Case Study 5: Deep Q-learning for Tic-Tac-Go challenge

Reporter: Zong Fan

Email: zongfan2@illinois.edu

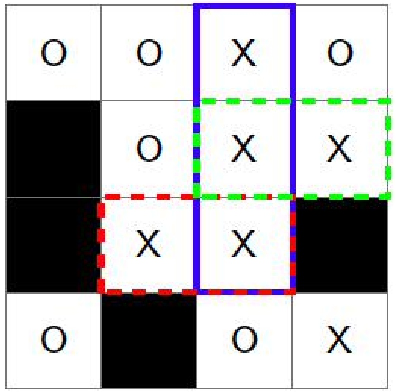
1. Objective

- Train an AI agent with deep Q-learning to play Tic-Tac-Go.
- What is Tic-Tac-Go?
  - Similar to Tic-Tac-Toe game, but is expanded 4×4 grid.
  - The winner is the player whose final score is higher than the other's.
  - The final score is computed as:

$$score = \sum_i (m_i - 1)^2 + \sum_j (n_j - 1)^2$$

$$winner \doteq argmax_i(score_{player\ i})$$

m_i, n_j is length of chain in
the i th row and j th column.



X score:

$$(2-1)^2+(2-1)^2+(3-1)^2 = 6$$

O score:

$$(2-1)^2 + (2-1)^2 = 2$$

2. Method

- In this study, we employed deep Q-learning to train a deep neural network as the agent.
- As the lecture said, the pipeline to train a deep Q-learning model includes:
    1. Construct a network Q(s,a) with random weights W.
    2. Generate a trajectory s_0, a_0, r_0, ..., s_T,r_T.
    3. Compute Q-factor target for each (s, a) pair as:
       $Q\prime = r_i + \gamma \times max_a\{Q(s_{i+1}, a)\}$ r: reward of the state, γ: discounting factor
    4. Compute loss between the network and the target using MSE loss.
    5. Backpropagate the gradient to update network parameters W.
    6. Go to #2 and repeat.

2.1 Player 1: Deep Network

- The input of the network is the borad status, including:
  - blocked positions S_b
  - empty positions S_e
  - palyer 1 occupied positions S_{p1}
  - player 2 occupire positions S_{p2}
- The output of the network is the probability of action to put on each board locations.
- We define the dimension of input and output as (N, 4, 4, 4) and (N, 16).

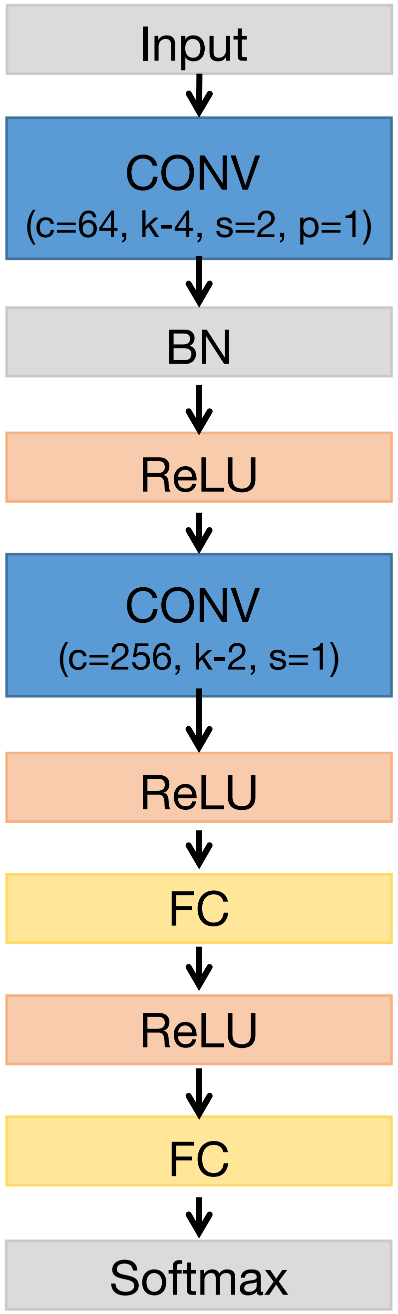
Output position code:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

2.2 Network architecure

- We designed a convolutional neural network with 2 convolutional layers and 2 fully-connected layers.
- Rectified linearUnits (ReLU) is followed after each layer as the non-linear acitvation function.
- The final softmax layer transform the output values into probabilities.

$$P(y=j|x) = \frac{e^x}{\sum_{j=0}^{k} e^x}$$



2.3 Player 2: random player

- For comparison, we designed a random player who randomly choose a empty position as the action when it was his turn.

- After the intial deep network player was trained, it could be used as the baseline AI agent for further optimization.

2.3 Training

- We trained the network by setting discounting factor as 0.95.
- Reward for win, lose, tie is 1, -1, 0 respectively.
- Adam optimizer is employed to update model weights with initial learning rate as 0.0001.
- 1000,000 rounds of games are played to train the network.
- To exploit game states, we play random rollout in first 1/4 iterations with proability of 0.6; it drops to 0.1 for the next 1/4 iterations; and the rest iterations uses 0.05.

3. Results

- During the training process, the maximum average reward of Q-player reaches 0.65. It indicates more training iterations should be done to achieve higher performance.

- Several game sets:

```
---------
|o|x|x|o|
---------
|o|o|#|x|
---------
|x|#|x|o|
---------
|o|x|#|#|
---------
```

```
---------
|x|o|o|x|
---------
|o|x|o|#|
---------
|x|o|#|#|
---------
|#|x|o|x|
---------
```

```
---------
|x|x|#|o|
---------
|o|o|o|x|
---------
|x|x|o|#|
---------
|o|x|#|#|
---------
```

Q-player: player 2 (o)

```
---------
|x|x|o|o|
---------
|x|x|#|x|
---------
|#|x|o|o|
---------
|#|o|o|#|
---------
```

```
---------
|o|#|o|o|
---------
|x|x|x|x|
---------
|o|o|x|#|
---------
|x|#|#|o|
---------
```

```
---------
|o|x|x|o|
---------
|#|x|x|x|
---------
|o|o|x|#|
---------
|#|o|#|o|
---------
```

Q-player: player 1(x)

is blocked positions

4. Code

Link to access the code:
https://gist.github.com/CasiaFan/6a83b4a159fc5a2f3ed75e9a76f38aa8