Solving Bomberman

with Q-table and Neural Networks

Behrooz Montazeran
Jannis Heising
Tomáš Sláma

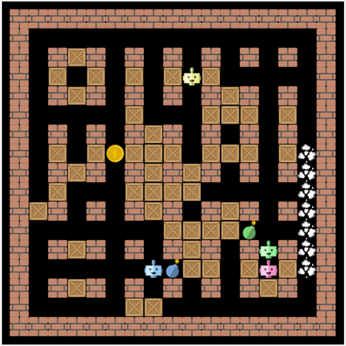
20. 12. 2023

Introduction

Problem Statement

Problem Statement

- **6 actions** – 4 movement directions + wait + place bomb
- destructible crates, indestructible walls
- **+1** for coin, **+5** for killing opponent



Methods Used

Methods Used

Q-learning – optimizing action-value Q-function:

- **Q-table:** store each input/output in a table
- **Neural Network:** learn function via a neural network

Methods Used

Q-learning – optimizing action-value Q-function:

- **Q-table:** store each input/output in a table
- **Neural Network:** learn function via a neural network

TD update – Use best possible predicted value in next step:

$$Y_t \leftarrow R_t + \gamma \cdot \max_a Q(s_{t+1}, a),$$
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \mu \cdot (Y_t - Q(s_t, a_t)) \, .$$

Q-learning – optimizing action-value Q-function:

- **Q-table:** store each input/output in a table
- **Neural Network:** learn function via a neural network

TD update – Use best possible predicted value in next step:

$$Y_t \leftarrow R_t + \gamma \cdot \max_a Q(s_{t+1}, a),$$
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \mu \cdot (Y_t - Q(s_t, a_t))\,.$$

Stability can be improved with policy/target model:

- **target**: trained on the results of actions
- **policy**: updated slowly / replaced periodically

Infrastructure

Infrastructure

Evaluation with `elo.py`:

- comparing relative strengths of agents using the elo system
- agents start with 1 000 elo, updated after win/loss/draw

Evaluation with `elo.py`:

- comparing relative strengths of agents using the elo system
- agents start with 1 000 elo, updated after win/loss/draw

Training with `train.py`:

- subcommands for training traits (coin picking, agent hunting)
- periodically calculates strength with `elo.py`
- useful training flags: `--infinite <n>`, `--continue`

Training Command Example

Training Command Example

```
TASKS = {
    ...
    "complete": [
        # first learn to pick up coins
        (["--scenario", "coin-heaven", "--n-rounds", "100"], False),
        # then learn to hunt/evade the rule-based agent
        (["rule_based_agent", "--scenario", "empty", "--n-rounds", "1000"], False),
        # then learn to work with crates
        (["--scenario", "classic", "--n-rounds", "1000"], False),
        # then learn to work with crates + hard agent
        (["rule_based_agent", "--scenario", "classic", "--n-rounds", "200"], True),
        # then learn to work with crates + really hard agent
        (["binary_agent_v3", "--scenario", "classic", "--n-rounds", "200"], True),
        # finally play against itself (None gets substituted)
        ([None, "--scenario", "classic", "--n-rounds", "200"], True),
    ],
    ...
}
```

Neural Networks

Overview

Network structure: fully connected layers with ReLU

- uses TD for Q-value updating

Overview

Network structure: fully connected layers with ReLU

- uses TD for Q-value updating

Main idea – carefully crafted binary state vector:

$1 \ldots 5$:	direction to the nearest **coin**
$6 \ldots 10$:	direction to the nearest **crate**
$11 \ldots 15$:	direction to where a bomb will **kill opponent**
$16 \ldots 20$:	direction to **safety** (if in danger of dying)
21:	**can place a bomb** and there is a way to escape it

Overview

Network structure: fully connected layers with ReLU

- uses TD for Q-value updating

Main idea – carefully crafted binary state vector:

1 … 5: direction to the nearest **coin**

6 … 10: direction to the nearest **crate**

11 … 15: direction to where a bomb will **kill opponent**

16 … 20: direction to **safety** (if in danger of dying)

21: **can place a bomb** and there is a way to escape it

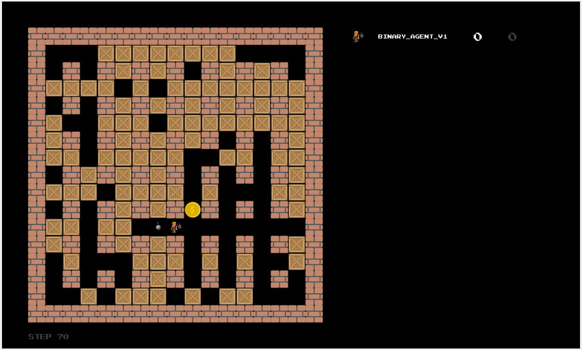
Calculated by **searching state space** (other players stand still)

$\Rightarrow$ finds path even in ongoing explosions

$\Rightarrow$ shortest on-board path $\neq$ shortest path stepwise

Q-vector example (1)

Q-VECTOR EXAMPLE (1)



`0 1 0 0 0 | 0 1 0 0 0 | 0 0 0 0 0 | 0 1 0 0 0 | 0`

Nearest coin to the right

Nearest crate to the right
(by state space search)

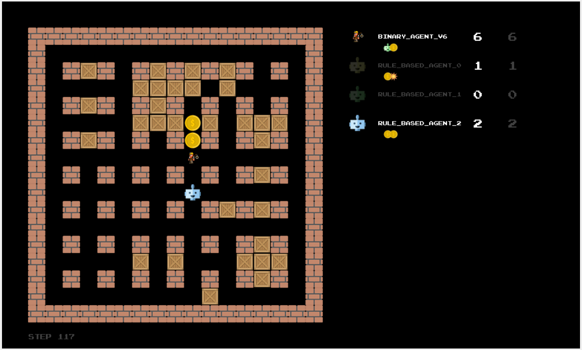
No players reachable

Safety to the right

Can't place a bomb

Q-vector example (2)

Q-vector example (2)



1 0 0 0 0 | 1 0 0 0 0 | 0 0 0 0 1 | 0 0 0 0 0 | 1

Nearest coin up

Nearest crate up

Place bomb to endanger player

Safe

Can place a bomb

Training Parameters

Training Parameters

Hyperparameters

```
BATCH_SIZE = 256
MEMORY_SIZE = 1000

# chances of picking random action
# happens per-round, not per-epoch!
EPS_START = 0.99
EPS_END = 0.05
EPS_DECAY = 10

# (soft) update rate of target network
TAU = 1e-3

# LR of the optimizer
LR = 1e-4

# discount for future states
GAMMA = 0.99

OPTIMIZER = optim.Adam
LAYER_SIZES = [1024, 1024]
```

Event Rewards

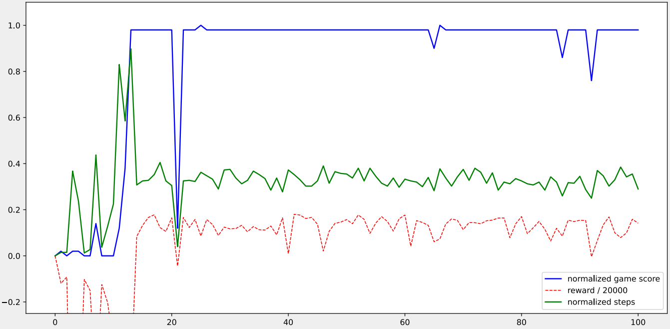
```
# hunt coins
MOVED_TOWARD_COIN: 50
DID_NOT_MOVE_TOWARD_COIN: -100

# hunt people
MOVED_TOWARD_PLAYER: 25
DID_NOT_MOVE_TOWARD_PLAYER: -10

# blow up crates
MOVED_TOWARD_CRATE: 1

# basic stuff
e.INVALID_ACTION: -100
DID_NOT_MOVE_TOWARD_SAFETY: -500

# be active!
USELESS_WAIT: -100

# meaningful bombs
PLACED_USEFUL_BOMB: 50
PLACED_SUPER_USEFUL_BOMB: 150
DID_NOT_PLACE_USEFUL_BOMB: -500
```

Training Loop

Training Loop

1. `train.py` with the `"complete"` task and `--infinite 2`
2. pick the best-performing model based on calculated elo
3. decrease `EPS_START`, `EPS_END`, `TAU` and `LR` hyperparameters
4. go to step 1 (with `--continue` to not overwrite the model)

1. `train.py` with the `"complete"` task and `--infinite 2`
2. pick the best-performing model based on calculated elo
3. decrease `EPS_START`, `EPS_END`, `TAU` and `LR` hyperparameters
4. go to step 1 (with `--continue` to not overwrite the model)

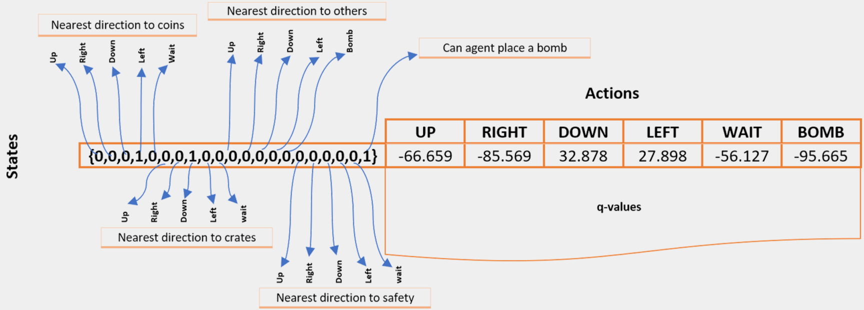


Q-table

Storing Q-Table

Storing Q-Table

- Theoretically $207 \times 2^{21} \times 6 = 2.60466278e9$ possible states
- 207 free tiles for the agent to go
- 2^{21} possible actions-state pairs
- 6 actions



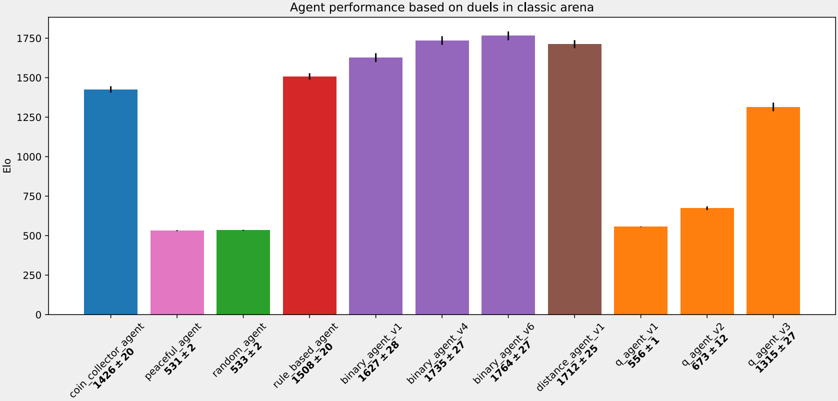
In reality, agent encounters much fewer states

Scenario	Opponent(s)	Number of Rounds	Maximum new encountered states	Encountered states	Run time	Total run time
Coin-heaven	-	5,000	~100	100	~00:03:40	00:03:40
Sparse-crate	-	30,000	~560	660	~00:20:30	00:24:10
Sparse-crate	Peaceful-agent	40,000	~2290	2950	~00:47:35	01:11:45
classic	Peaceful-agent	40,000	0	2950	~00:27:28	01:39:13
classic	Peaceful-agent & rule-based-agent	40,000	~2530	5480	~01:15:17	02:54:30
classic	rule-based-agent & rule-based-agent & rule-based-agent	100,000	~5315	10795	~05:44:10	08:38:40
Doubled	Above in order	255,000	~16	10811	~04:20:40	12:59:20

Conclusion

Results

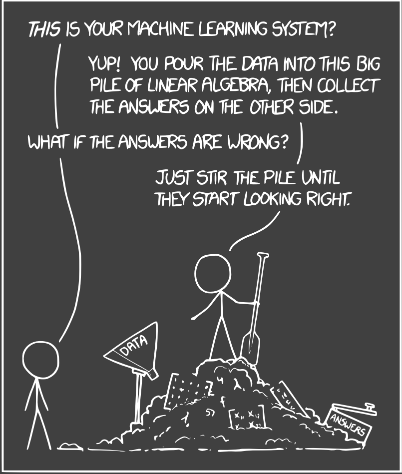
Results



Agent performance based on duels in classic arena
Elo
1750
1500
1250
1000
750
500
250
0
coin_collector_agent
1426 ± 20
peaceful_agent
531 ± 2
random_agent
533 ± 2
rule_based_agent
1508 ± 20
binary_agent_v1
1627 ± 28
binary_agent_v4
1735 ± 27
binary_agent_v6
1764 ± 27
distance_agent_v1
1712 ± 25
q_agent_v1
556 ± 1
q_agent_v2
673 ± 12
q_agent_v3
1315 ± 27

Thanks for your attention!

Special thanks to Ullrich Köthe and tutors



References