QDN Network For Maze Solving

Axel Ind (*4563539*), Fernando Cristiani *(4527671)*

January 23, 2018

Abstract

This document details work to create and evaluate a convolutional neural network that solves mazes with a partially observable environment using reinforcement learning. Training is conducted via bounded exploration of the environment. The network architecture consisted of 2 convolutional layers, each followed by a pooling layer and a single fully connected softmax output layer. Training loss of ≈ 0.003 is achieved for history ≥ 4. Testing accuracy of 100% is achieved for steps> 8000.

1 Dataset

1.1 Data Format

Each input consists of 3-dimensional numeric input of depth d. The $25 \times 25 \times d$ matrix provides a instantaneous grayscale representation of the current state of the agent in the maze, and the same information for each of its last $d-1$ states. Individual values in the matrix range from 0 to 255 and are used to represent cell occupation.

1.2 Data Generation

Data is generated and integrated into the neural network after each action taken by the agent.

2 Network Architecture

2.1 Inputs

Stochastic Gradient Descent: $25 \times 25 \times d \times n$ (where d is the history length, n is the batch size).

2.2 Outputs

$Y = (y_0, y_1, y_2, y_3, y_4)$
Each class corresponds to a classification of: $(0, 4) \in \mathbb{Z}$.

More precisely the actions are mapped as follows:

Action	**Agent Motion**
0	Do nothing
1	Up
2	Down
3	Left
4	Right

3 Training

3.1 Propagation

All training results are achieved using the AdamOptimizer to update weights.

3.2 Testing Hyperparameters

- **Network architecture**: Input$-\{$Conv-Relu-Pool$\}^2-$FullyConnectedLayer-Softmax
- **NumFilters**: $32^{|1|}, 32^{|2|}$
- **KernelSize**: 5×5
- **BatchSize**: 32
- **Stride**: 2
- **DiscountFactor**: 0.8

Testing Architecture

CPU	i7-6700HQ
GPU	GTX970 (6GB)
RAM	16GB
Hard Disk	SSD

4 Questions

4.1 Update rule for Q-Function

$$Q(i,u) = (1-\alpha)Q(i,u) + \alpha\left(r(i,u) + (\gamma = 0.5) max_{u' \in U_j} Q(j,u') - Q(j,u')\right) \quad (1)$$

where i is the intitial state, j is the successor state, α is the learning rate, u is the action taken from i, U_j is the set of applicable operators from state j and γ is the learning rate.

4.2 Transitions to Absorbing Goal State

- Equation 1 accurately describes the update procedure for a transition to an absorbing goal state. However, the reward for entering the goal state could be set *a priori* to allow the Q-function to update accurately.

4.3 Transitions from Absorbing Goal State

- The Q-value of transitions from absorbing goal states should not change, therefore a check should be added to Equation 1. This would require a lookup table of goal states.
- Option 1: Prevent any actions from being valid in the goal state. (As is Figure 1).
- Option 2: Set the cost of any action from a goal state (except "Do Nothing") to ∞.
- Option 3: End the simulation when the goal is reached[1].

4.4 Q-Function Example

Changed Q-Values

$Q((0,0), u = d), [Q((1,0), u = r), [Q((1,1), u = u), [Q((1,1), u = r), [Q((1,2), u = u)$

Improved Q-Values

1. $$Q((0,0), u = d) = -1 + 0.5(0) = -1$$
2. $$Q((1,0), u = r) = -1 + 0.5(0) = -1$$
3. $$Q((1,1), u = u) = -1 + 0.5(0) = -1$$

[1]This is the method applied in the implementation this paper describes

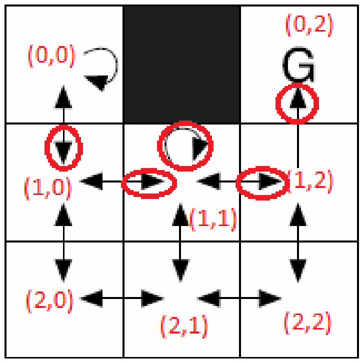

Figure 1: Figure showing the changed Q-functions in the toy example.

4. $$Q((1,1), u = r) = -1 + 0.5(0) = -1$$

5. $$Q((1,2), u = u) = -1 + 0.5(0) = -1$$

6. All other Q values are 0.

5 Results

5.1 Loss

The AdamOptimizer was used to minimize the loss function described in *q_loss.Q_loss()*. Loss values show a weakly-logarithmic decrease (Figure 2).

Notes on Loss

Normalization: Normalising Q-Values prevented gradient divergence.

5.2 Goal-Finding Accuracy

- **Learning Rate**: 0.0005
- **Dropout**: 0.4
- **Alpha** (exploration): 0.3

The agent is able to accurately reach the goal 100% of the time after 2000th step for the small training map (Figure 3).

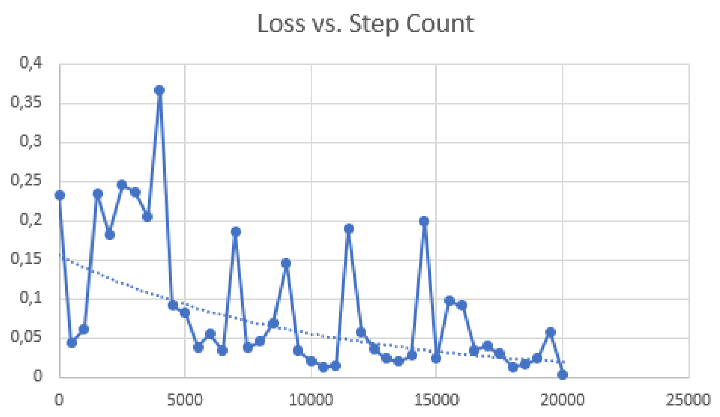

Figure 2: Graph showing the loss values (500-step intervals) achieved by the QDN.

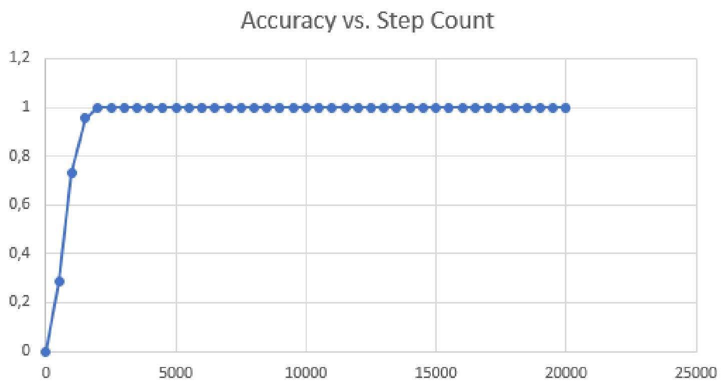

Figure 3: Graph showing the accuracy values (500-step intervals) achieved by the QDN.

5.3 Changed Target

A trained agent adapts to the new target very quickly (Figure 4) and after 1500 training steps is able to reach the goal with 100% accuracy.

5.4 Changed Map

For the purposes of this experiment, the trained agent was made to train on an unseen map environment. It quickly adapted to the new map and, after 1500 training steps, was able to reach the new goal with 100% accuracy (Figure 5).

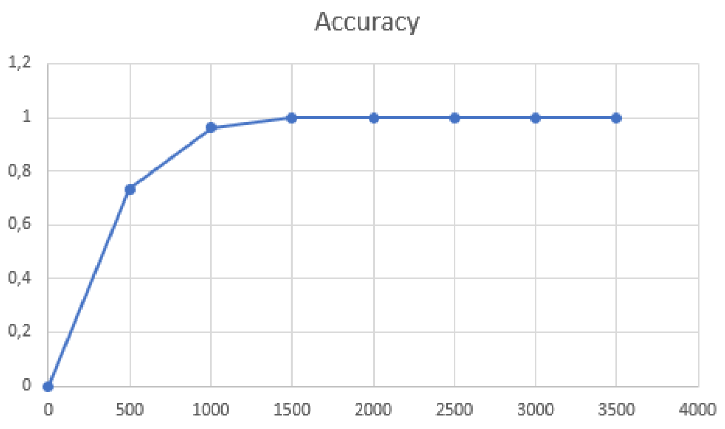

Figure 4: Graph showing the accuracy values (500-step intervals) achieved by the trained QDN with a new target.

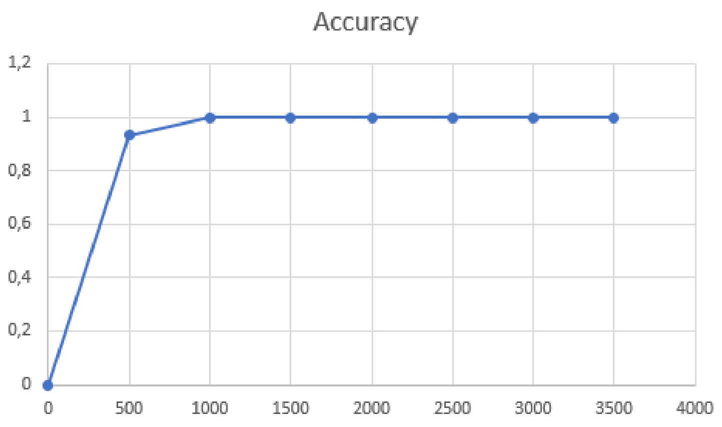

Figure 5: Graph showing the accuracy values (500-step intervals) achieved by the trained QDN with an unknown map.

6 Conclusion

This paper has illustrated the suitability of convolutional neural networks for the problem of emulating dynamic Q-learning search results in maze solving. It has shown that, given a static or dynamic state space and a semi-observable environment, this problem is tractable with a QDN.