Chapter 1

Reinforcement Learning

The neural network structures previously discussed for the Boston Housing data are essentially curve fitting models, with continuous inputs and output. The character recognition networks for the MNIST dataset can also be though of as an abstract curve fitting model with multiple outputs, where the result is a probability value.

In both cases, training is performed on a set of inputs with known target outputs.

Not all problems can be constructed in this manner as some problems require discrete actions to be taken. Similarly, problems may contain reward systems. Such systems may have delayed rewards, were one action provides an immediate reward, but prevents reaching a state of higher reward.

Consider the "Chain" problem from Strens, 2000:

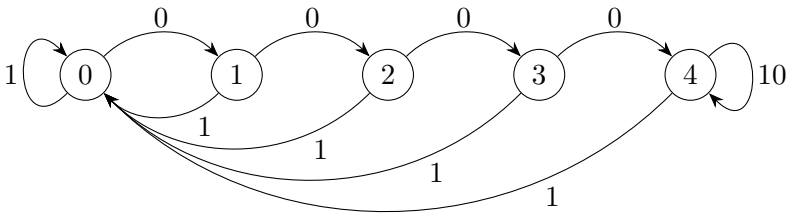

Figure 1.1: Five-state chain problem. Arrows from the top of the states denote action 0, and from the bottom denote action 1. The labels on each arrow denote the reward associated with that action. The system is initially at state 0.

The optimal, long term strategy for this system is to always perform action 0, which will cause the system to reach state 4 and obtain a reward of 10 on each step. A greedy algorithm will discover the reward from always performing action 1, which will cause the system to stay at

state 0 and obtain a reward of 1 on each step. The greedy algorithm is only optimal for four steps or less.

Q-Learning

One way to find a solution to this problem without neural networks is Q-Learning. Proposed by Watkins and Dayan, 1992, Q-Learning is an algorithm for finding optimal solutions to Markovian domain problems, such as the chain.

The algorithm starts with a table, referred to as the Q-table, which has a row for each system state, and a column for each action, initialised as zero. Once trained, each element of the Q-table will represent the maximum expected reward for each action in each state.

Let $Q(s, a)$ denote the Q-table value for state s and action a.

Initially, actions are taken at random, which allows the agent to explore the network; but as training progresses, the agent gradually shifts towards making decisions based on the Q-table values.

Each time an action a is taken from state s, the reward r and new state s' are observed, and the Q-table is updated using the equation

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)),$$

where α is the learning rate, and γ is the discount rate, which is needed to prevent the values from growing exponentially. Note that $\max_{a'} Q(s', a')$ is the maximum expected future reward of state s', according to the current Q-table values.

The algorithm was applied to the chain problem with learning rate $\alpha = 0.8$, and discount rate $\gamma = 0.7$, for 200 episodes, each consisting of 50 steps. During training, the probability of the agent performing a random action was given by $0.01^{t/200}$, where t is the episode number.

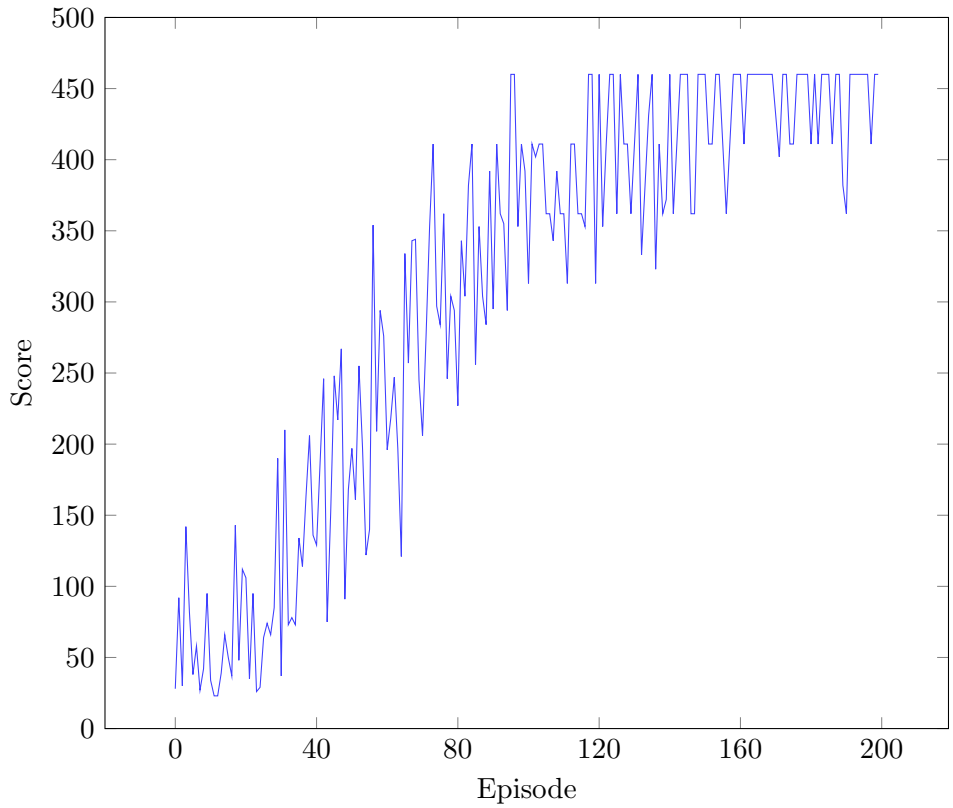

Figure 1.2: Graph of score against episode number for the Q-Learning chain problem.

Once trained, the agent was able to obtain the highest possible score of 490. The final values of the Q-table are provided below.

State	Action 0	Action 1
0	8.00	6.60
1	11.43	6.60
2	16.33	6.60
3	23.33	6.60
4	33.33	6.60

Figure 1.3: Q-table for the full trained Q-Learning chain problem.

Note that all of the values in the Action 0 column are larger than the corresponding values in the Action 1 column, which denotes the larger expected future reward.

Deep Q-Learning

The Q-Learning algorithm works well for problems with well defined states and actions, but cannot accommodate problems where the state in not discrete.

To solve this problem,

Chapter 2

Null

TODO: *Decide parent chapter for CNN, RNN, and LSTM sections*

Appendices

Appendix A

Derivation of Multi-layer Neural Network Equation

I is the number of inputs per sample, N is the number of samples, H is the number of hidden neurons.

$$
\begin{aligned}
h_i &= \tanh(s_i), \\
s_i &= b_i + \sum_{j=0}^{I} w_{i,j} x_j \\
y &= b + \sum_{i=0}^{H} w_i h_i.
\end{aligned}
$$

Vectorise:

$$
s_i = \begin{pmatrix} w_{i,1} & \cdots & w_{i,I} & b_i \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_I \\ 1 \end{pmatrix}, \qquad y = \begin{pmatrix} w_1 & \cdots & w_H & b \end{pmatrix} \cdot \begin{pmatrix} h_1 \\ \vdots \\ h_H \\ 1 \end{pmatrix}.
$$

$$
\mathbf{s} = \begin{pmatrix} s_1 \\ \vdots \\ s_H \end{pmatrix} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,I} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{H,1} & \cdots & w_{H,I} & b_H \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_I \\ 1 \end{pmatrix} = W \cdot \mathbf{x},
$$

$$
\mathbf{h} = \begin{pmatrix} \tanh(\mathbf{s}) \\ 1 \end{pmatrix}, \qquad y = \mathbf{w} \cdot \mathbf{h}.
$$

Batch:

$$
S = \begin{pmatrix} \mathbf{s}^{(1)} & \cdots & \mathbf{s}^{(N)} \end{pmatrix} = W \cdot \begin{pmatrix} \mathbf{x}^{(1)} & \cdots & \mathbf{x}^{(N)} \end{pmatrix},
$$

$$
\Phi = \tanh(S), \qquad \Psi = \begin{pmatrix} \Phi \\ \mathbf{1} \end{pmatrix},
$$

$$\mathbf{y} = \begin{pmatrix} y^{(1)} & \cdots & y^{(N)} \end{pmatrix} = \mathbf{w} \cdot \Psi.$$

Backpropagation:

$$\begin{aligned} d &= y - y_t, \\ e &= \frac{1}{2} d^2, \end{aligned}$$

$$\begin{aligned} \frac{\partial s_i}{\partial w_{i,j}} &= x_j, & \frac{\partial h_i}{\partial s_i} &= 1 - h_i^2, \\ \frac{\partial y}{\partial w_i} &= h_i, & \frac{\partial y}{\partial h_i} &= w_i, \\ \frac{\partial e}{\partial y} &= d. \end{aligned}$$

$$\begin{aligned} \frac{\partial e}{\partial w_i} &= \frac{\partial e}{\partial y}\frac{\partial y}{\partial w_i} = dh_i, \\ \frac{\partial e}{\partial w_{i,j}} &= \frac{\partial e}{\partial y}\frac{\partial y}{\partial h_i}\frac{\partial h_i}{\partial s_i}\frac{\partial s_i}{\partial w_{i,j}} = dw_i(1 - h_i^2)x_j. \end{aligned}$$

Batch:

$$\begin{aligned} \mathbf{d} &= \mathbf{y} - \mathbf{y}_t, \\ \frac{\partial e}{\partial w_i} &= \sum_k^N d^{(k)} h_i^{(k)} \\ &= \begin{pmatrix} d^{(1)} & \cdots & d^{(N)} \end{pmatrix} \cdot \begin{pmatrix} h_i^{(1)} & \cdots & h_i^{(N)} \end{pmatrix} \\ &= \mathbf{d} \cdot \mathbf{h}_i. \\ \frac{\partial e}{\partial w_{i,j}} &= \sum_k^N d^{(k)} w_i \left(1 - \left(h_i^{(k)}\right)^2\right) x_j^{(k)} \\ &= (\mathbf{1} - \mathbf{h}_i \odot \mathbf{h}_i) \odot (w_i \mathbf{d}) \cdot \mathbf{x}_j. \end{aligned}$$

Vectorise:

$$\begin{aligned} \frac{\partial e}{\partial \mathbf{w}} &= \mathbf{d} \cdot \begin{pmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{H+1} \end{pmatrix} = \mathbf{d} \cdot \Psi^T. \\ \frac{\partial e}{\partial W} &= \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_{I+1} \end{pmatrix} = \left(1 - \begin{pmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_H \end{pmatrix} \odot \begin{pmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_H \end{pmatrix}\right) \odot \begin{pmatrix} w_1 \mathbf{d} \\ \vdots \\ w_H \mathbf{d} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_1^T & \cdots & \mathbf{x}_{I+1}^T \end{pmatrix} \\ &= (1 - \Phi \odot \Phi) \odot (\hat{\mathbf{w}} \cdot \mathbf{d}) \cdot X^T, \end{aligned}$$

Bibliography

Strens, Malcolm J. A. (2000). “A Bayesian Framework for Reinforcement Learning”. In: *ICML*.
Watkins, Christopher JCH and Peter Dayan (1992). “Q-learning”. In: *Machine learning* 8.3-4, pp. 279–292.