f is sigmoid function (used in our model) and n denotes total number of units in layer (l-1), x_j's are inputs to unit i in layer l. The output of the network can be represented as:

$$h_{W,b}(x) = a_1^{(nl)} = f(\sum_{j=1}^{n} W_{1j}^{(nl-1)} x_j + b_1^{(nl-1)})$$

In the paper, we use back propagation algorithm to train our DNN model:

First, define the cost function with respect to one single training example (x, y) to be:

$$J_{W,b}(x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

Thus, if we also denote s_l to be the number of nodes in layer l (not counting the bias unit), then overall cost function with respect to the whole training set can be written as:

$$J_{W,b} = [\frac{1}{m}\sum_{i=1}^{m} J_{W,b}(x,y)] + \frac{\lambda}{2}\sum_{l=1}^{nl-1}\sum_{i=1}^{sl}\sum_{j=1}^{sl+1}\left(W_{ji}^{(l)}\right)^2$$

where the second term is a regularization term, and λ is weight decay parameter.

So, the optimization problem now becomes minimization of the above overall cost function as a function of W and b. We initialize parameters $W^{(l)}$'s and $b^{(l)}$'s to small random values near zero and then apply batch gradient descent. One iteration updates W, b as follows:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}} J_{W,b}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}} J_{W,b}$$

where α is the learning rate, and we use back-propagation to compute above partial derivatives.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J_{W,b} = \{\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ij}^{(l)}} J_{W,b}(x^{(i)}, y^{(i)})\} + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J_{W,b} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_i^{(l)}} J_{W,b}(x^{(i)}, y^{(i)})$$

Thus, we can update the network with batch gradient descent. In this paper, we also update the network in a mini-batch of data in certain number of iterations.

4) Action Selection

We use greedy action selection. When Q(x,a) converge to Q*, then we select the policy by $\pi(x) = argmax_{a' \in A} Q(x,a)$.

5) Learning procedure for the Q-learning algorithm with deep learning

Step1: Initialize the deep neural network with n levels with non-linear activation function (combination of linear activation is still linear activation)

Step2: Input state data, and extract as many features according to experience as possible

Step3: Select action according to Q(x,a) for a∈A where A is action set

Step4: Environment performs the action and response rewards and states

Step5: Update Q-function according to

$$\rho' = \rho - \eta(Q_{opt}(x,a,\rho) - (r(x,a) + \gamma V(x')))\nabla_\rho Q_{opt}(x,a,\rho)$$

Where $Q_{opt}(x,a,\rho) = \rho(DNN(x,a))$, ρ is a parameter set contains all parameters in DNN

Step6: Repeat Step2 to Step5 until converge

IV. SIMULATION:

To test the performance of the combination of Q-learning and deep learning, an example of Pacman is applied in the game.

Pros:

- The Pacman is easy to implement and open to public
- The state space and the action space of this game are discrete and could easily define each state

Cons:

- The state of Pacman seems to have Markov property, each decision could be made based on current conditions without consideration of past steps. Therefore, this is not a perfect example for deep learning, because the layers might be shallow and only a few neutrons in each layer would be required to solve the problem.
- The game is deterministic; The Q-learning with function approximation could give us good result.

A. Introduction of Simulation Program

State: a state s consists of the map, Pacman's location, ghosts' location, remaining food's location and current score.

Action: there are five actions: go left, right, up, down and stop. However, the Pacman might be blocked if it encountered a wall.

Reward: If a ghost eats the Pacman, the Pacman gains -500 points and game ends. If he gets a dot, he gains 10 points. If he successfully eats all dots, he gains extra 500 points and he

wins the game. If he eats a ghost, he would gain 200 points. Any step would cost -1 points.

B. Method we use

In order to evaluate the performance of Q-learning with deep learning, we need to set up some benchmarks and compare with other methods. Q learning, approximation with linear regression method and Q-learning with deep learning (our proposed method) were applied. We could also play Pacman by using game method. Therefore, minimax and expectimax method were used to compare with the performance of reinforcement learning.

1) Data

For minimax and expectimax method, there is no requirements for data.

However, for Q-learning and the other two approximation methods, data is gathered by running simulation.

2) Features

There are some basic feature inputs: distance from closet food, whether ghost is close, whether to eat food, whether ghost is eatable, number of food remaining and others.

For deep learning, there are hidden features to be discovered. However, the number of hidden features or perceptron needs to be carefully selected in order to reach optimal result.

3) Environments

Different methods performed differently under different circumstances. Two maps were implemented in the simulation. Small map:

fig 5 small map

Medium map:

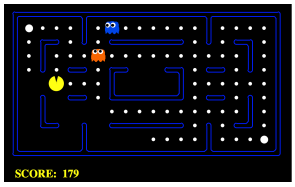

fig 6 medium map

4) Structure of Deep Neural Network

The structure of deep neural network might influence the score that could be obtained in the game.

There are two parameters that could be tuned: the number of layers of network, the number of perceptron in each layer.

Due to the fact that Pacman works in a non-complex pattern, the number of layers and number of perceptron should be restricted in order to avoid over fitting. Therefore, we would set the structure of the network as input layer-20-20-output layer, input layer-12-12-output layer, input layer-20-output layer, where 20 is the number of perceptron in that layer.

5) Deep learning implementation method

Deep learning is implemented as:

We explore the map and gather data by running approximation method. Then these data was trained by using deep neural network. After gathering last layer before the output, we plug this layer back to predict Q-Value and update parameter of this layer.

C. Result Comparison

Parameters for simulation: iteration is 3000, the learning rate is 0.2, exploration rate is 0.05, and discount factor is 0.8.

As we can see in the chart, Q-learning did not perform well in the medium map. Approximation method converged quickly during the learning process and can reach score around 1300.

Deep learning method converged after 2000 iterations. It did not outperform the approximation method during the simulation. (horizontal axis represents number of iterations in hundred, e.g. 22 represents 2200 iterations)

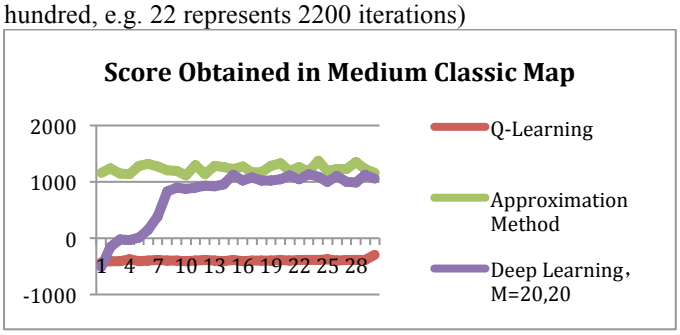

fig 7 score in medium map

Algorithm	Average Score
Q-Learning	-478
Minimax	240.39
Expectimax	1479.86
Approximation with linear regression	1812.75
Approximation with Deep Learning	1746.78

table 1 running score withour exploration

After setting exploration probability to 0, 10 more simulations were run to test the performance of each method whose result is shown in the table

The score of small map is shown as follow. Q-learning's performance is improved compared with medium map. Q-learning with approximation perform well in the game. Q-learning with deep learning does not perform well under 2000 iterations. However, it improved a lot after 8000 iterations. The reason behind this could be that Q-learning with deep learning requires enough data to converge and make right decisions. In a small map, the amount of data can be obtained in each iteration is much smaller than in the medium map. Therefore, it requires more iteration, about 8000, for deep neural nets to converge and improve performances.

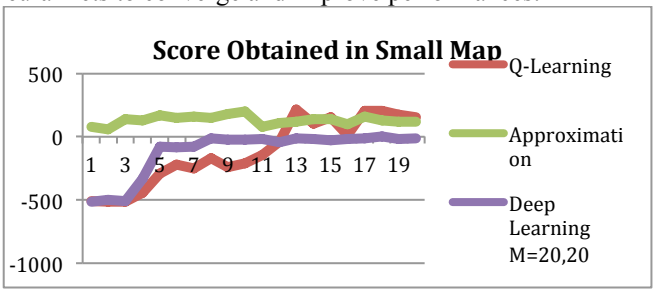

fig 8 Score obtained in small map

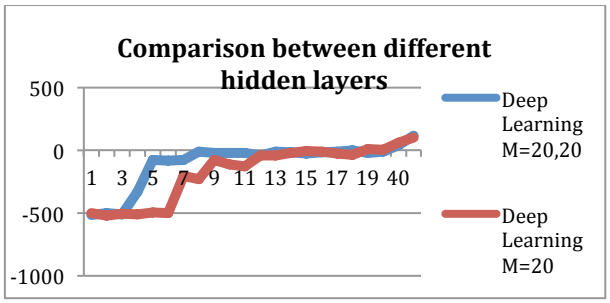

fig 9 Comparison between No. Layers

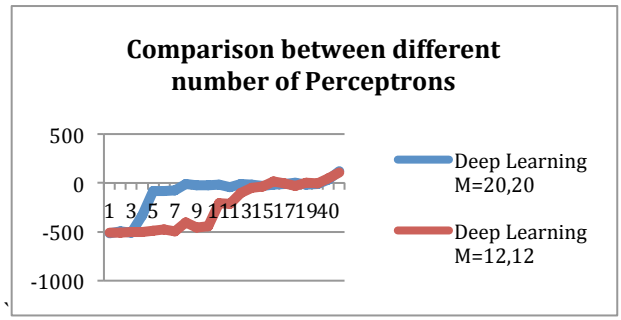

fig 10 Comparison between No. Perceptron

This paper also investigated the structure of deep neural network as shown in fig9 and fig10. The network with more layers converges quicker than the one with fewer layers. The network with more perceptron in each layer converges quicker than the one with less perceptron.

D. Discussion

From previous results we can discover that:

- The Pacman problem fits better by using Q-learning with linear regression approximation.
- During the construction of deep networks, it is discovered that parameters dissipated quickly as the number of layers and perceptron increase.
- The computational cost of deep learning is large, that it takes longer for deep learning to calculate parameters and update values.
- For simple problems, linear regression model performs better than more complex models.
- Compared with linear regression, the deep learning method takes more iterations to converge.
- Deep learning method requires much data input.
- For simple problems, the more layers and perceptron a network has, the faster it converges. However, these numbers should be carefully selected to avoid dissipation.

V. CONCLUSION:

In this paper, we tried to apply mature supervised learning methods to reinforcement learning for the purpose of helping solve complex non-Markov problems in daily life. In particular, we combined deep learning (DNN) with Q-learning and proposed a new learning method for reinforcement learning (RL). At the beginning, we investigated reinforcement learning in details and found similarities between RL and supervised learning, which proved that indeed DNN could possibly be used in Q-learning. Then we constructed our new model by using DNN. Finally, to test the performances of our model and compare with other common methods, we used the game of Pacman as a platform and ran simulations on it.

From the results shown in previous sections, it seems that deep neural nets require large amount of data to converge. Thus, our model, Q-learning with DNN (deep learning), performed better and converged much faster in medium map than small map.

Compared with other methods, our model in general performed better than Q learning. However, it did not outperform Q-learning with linear approximation. For the medium map, our model tied Q-learning with linear approximation in performance (scores), but in the small map, linear approximation is better. The game of Pacman may be a relatively simple problem and almost deterministic. Thus, linear approximation performed quite well for this specific problem. While using our model, it took longer to converge and the Pacman in the game seems to be more hesitated when making next decisions or actions – "think too much" as our model takes into account more features and hidden features. The overall results were not as good as linear approximation.

We also found out that the structure of the DNN matters. Parameters dissipated quickly as the number of layers and perceptron increase. Thus, each layer needs to be subtly designed. In our simulation, the network with two hidden layers and 20 neurons in each layer performed well.

The computational cost of our proposed model with DNN is a major concern. For future work, we will strive for and focus on reduction of computation costs. It may help in the future, if we could design a better set of input features or design in great details the structure of DNN. Designing the structure of the net in a way that state features and actions features can be separated in hidden layers may help improve the performance of Q-learning with deep learning model.

REFERENCES

[1] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," p. 75, Apr. 2014.

[2] S. Dini and M. Serrano, "Combining Q-Learning with Artificial Neural Networks in an Adaptive Light Seeking Robot Q-Learning," no. fig 1, 2012.

[3] J. Telecom, "REINFORCEMENT LEARNING NEURAL NETWORK TO THE PROBLEM OF AUTONOMOUS MOBILE ROBOT OBSTACLE AVOIDANCE," no. August, pp. 18–21, 2005.