Reinforcement Learning in Games

Ege Can Özer

Motivation

Motivation

- Less complicated, controlled environment

Motivation

- Less complicated, controlled environment
- Clear-cut rules

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements
  - Win %, kill rate, score, time spent

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements
  - Win %, kill rate, score, time spent
- Easy to realize

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements
  - Win %, kill rate, score, time spent
- Easy to realize
  - At least no need to wait for a fund to buy a robot arm ☺

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements
  - Win %, kill rate, score, time spent
- Easy to realize
  - At least no need to wait for a fund to buy a robot arm ☺
- Provides a common tool

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements
  - Win %, kill rate, score, time spent
- Easy to realize
  - At least no need to wait for a fund to buy a robot arm ☺
- Provides a common tool
  - Training and testing (Benchmark)

Motivation

- Less complicated, controlled environment
- Clear-cut rules
- Concrete evaluation measurements
  - Win %, kill rate, score, time spent
- Easy to realize
  - At least no need to wait for a fund to buy a robot arm ☺
- Provides a common tool
  - Training and testing (Benchmark)

- More focus on the algorithm

Challenges

Challenges

- Implementation POV

Challenges

- Implementation POV
  - Hard to find SDK, let’s create the game from scratch

Challenges

- Implementation POV
  - Hard to find SDK, let’s create the game from scratch
- Abstracting the environment

Challenges

- Implementation POV
  - Hard to find SDK, let's create the game from scratch
- Abstracting the environment
  - Low-level, high-level, context-free

Challenges

- Implementation POV
  - Hard to find SDK, let’s create the game from scratch
- Abstracting the environment
  - Low-level, high-level, context-free
- Performance measures: No hand labeled ground-truth set

Challenges

- Implementation POV
  - Hard to find SDK, let's create the game from scratch
- Abstracting the environment
  - Low-level, high-level, context-free
- Performance measures: No hand labeled ground-truth set
  - Benchmarking tools, Human players, Evaluate individual decisions

Challenges

- Implementation POV
  - Hard to find SDK, let’s create the game from scratch
- Abstracting the environment
  - Low-level, high-level, context-free
- Performance measures: No hand labeled ground-truth set
  - Benchmarking tools, Human players, Evaluate individual decisions
- Further challenges

Challenges

- Implementation POV
  - Hard to find SDK, let's create the game from scratch
- Abstracting the environment
  - Low-level, high-level, context-free
- Performance measures: No hand labeled ground-truth set
  - Benchmarking tools, Human players, Evaluate individual decisions
- Further challenges
  - Exploitation vs exploration

Challenges

- Implementation POV
  - Hard to find SDK, let's create the game from scratch
- Abstracting the environment
  - Low-level, high-level, context-free
- Performance measures: No hand labeled ground-truth set
  - Benchmarking tools, Human players, Evaluate individual decisions
- Further challenges
  - Exploitation vs exploration
  - Many problems DL perspective

Outline

- Reinforcement Learning to Play Mario [LYY12]
- High-level Reinforcement Learning in Strategy Games [AmS10]
- Temporal Difference Learning and TD-Gammon [Tes95]
- Playing Atari with Reinforcement Learning [MKS13]

} 4 articles

Outline

- Reinforcement Learning to Play Mario [LYY12]
- High-level Reinforcement Learning in Strategy Games [AmS10]
- Temporal Difference Learning and TD-Gammon [Tes95]
- Playing Atari with Reinforcement Learning [MKS13]

Outline

- Reinforcement Learning to Play Mario [LYY12]
- High-level Reinforcement Learning in Strategy Games [AmS10]
- Temporal Difference Learning and TD-Gammon [Tes95]
- Playing Atari with Reinforcement Learning [MKS13]

Outline

- Reinforcement Learning to Play Mario [LYY12]
- High-level Reinforcement Learning in Strategy Games [AmS10]

- Temporal Difference Learning and TD-Gammon [Tes95]

- Playing Atari with Reinforcement Learning [MKS13]

Outline

- Reinforcement Learning to Play Mario [LYY12]
- High-level Reinforcement Learning in Strategy Games [AmS10]

- Temporal Difference Learning and TD-Gammon [Tes95]

- Playing Atari with Reinforcement Learning [MKS13]

Reinforcement Learning to Play Mario [LYY12]

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free
  - Algorithm oblivious to transitional probabilities

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free
  - Algorithm oblivious to transitional probabilities
  - Performs well under specific constraints

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free
  - Algorithm oblivious to transitional probabilities
  - Performs well under specific constraints
- Mario AI Framework

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free
  - Algorithm oblivious to transitional probabilities
  - Performs well under specific constraints
- Mario AI Framework
  - Used to both train and test the system

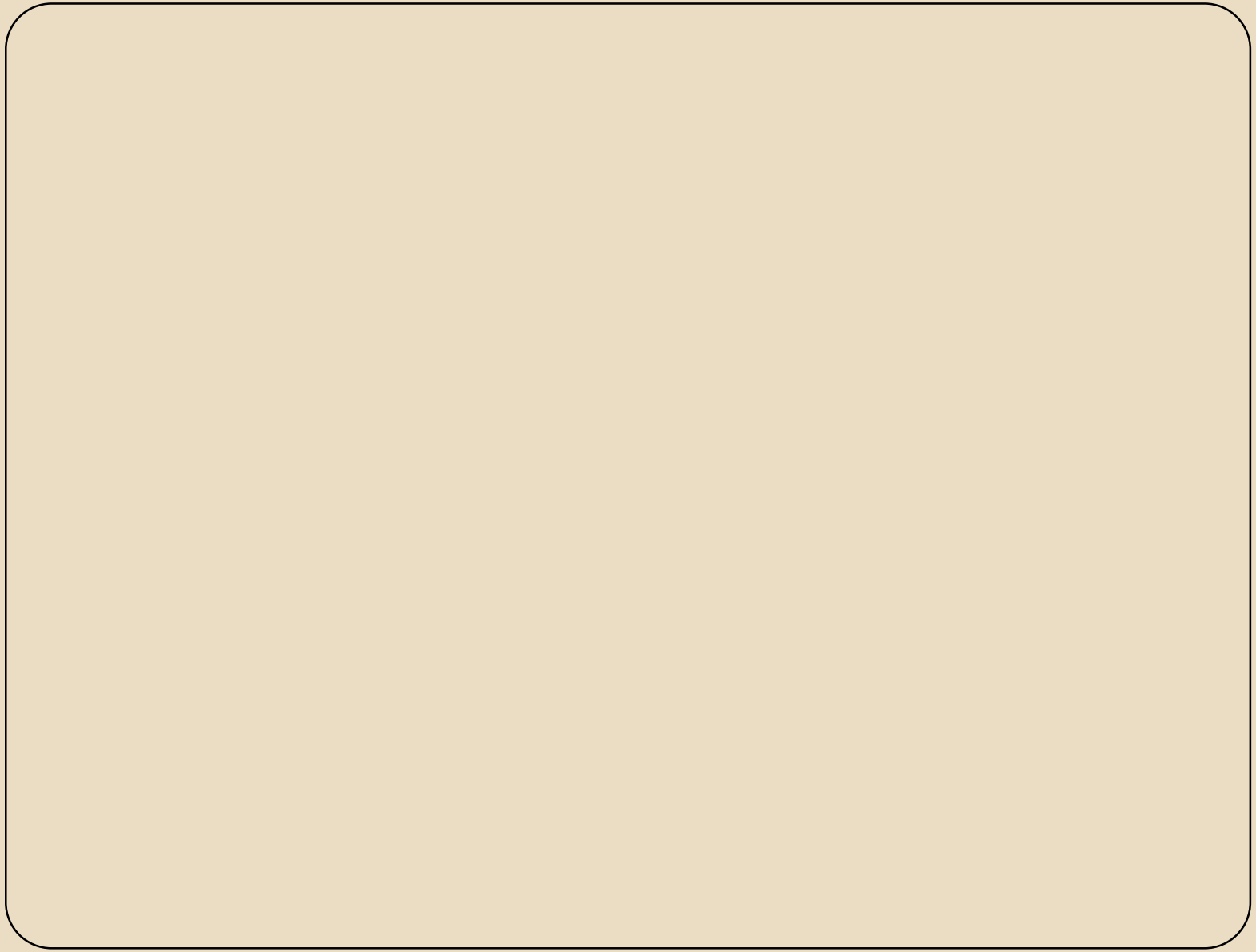
Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free
  - Algorithm oblivious to transitional probabilities
  - Performs well under specific constraints
- Mario AI Framework
  - Used to both train and test the system
- Fast convergence and decent results

Reinforcement Learning to Play Mario [LYY12]

- Aim of this study
  - Design an agent to play and beat the game Mario
- Q-learning is model-free
  - Algorithm oblivious to transitional probabilities
  - Performs well under specific constraints
- Mario AI Framework
  - Used to both train and test the system
- Fast convergence and decent results
  - 5000 iterations, 90% win probability

$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

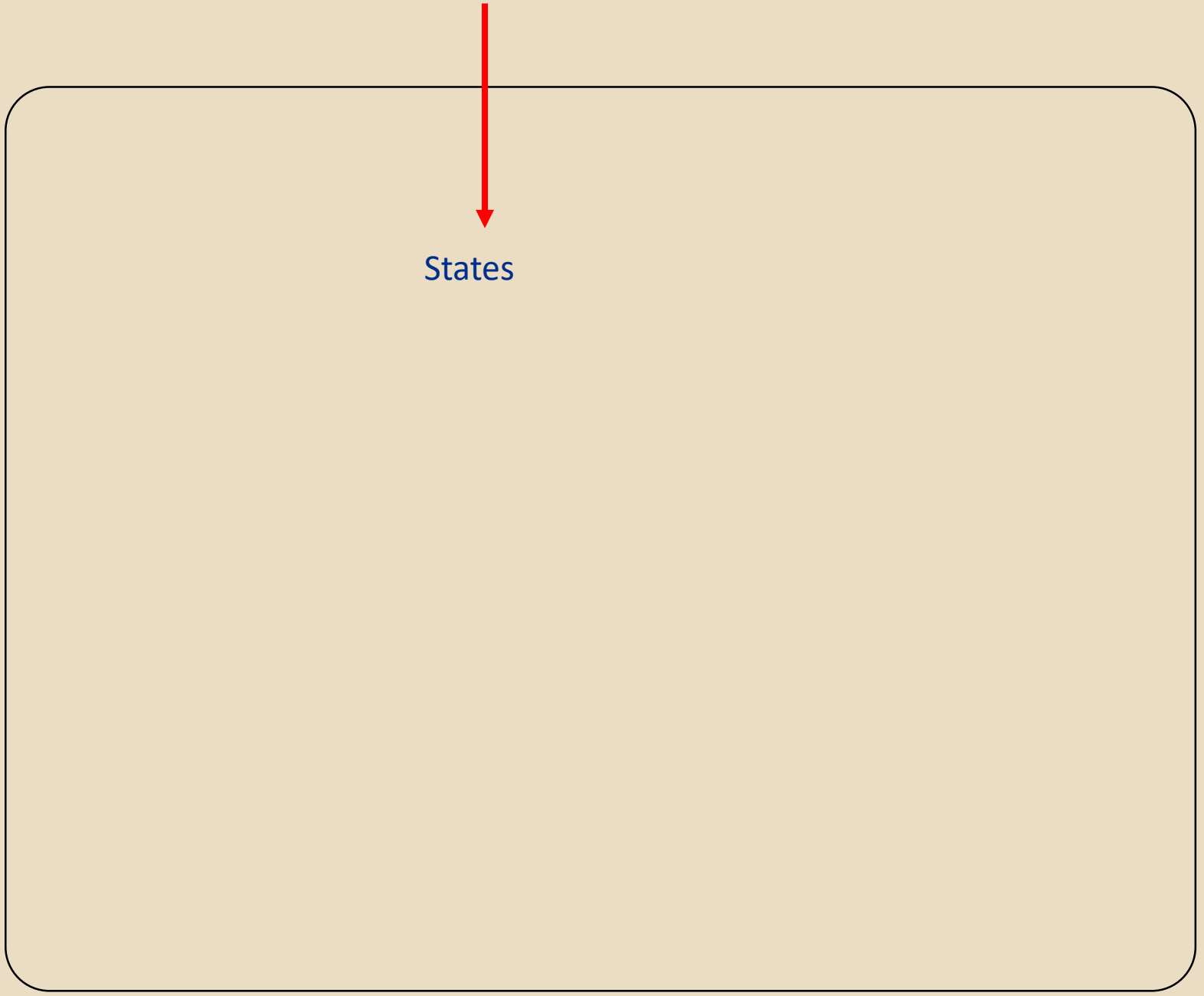


$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

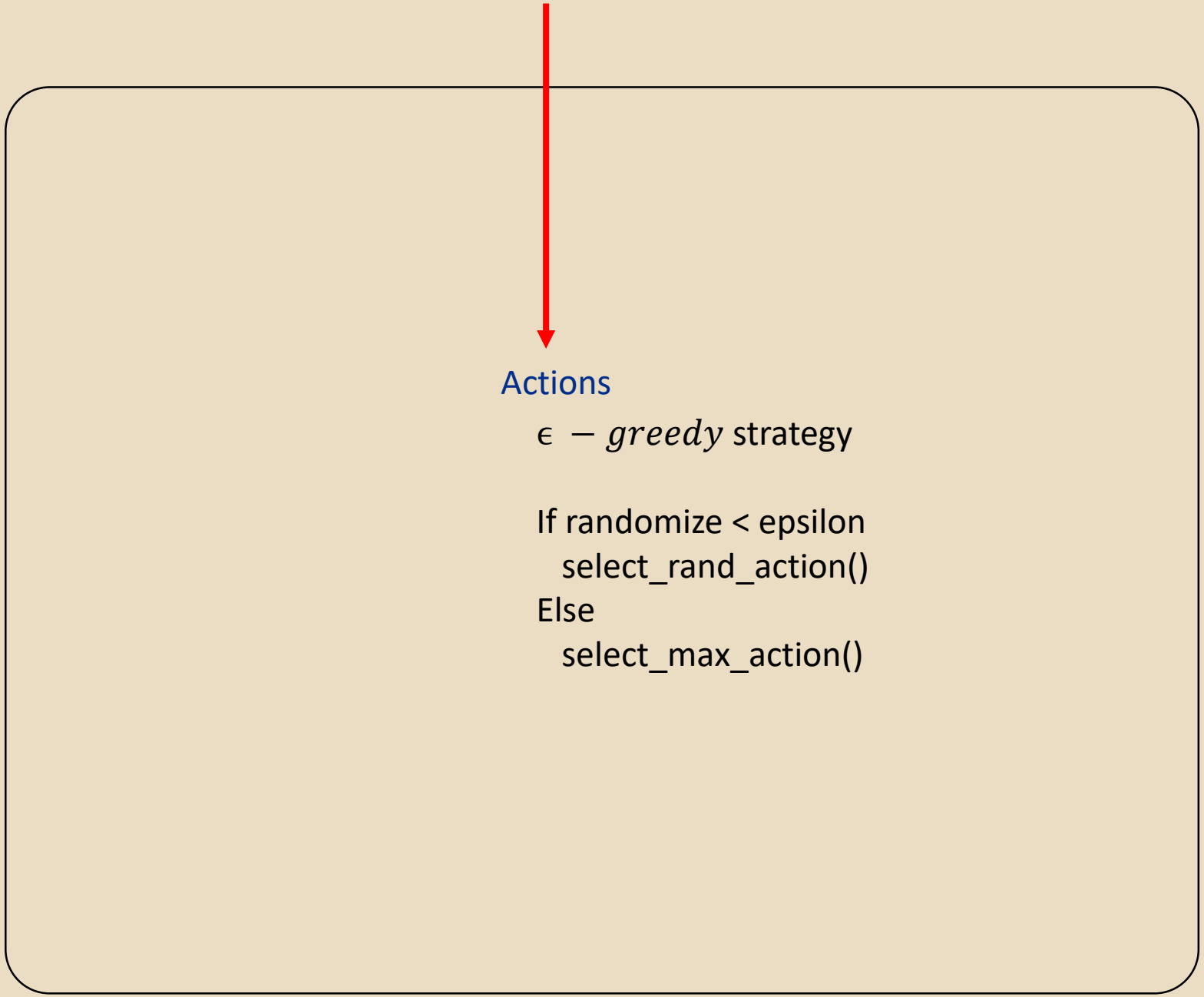
Q-table

Initialized with a
uniform distribution

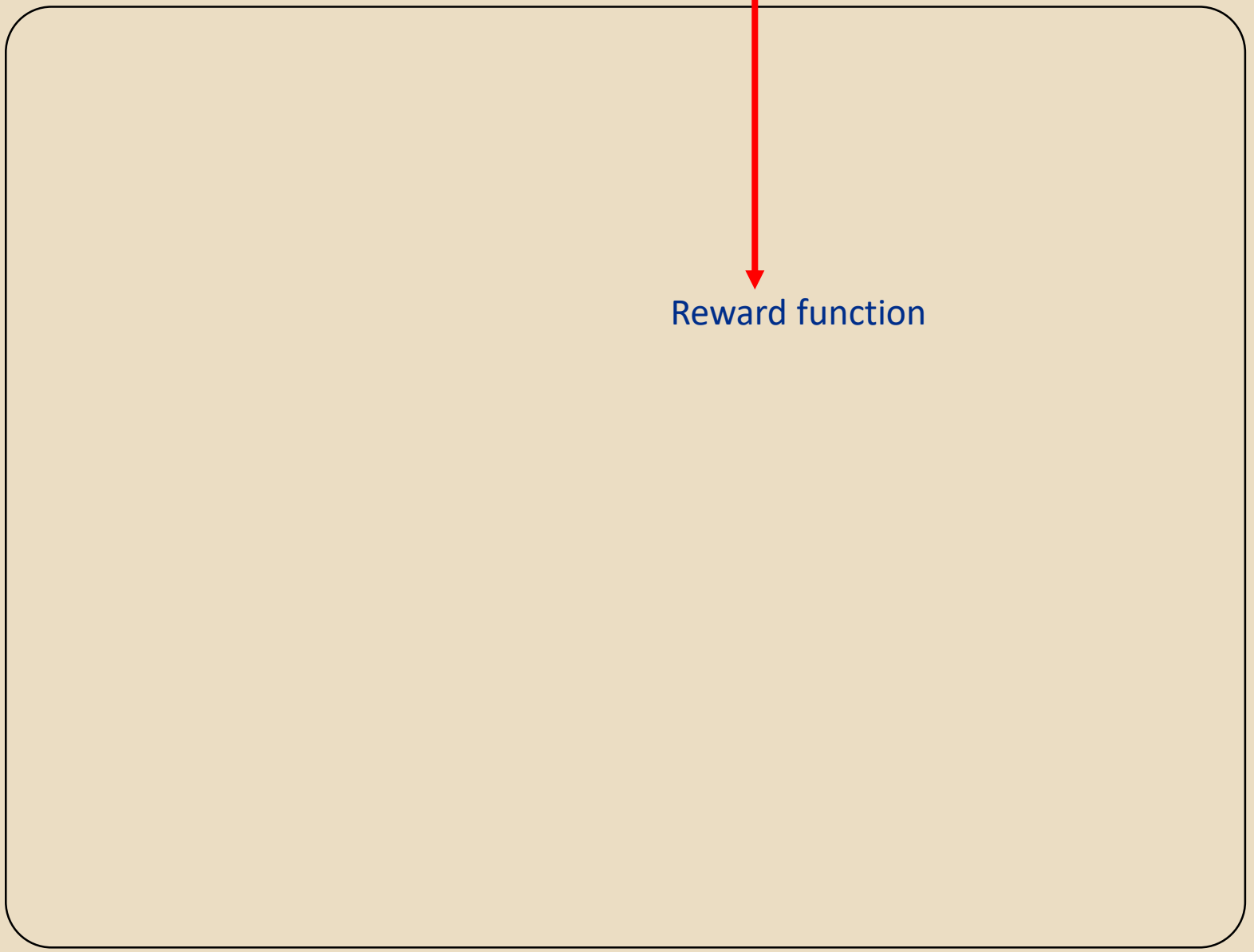
$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

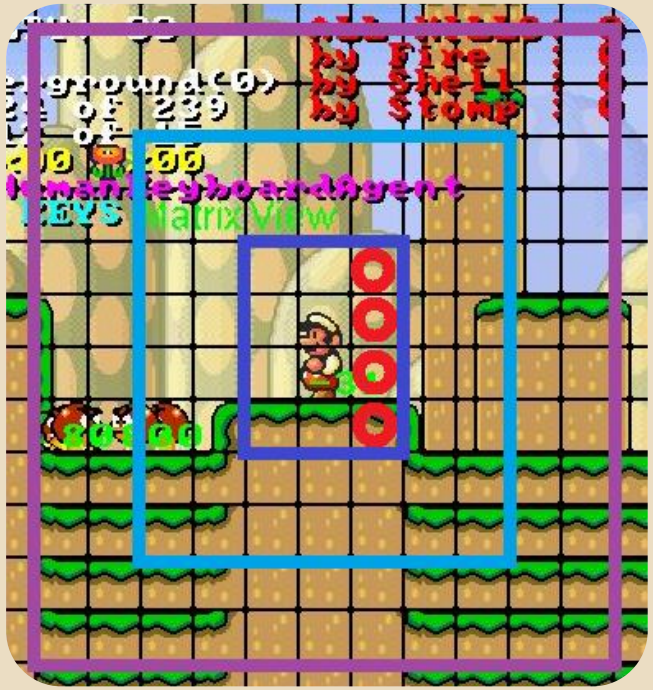
Learning rate

How fast learning converges

$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

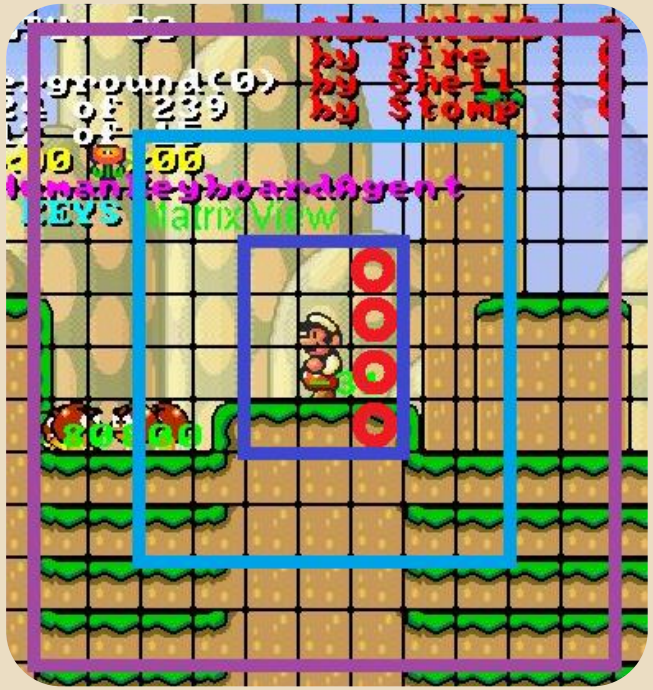
Discount factor

How much future state is taken into account



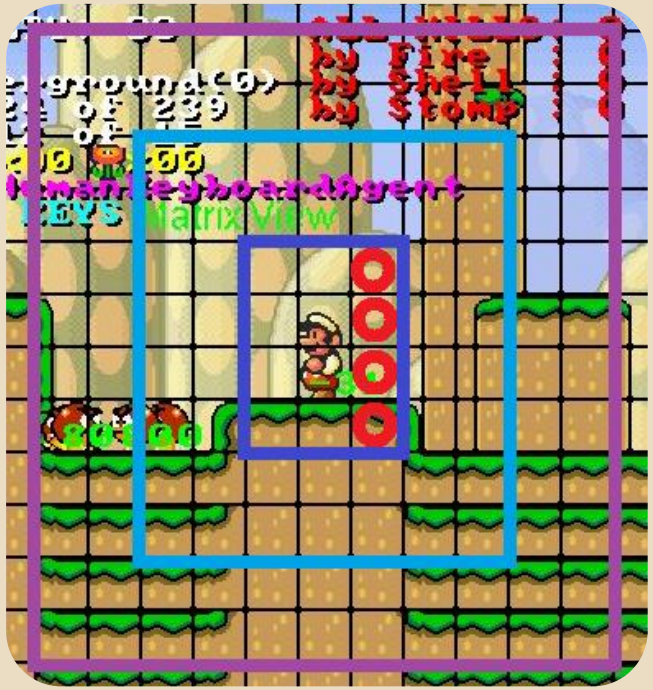
$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

Using Mario AI framework
those variables defined



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

- States



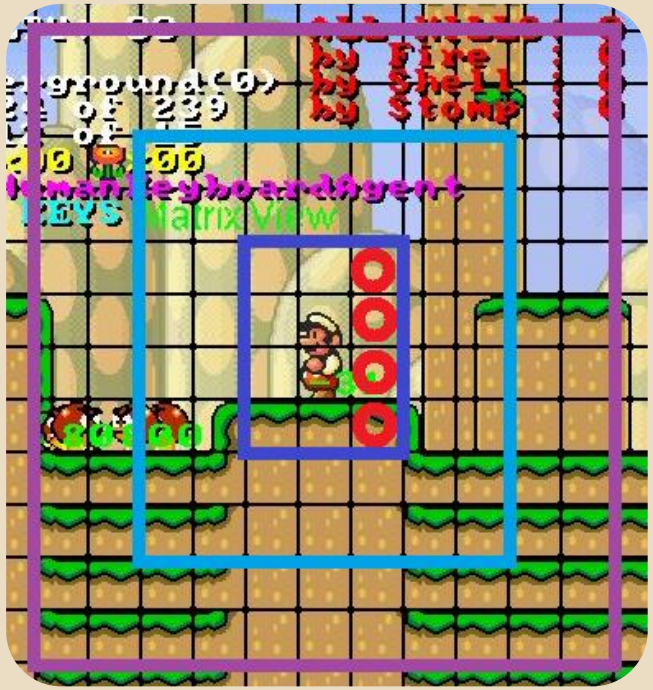
- States

$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

- Mario mode: 0 – small, 1 – big, 2-fire
- If on ground: 0 or 1
- If can jump: 0 or 1
- If stuck: 0 or 1
- Nearby, midrange, far enemies
- If enemies killed by stomp: 0 or 1

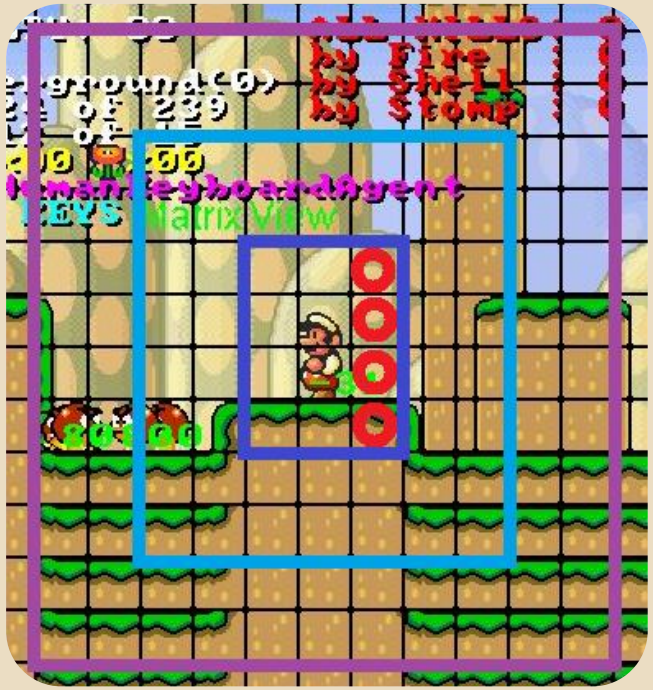
...

A state needs **39 bits** to encode



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

- States
- Actions

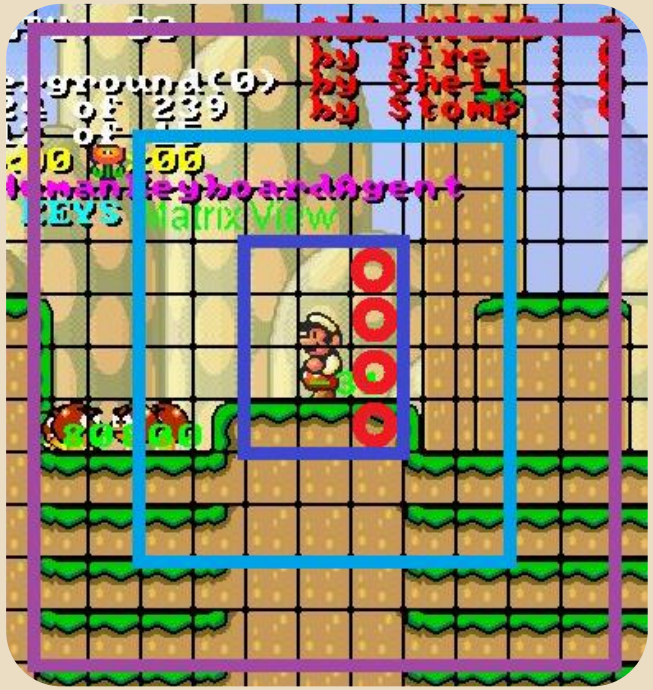


$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

- States
- Actions

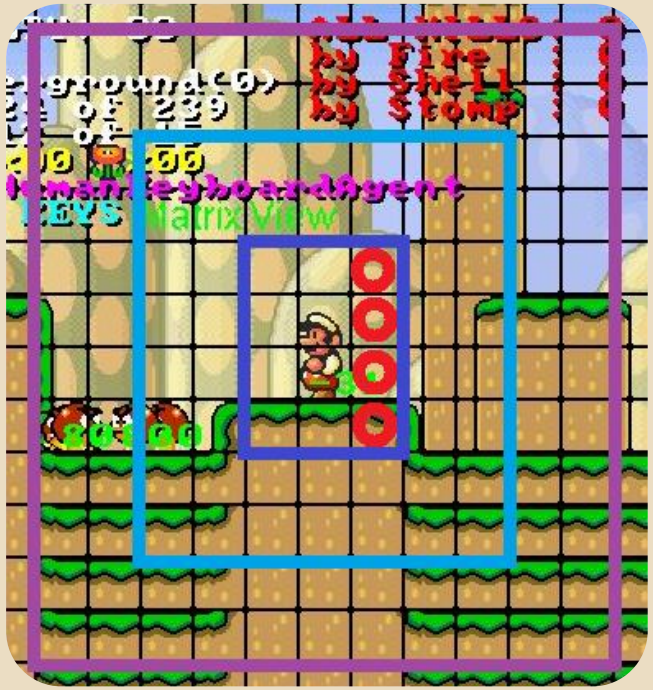
Mario performs one of 12 actions

{LEFT, RIGHT, STAY} X
{JUMP, NOT_JUMP} X
{SPEED/FIRE, NOSPEED} X



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

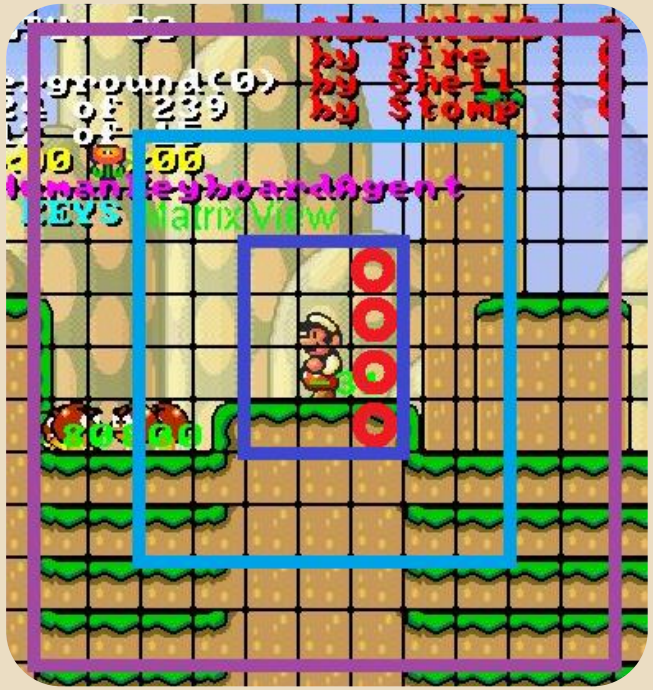
- States
- Actions
- Reward function



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

- States
- Actions
- Reward function

- Positive reward
  - Moving forward, jump on higher platforms
    - Reward decreases when there is an enemy nearby
  - Killing enemies
- Negative reward
  - Colliding with enemies
  - Being stuck



$$Q(s_t, a_t) \leftarrow (1 - a_{s,a})Q(s_t, a_t) + a_{s,a}(r + \gamma max(Q(s_{t+1}, a_{t+1})))$$

- States
- Actions
- Reward function
- Decreasing learning rate
- Discount factor

$$a(s_t, a_t) = \frac{a_0}{\#\ of\ times\ a_t\ performed\ in\ s_t}$$

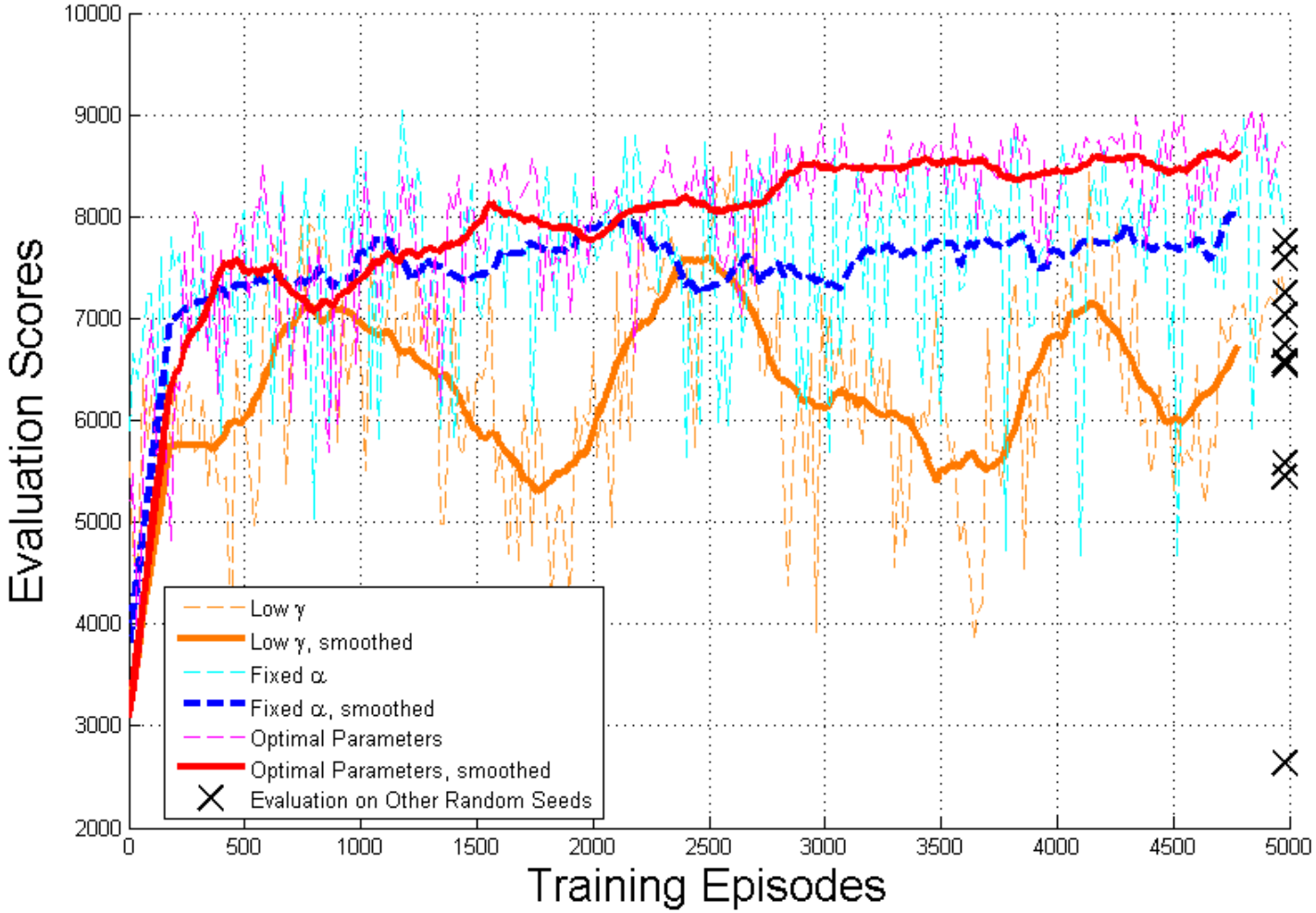
After several experiments
optimum values are selected

Reinforcement Learning to Play Mario [LYY12]

Reinforcement Learning to Play Mario [LYY12]

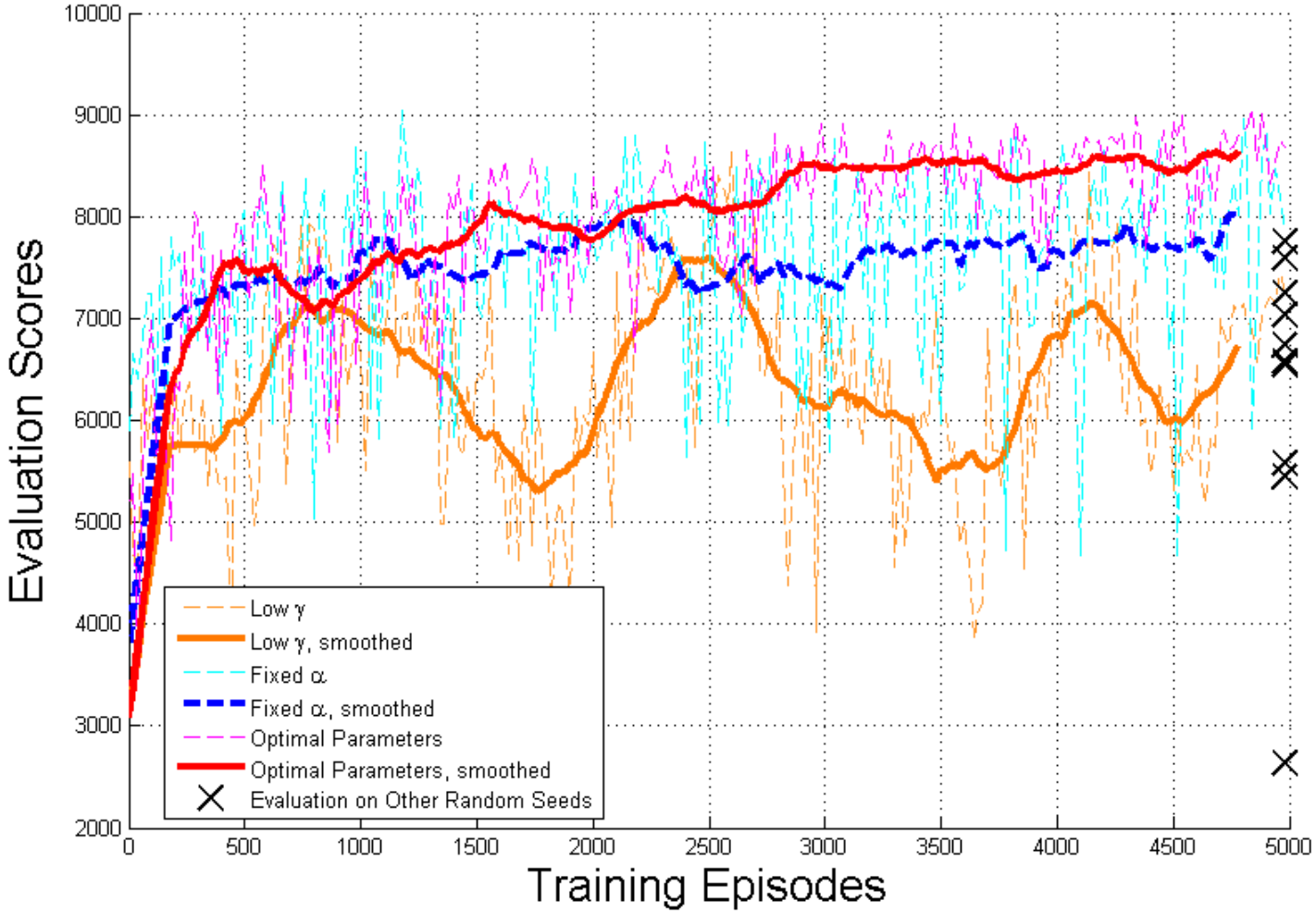
Reinforcement Learning to Play Mario [LYY12]

- Four measures
  - Composite score
  - Winning probability
  - % of Monster killed
  - Time spent



Reinforcement Learning to Play Mario [LYY12]

- Four measures
  - Composite score
    - Win status
    - Kills in total
    - Distance passed
    - Time spent in frames



High-level Reinforcement Learning in Strategy Games [AmS10]

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies
    - Change leader traits according to situtation

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies
    - Change leader traits according to situtation
  - Leave the low-level actions to build-in AI

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies
    - Change leader traits according to situtation
  - Leave the low-level actions to build-in AI
    - Create unit, upgrades etc.

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies
    - Change leader traits according to situtation
  - Leave the low-level actions to build-in AI
    - Create unit, upgrades etc.
- They compared 3 algorithms:

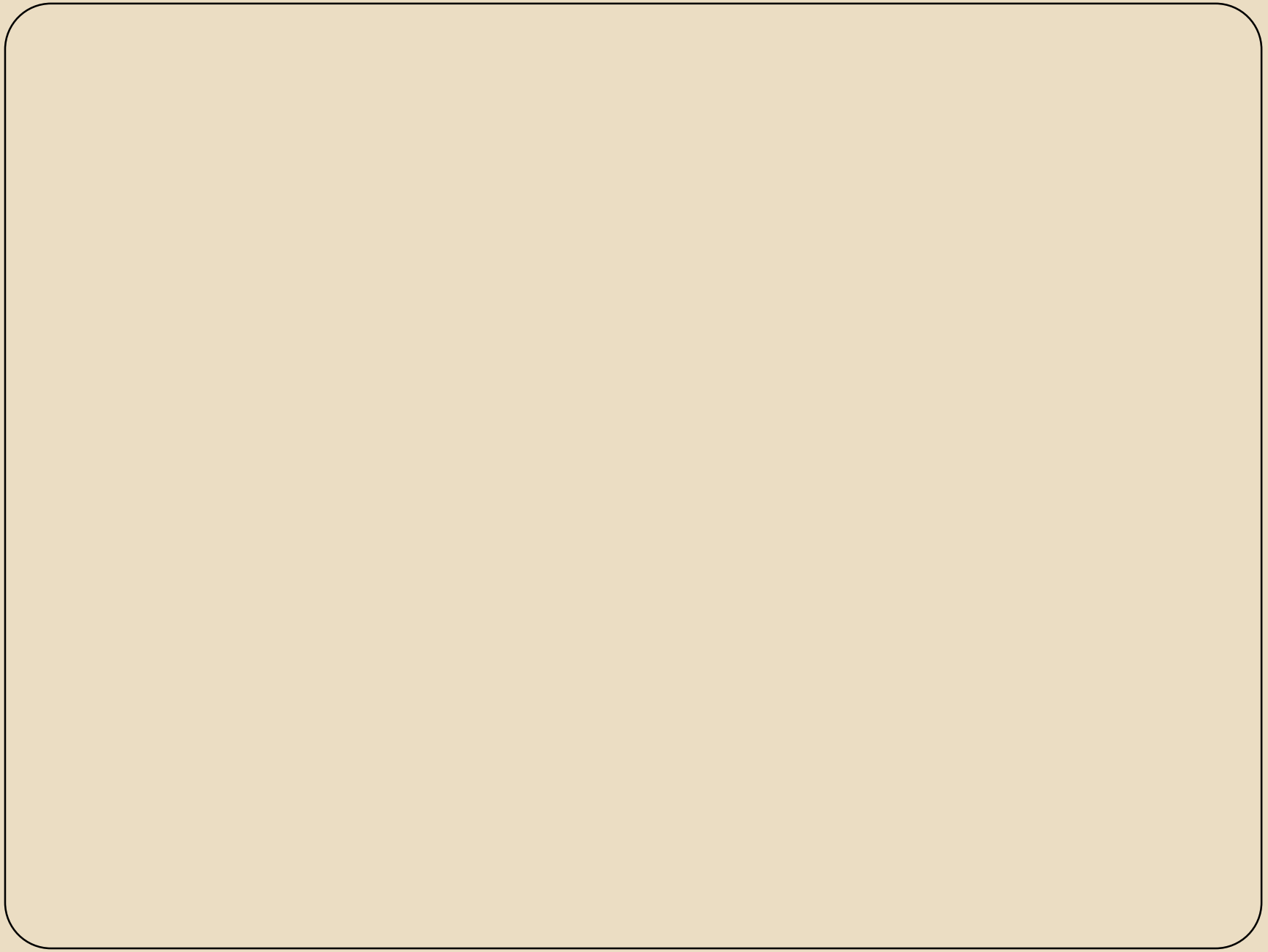
High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies
    - Change leader traits according to situtation
  - Leave the low-level actions to build-in AI
    - Create unit, upgrades etc.
- They compared 3 algorithms:
  - Q-learning vs Dyna-Q vs Dyna-Q with factored state representation

High-level Reinforcement Learning in Strategy Games [AmS10]

- Civilization IV, turn-based strategy games
  - Many paths to victory
- Goal is to improve built-in AI
  - Switching between high-level strategies
    - Change leader traits according to situtation
  - Leave the low-level actions to build-in AI
    - Create unit, upgrades etc.
- They compared 3 algorithms:
  - Q-learning vs Dyna-Q vs Dyna-Q with factored state representation
  - Focus only on high-level RL learning and abstraction of the game environment!

- State space



- State space

- 4 state features, in total 81 possible states

- State space

- 4 state features, in total 81 possible states
  - Population difference
  - Land diff.
  - Military power diff.
  - Remaining land

} All common and provided by game

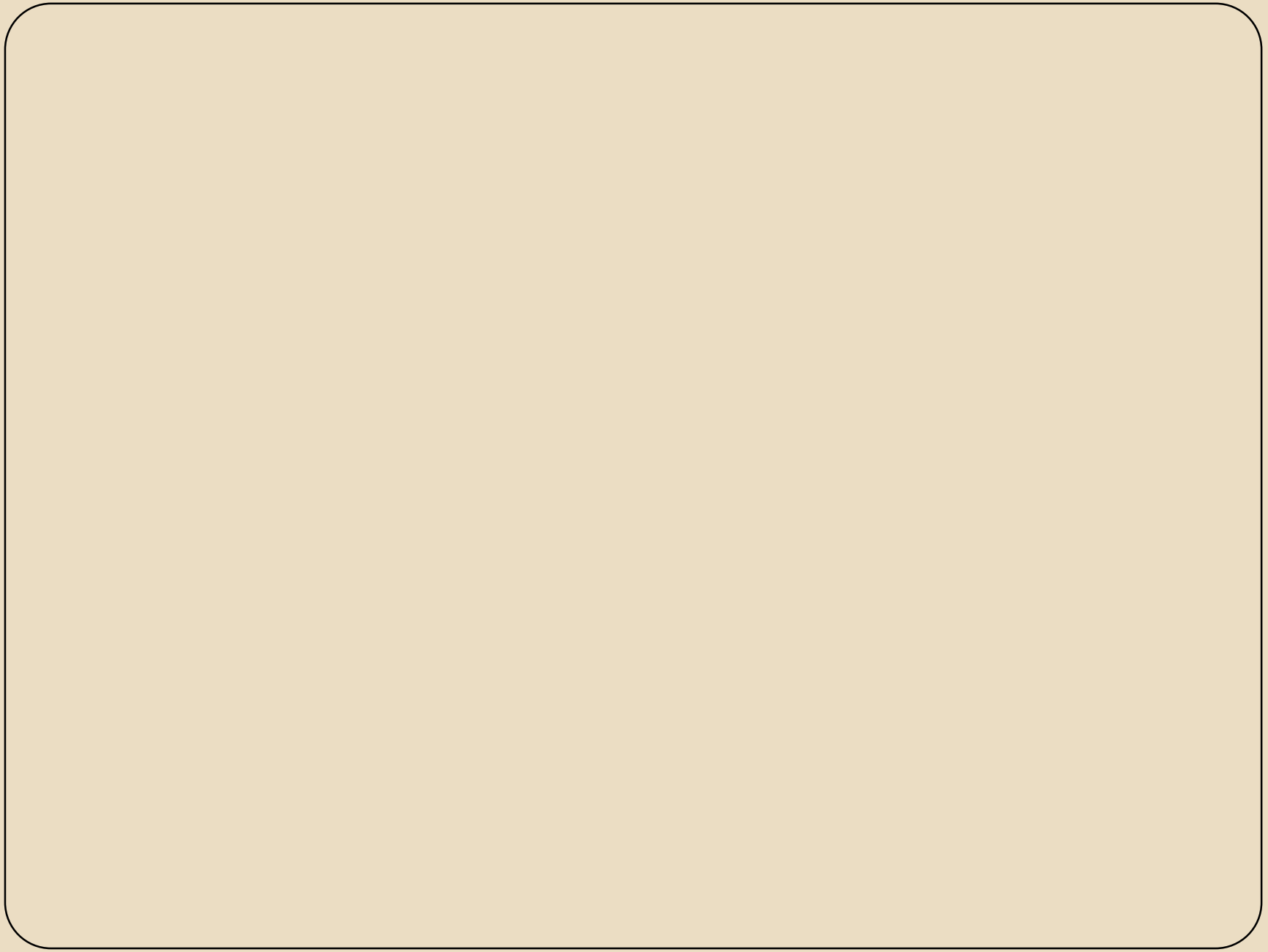
- State space

- 4 state features, in total 81 possible states
  - Population difference
  - Land diff.
  - Military power diff.
  - Remaining land

All common and provided by game

$$f_i = \begin{cases} 2, & if\ diff > \mathbf{10} \\ 1, & if\ -\mathbf{10} < diff < \mathbf{10} \\ 0, & if\ diff < -\mathbf{10} \end{cases}$$

- State space
- Action space



- State space
- Action space

- What is the Action?

- State space
- Action space

- What is the Action?
  - Choice of a new strategy (high-level)

- State space
- Action space

- What is the Action?
  - Choice of a new strategy (high-level)
  - For making low-level decisions
    - Build, explore, make war etc.

- State space
- Action space

- What is the Action?
  - Choice of a new strategy (high-level)
  - For making low-level decisions
    - Build, explore, make war etc.
- Limited to 4 leaders

- State space
- Action space

- What is the Action?
  - Choice of a new strategy (high-level)
  - For making low-level decisions
    - Build, explore, make war etc.
- Limited to 4 leaders
  - Covers all 8 possible leader traits

- State space
- Action space

- What is the Action?
  - Choice of a new strategy (high-level)
  - For making low-level decisions
    - Build, explore, make war etc.
- Limited to 4 leaders
  - Covers all 8 possible leader traits
  - For example

- State space
- Action space

- What is the Action?
  - Choice of a new strategy (high-level)
  - For making low-level decisions
    - Build, explore, make war etc.
- Limited to 4 leaders
  - Covers all 8 possible leader traits
  - For example
    - Washington is Financial and Organized

- State space
- Action space
- Reward model

- State space
- Action space
- Reward model

- Reward based on the score provided by game

- State space
- Action space
- Reward model

- Reward based on the score provided by game
  - Immediate reward, given after each step as

- State space
- Action space
- Reward model

- Reward based on the score provided by game
  - Immediate reward, given after each step as

thisStepScore = myTotalScore – yourTotalScore

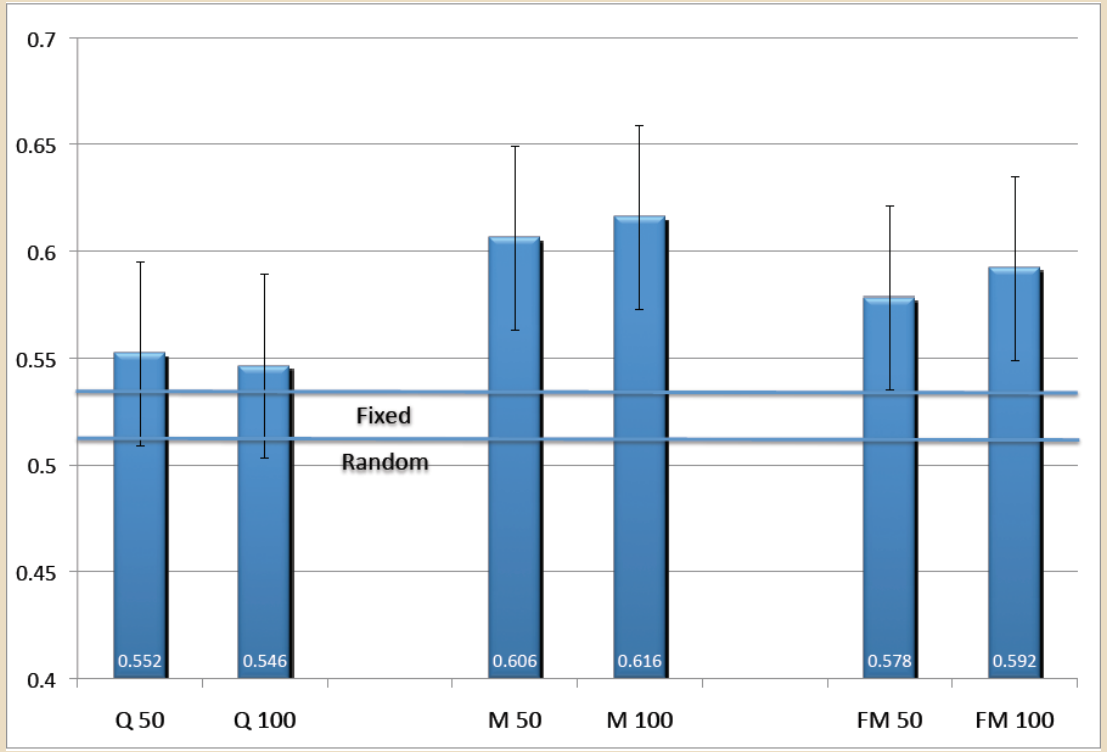
- State space
- Action space
- Reward model

- Reward based on the score provided by game
  - Immediate reward, given after each step as

thisStepScore = myTotalScore – yourTotalScore

- Still possible to lose the game while having high score

High-level Reinforcement Learning in Strategy Games [AmS10]



versus

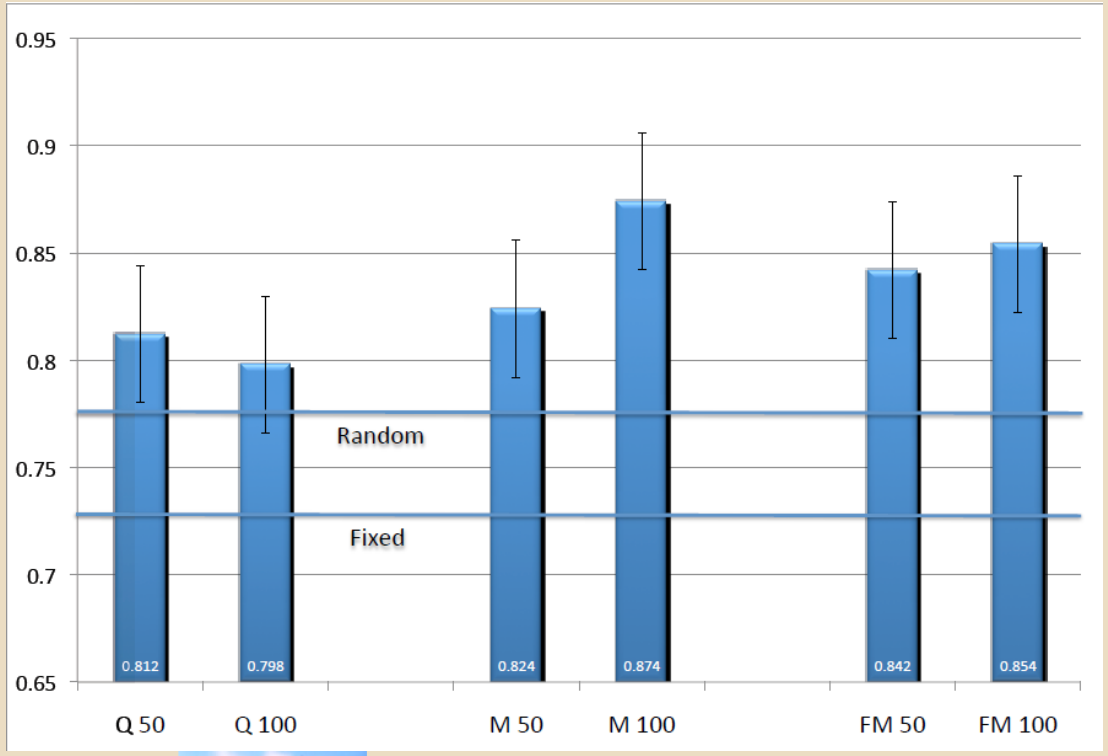
Fredrick

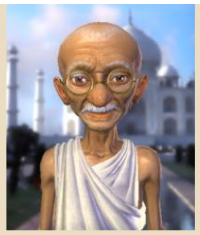
- Organized
- Philosophical

Washington

- Expansive
- Charismatic

High-level Reinforcement Learning in Strategy Games [AmS10]





versus

Gandhi

- Spiritual
- Philosophical

Genghis Khan

- Aggressive
- Imperialistic

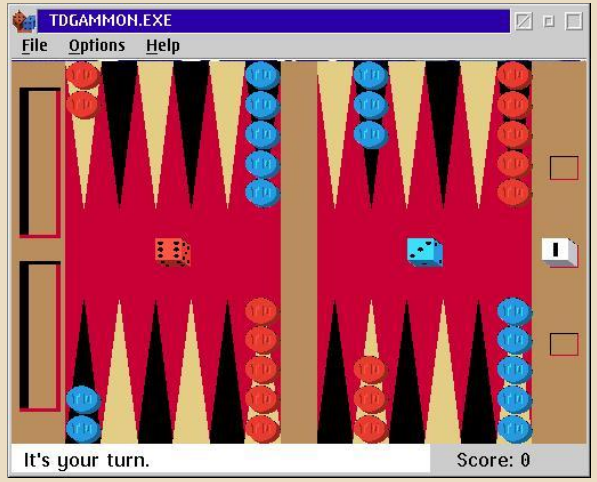
Temporal Difference Learning and TD-Gammon [Tes95]

- $TD(\lambda)$ algorithm and NN
- Helped set the groundwork for many advancements made
- Referenced in both Atari and AlphaGo papers

- 10^{20}

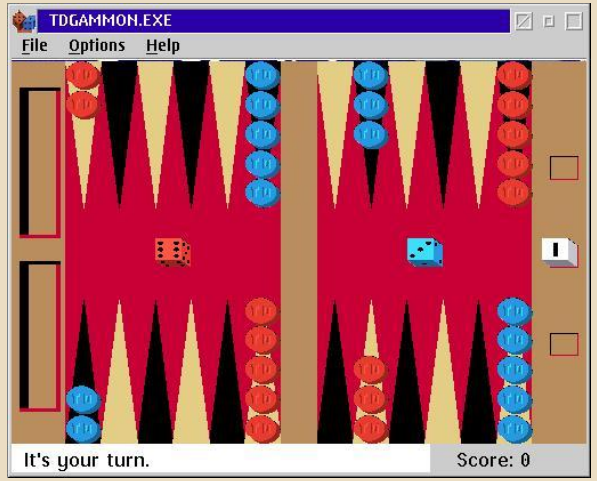
Temporal Difference Learning and TD-Gammon [Tes95]

- Backgammon playing agent
  - $TD(\lambda)$ algorithm and NN
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
  - 10^{20}



Temporal Difference Learning and TD-Gammon [Tes95]

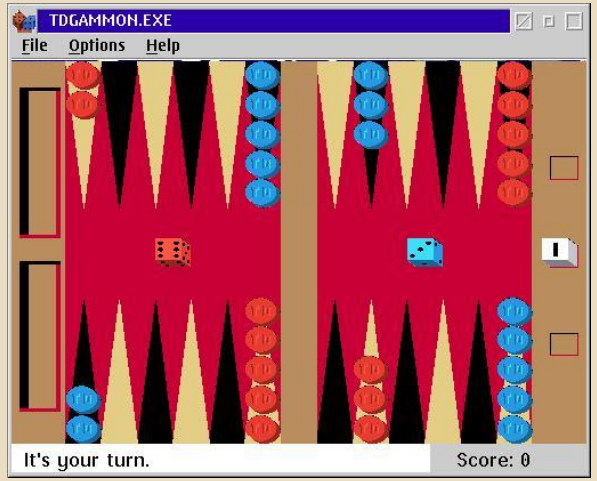
- Backgammon playing agent
- First successful NN & RL approach
  - $TD(\lambda)$ algorithm and NN
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers



  - 10^{20}

Temporal Difference Learning and TD-Gammon [Tes95]

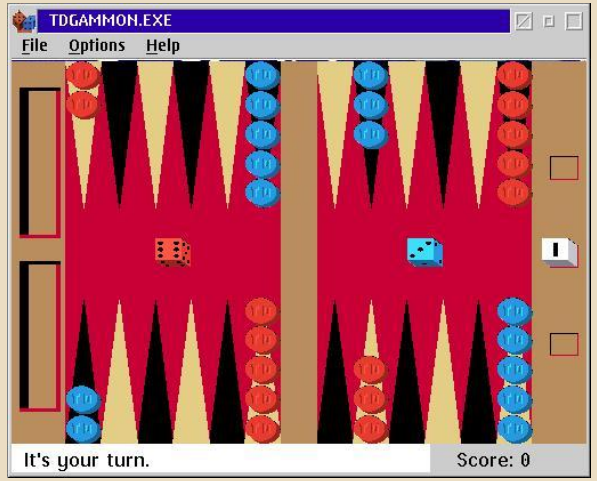
- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Combination of $TD(\lambda)$ algorithm and NN
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers



  - 10^{20}

Temporal Difference Learning and TD-Gammon [Tes95]

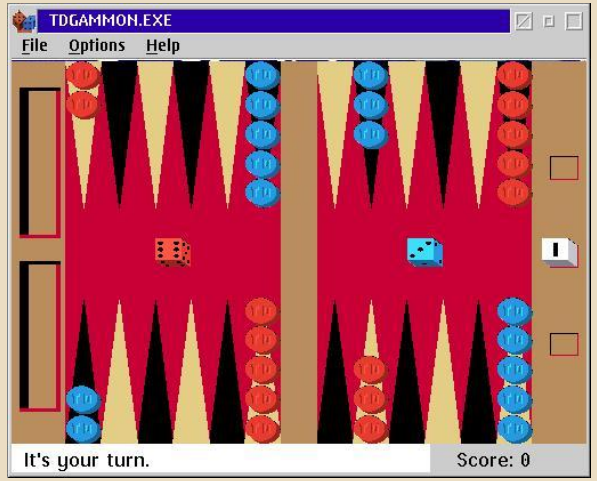
- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers



  - 10^{20}

Temporal Difference Learning and TD-Gammon [Tes95]

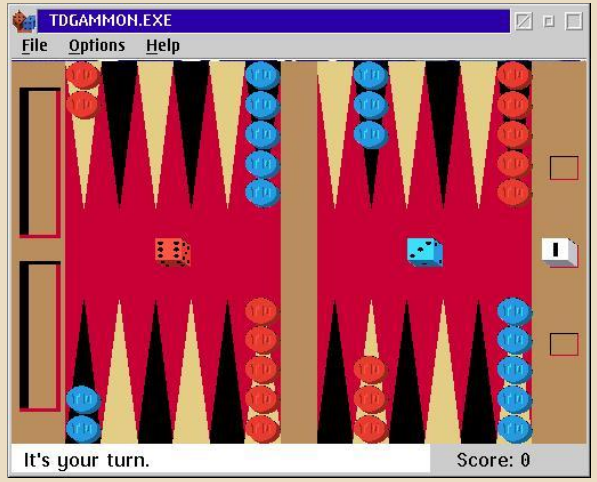
- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
  - Referenced in both Atari and AlphaGo papers



  - 10^{20}

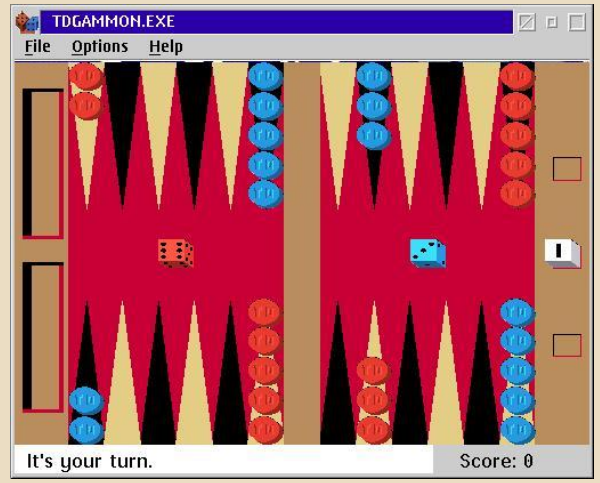
Temporal Difference Learning and TD-Gammon [Tes95]

- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
- Other approaches did not work well
  - 10^{20}



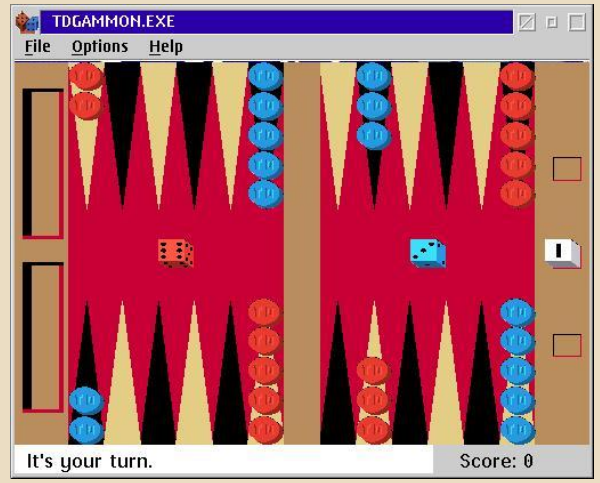
Temporal Difference Learning and TD-Gammon [Tes95]

- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
- Other approaches did not work well
  - Brute-force not feasible
  - 10^{20}



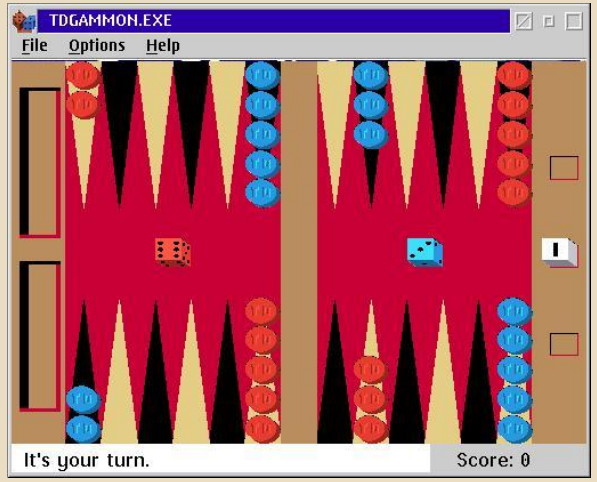
Temporal Difference Learning and TD-Gammon [Tes95]

- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
- Other approaches did not work well
  - Brute-force not feasible
    - High branching ratio: 21 possible dice combinations
  - 10^{20}



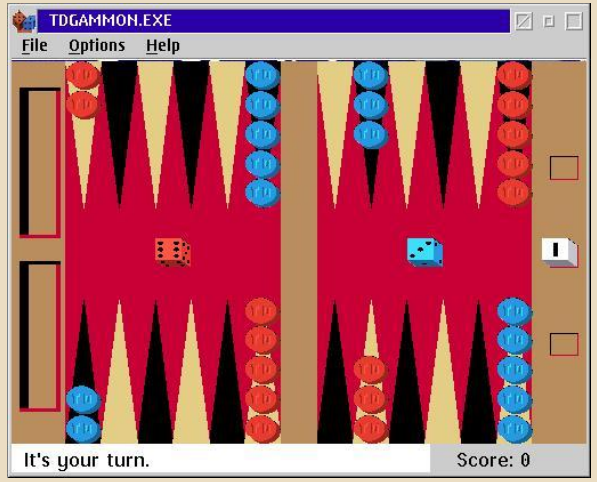
Temporal Difference Learning and TD-Gammon [Tes95]

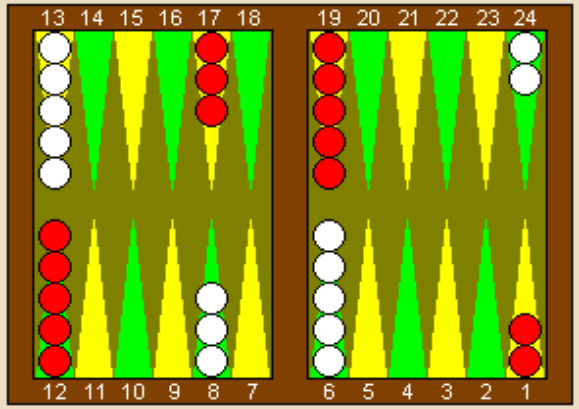
- 10 20 10 10 20 20 10 20
- $TTDD$ λ λ λ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
- Other approaches did not work well
  - Brute-force not feasible
    - High branching ratio: 21 possible dice combinations
  - Huge search space ≈ 10^{20}
  - 10^{20}



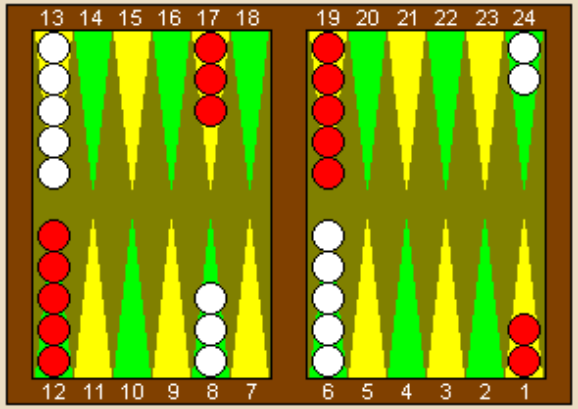
Temporal Difference Learning and TD-Gammon [Tes95]

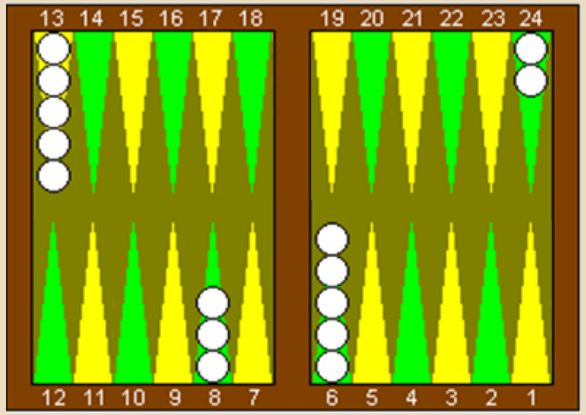
- 10 20 10 10 20 20 10 20
- $TTDD\ \lambda\ \lambda\ \lambda$ algorithm and NN
- Backgammon playing agent
- First successful NN & RL approach
  - Helped set the groundwork for many advancements made
  - Referenced in both Atari and AlphaGo papers
- Other approaches did not work well
  - Brute-force not feasible
    - High branching ratio: 21 possible dice combinations
  - Supervised method, hand-crafted features achieved only high-intermediate level
  - 10^{20}



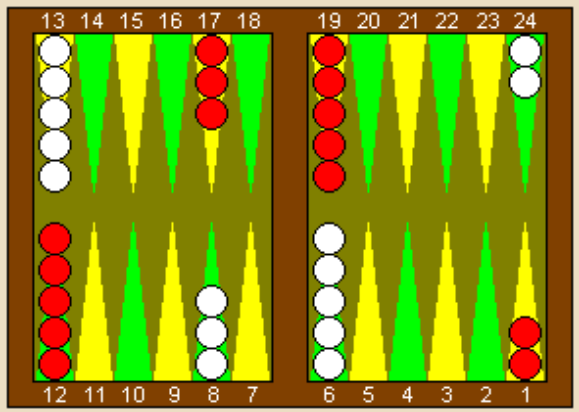

13 14 15 16 17 18
19 20 21 22 23 24
12 11 10 9 8 7
6 5 4 3 2 1

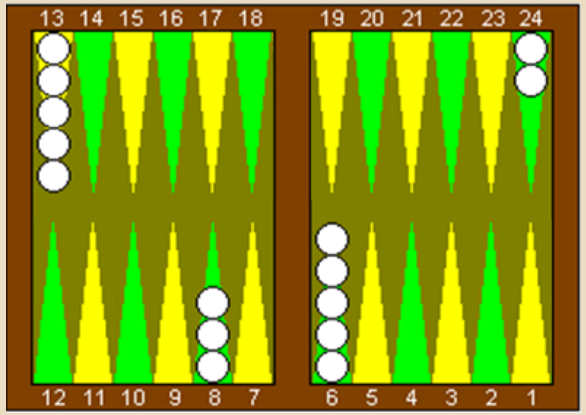
Only one side



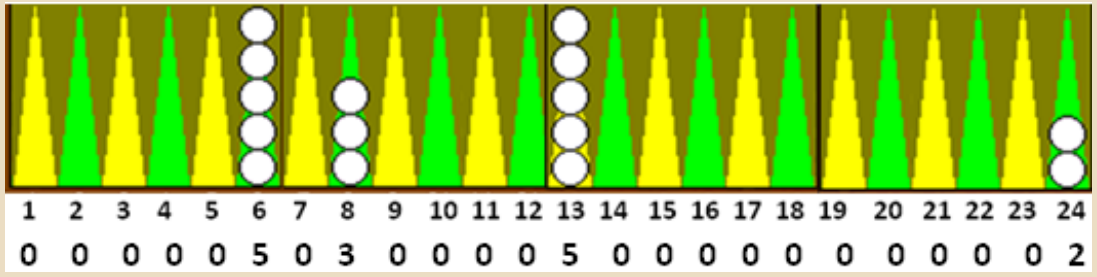


Only one side



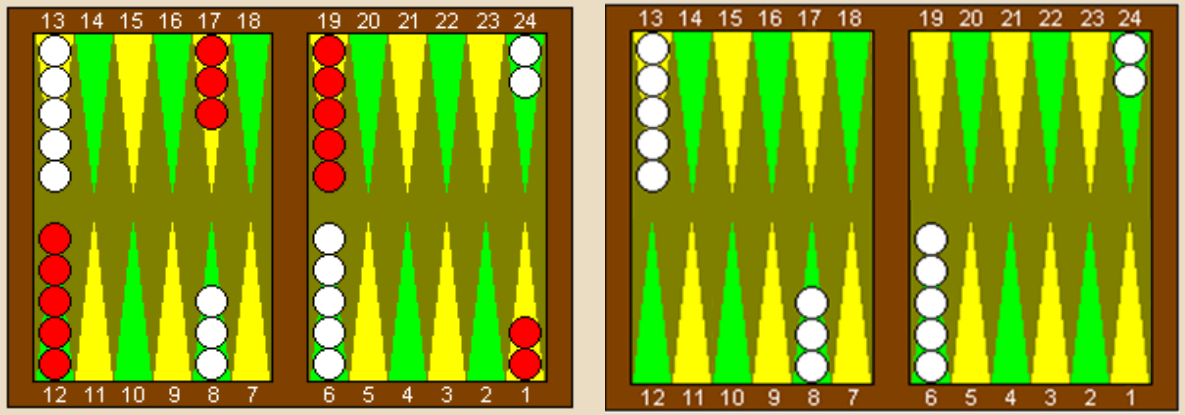


Checkers position as feature vectors

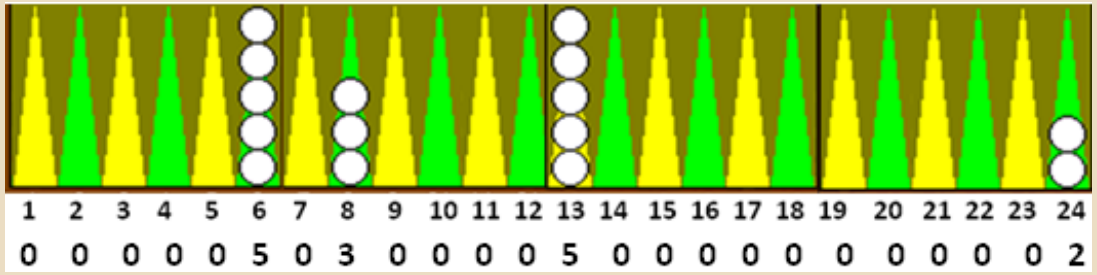


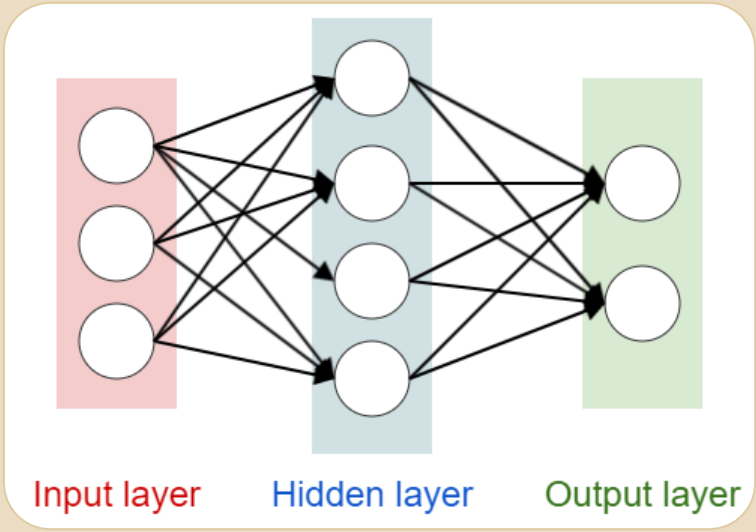
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	0	0	0	5	0	3	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	2

Only one side

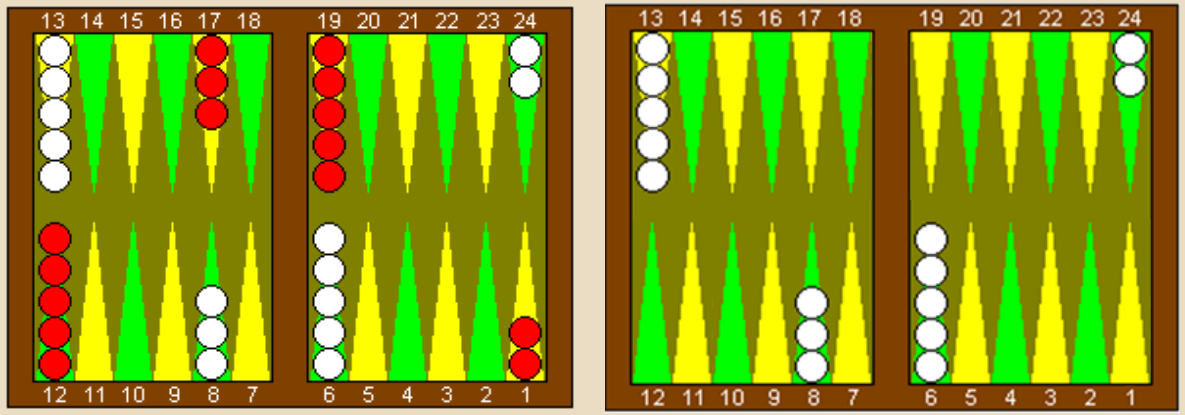


Checkers position as feature vectors

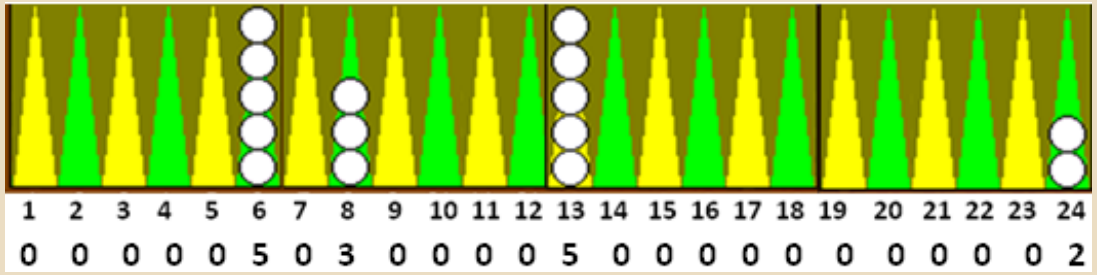


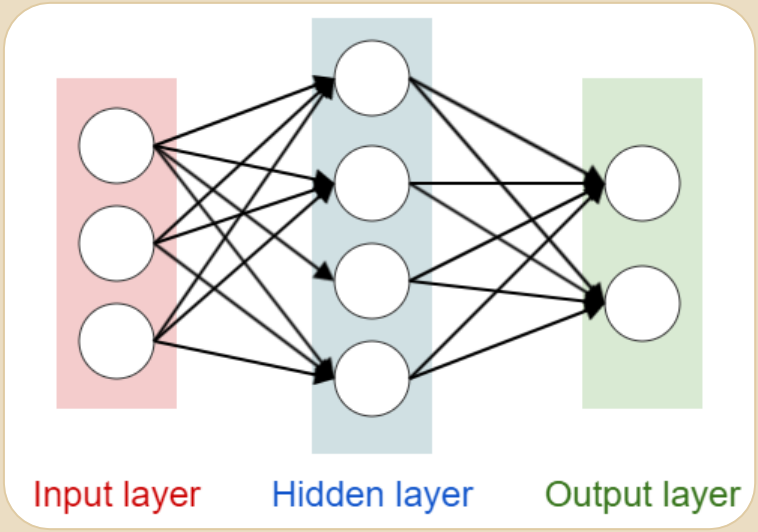


Only one side

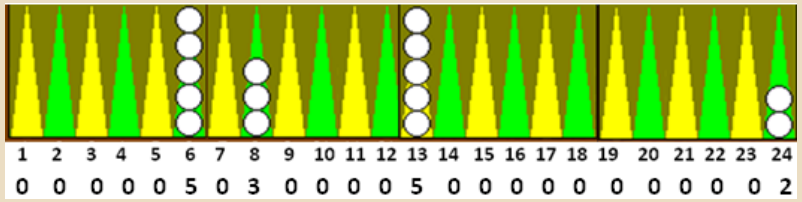


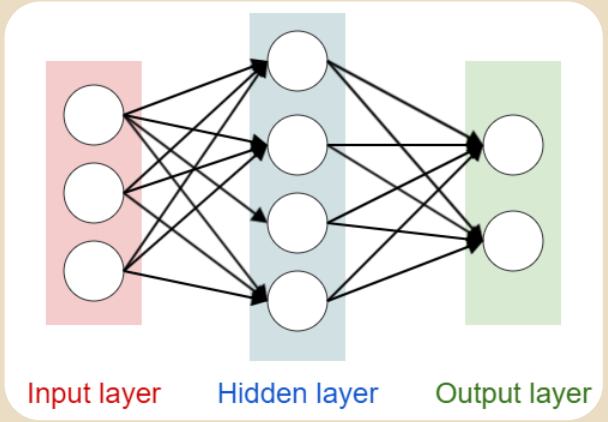
Checkers position as feature vectors



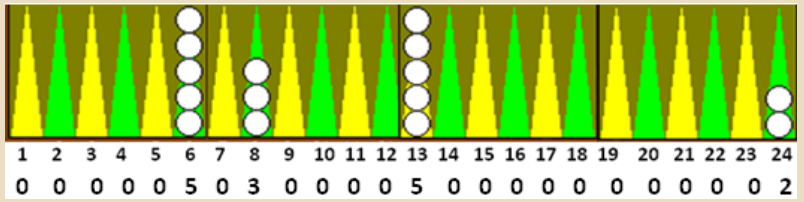


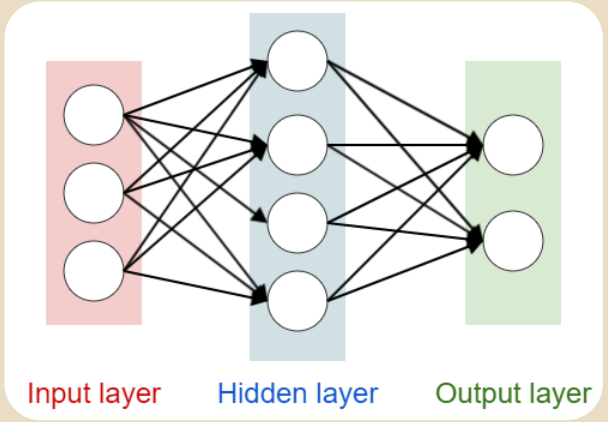
White	Normal
Red	Gammon





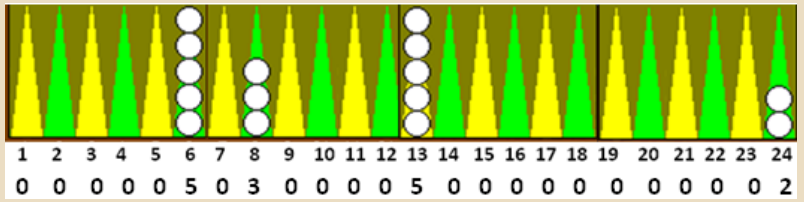
- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

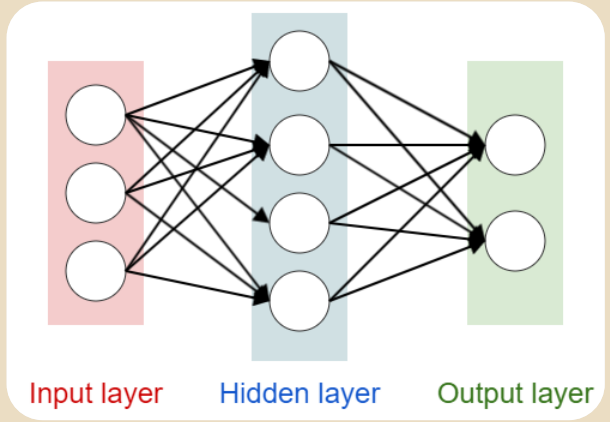




- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

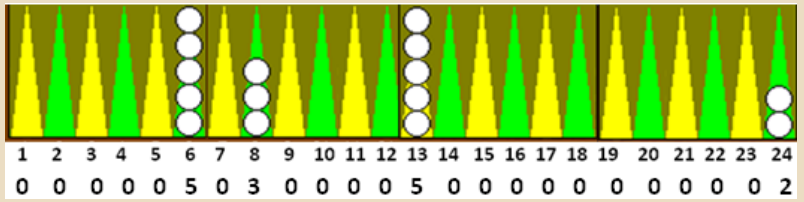


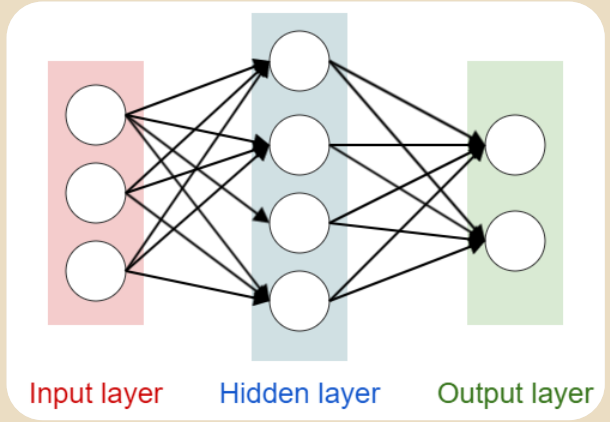


- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$



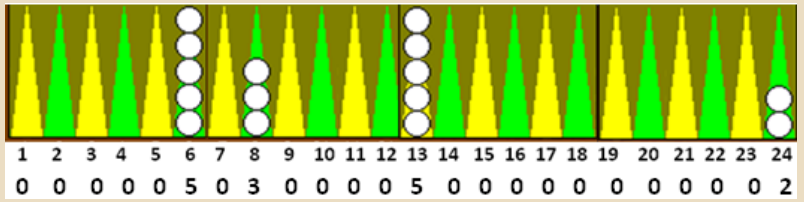


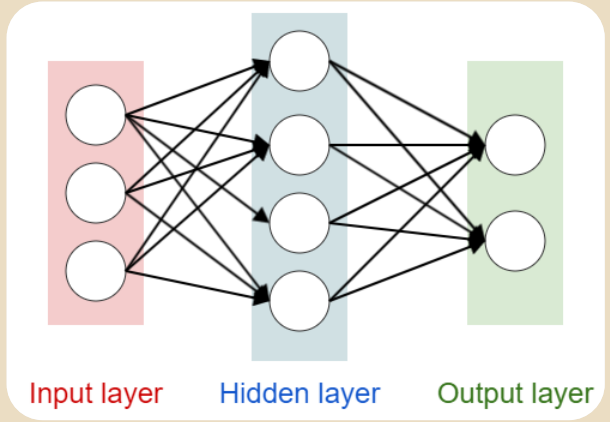
- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

$$\underbrace{w_{t+1} - w_t} = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

to change the weight from its value on the previous turn



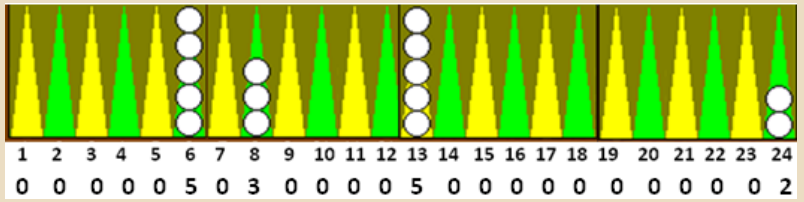


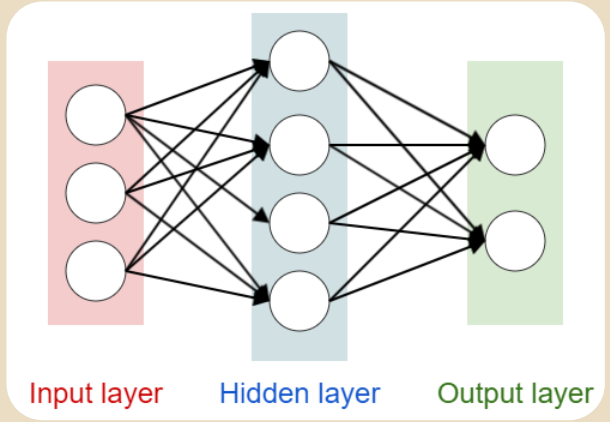
- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

The difference between current and previous turn's board evaluations



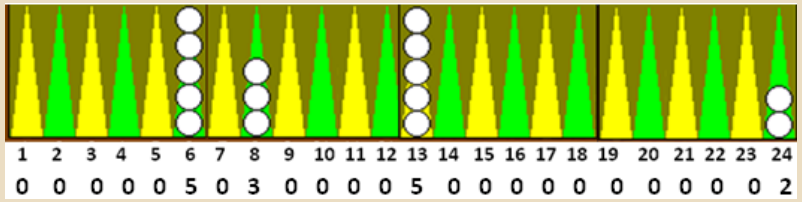


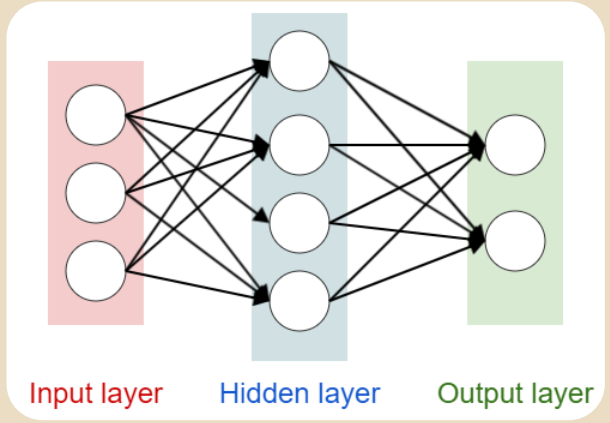
- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

Learning rate





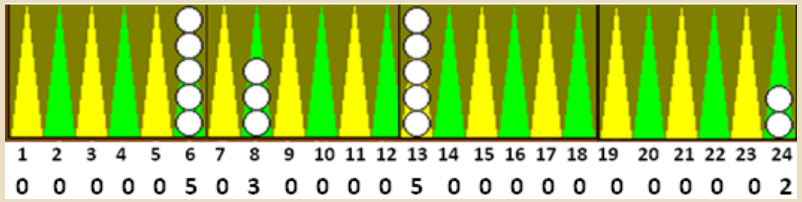
- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

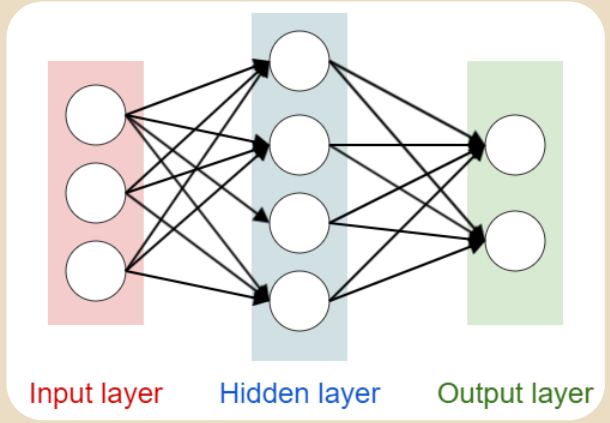
- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

Controls the decay of older estimates.

If => 0
Corrects only the previous turn's estimate
If => 1
Attempts to correct all previous turns



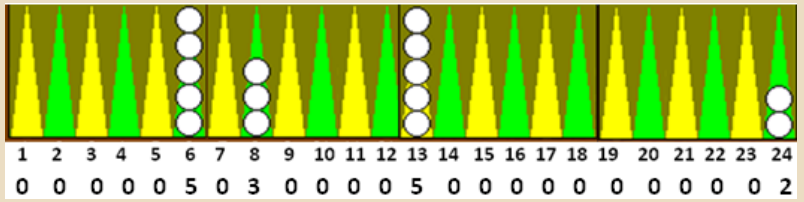


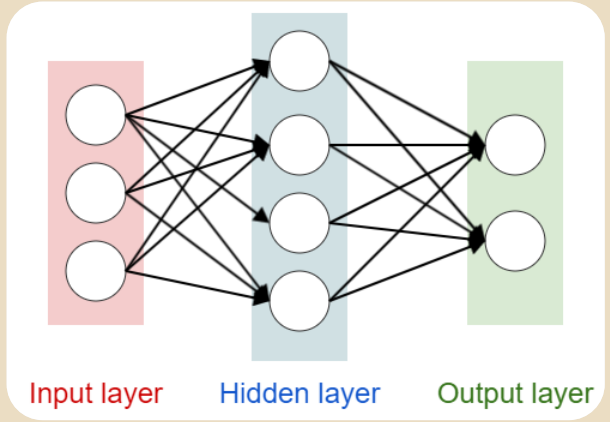
- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

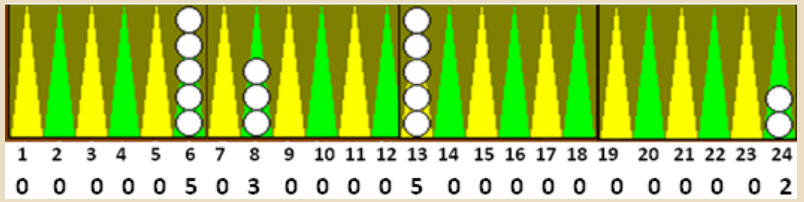
The gradient of NN output w.r.t to weights

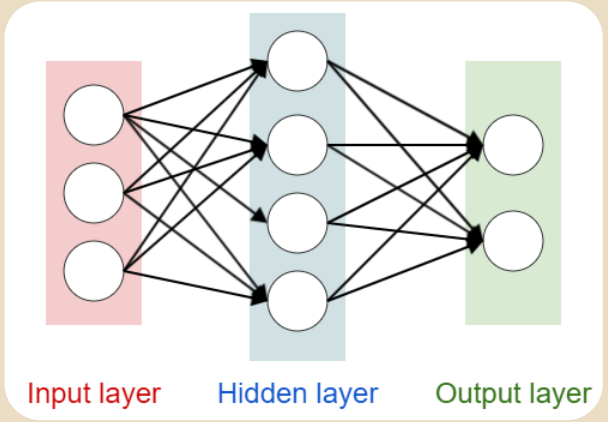




- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

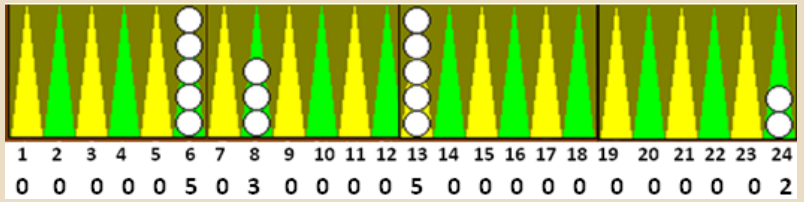
- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:
- First chooses any legal move

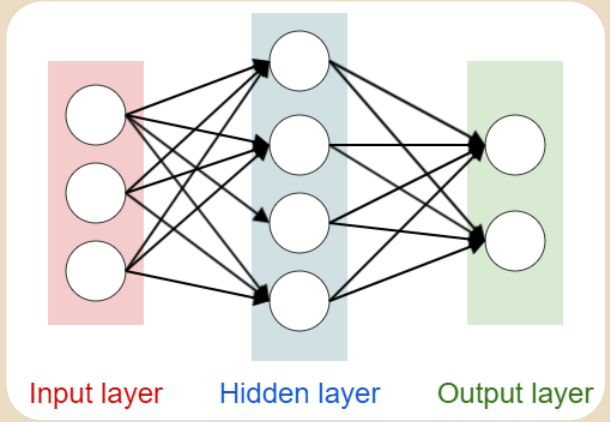




- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:
- First chooses any legal move
  - After NN weights updated (backpropagation), picks the maximum weighted corresponding move.





- Pr(White wins)
- Pr(White gammons)
- Pr(Red wins)
- Pr(Red gammons)

- After each turn, each weight in NN gets updated according to $\mathrm{TD}(\lambda)$:
- First chooses any legal move
  - After NN weights updated (backpropagation), picks the maximum weighted corresponding move.
- Self-training scheme and stochastic environment of the game improves the NN

Temporal Difference Learning and TD-Gammon [Tes95]

Program	Training Games	Opponents	Results
TDG 1.0	300,000	Robertie, Davis, Magriel	-13 pts/51 games (-0.25 ppg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0.18 ppg)
TDG 2.1	1,500,000	Robertie	-1 pt/40 games (-0.02 ppg)

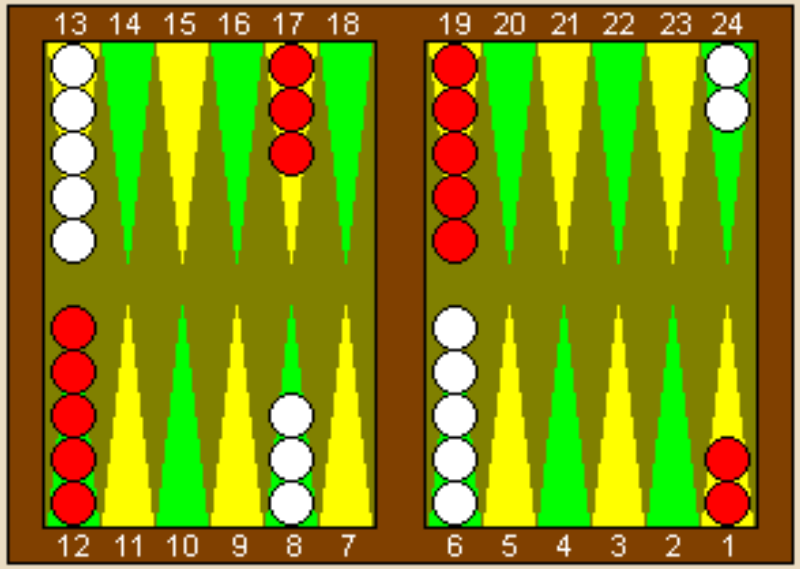
Temporal Difference Learning and TD-Gammon [Tes95]

Program	Training Games	Opponents	Results
TDG 1.0	300,000	Robertie, Davis, Magriel	-13 pts/51 games (-0.25 ppg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0.18 ppg)
TDG 2.1	1,500,000	Robertie	-1 pt/40 games (-0.02 ppg)

Addition!

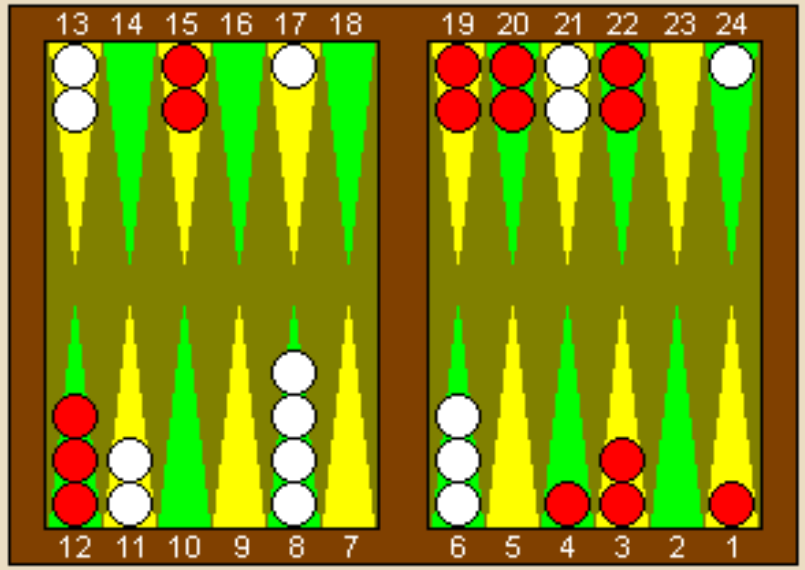
Later in TDG 3.0 using 160 hidden nodes and a selective 3-ply search, managed to beat Kazaros +6pts /20 games

Temporal Difference Learning and TD-Gammon [Tes95]



White is to play 4 - 1

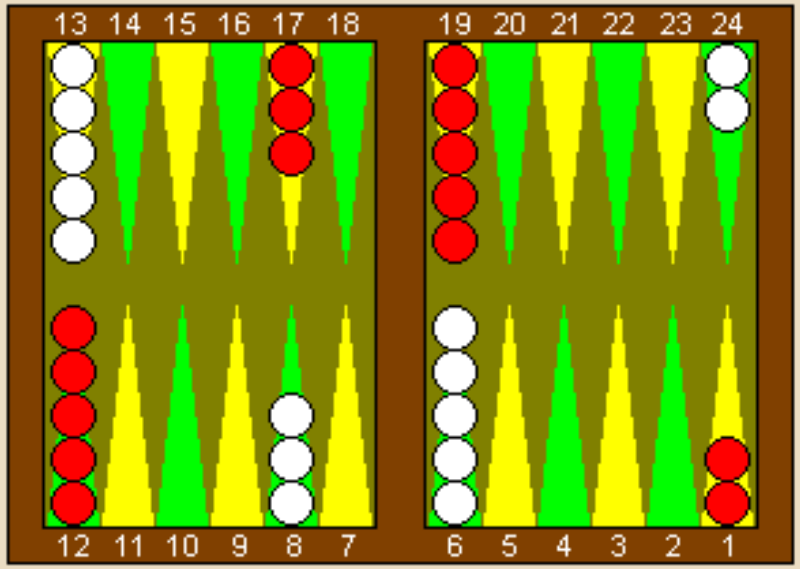
Move	Estimate	Rollout
13-9, 6-5 13-9, 24-23	−0.014 +0.005	−0.040 +0.005



White is to play 4 - 4

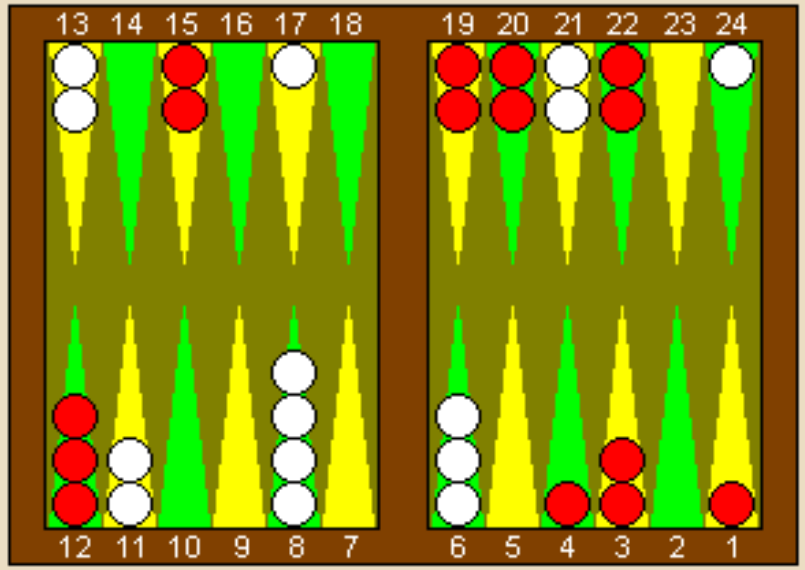
Move	Estimate	Rollout
8-4*, 8-4, 11-7, 11-7 8-4*, 8-4, 21-17, 21-17	+0.184 +0.238	+0.139 +0.221

Temporal Difference Learning and TD-Gammon [Tes95]



White is to play 4 - 1

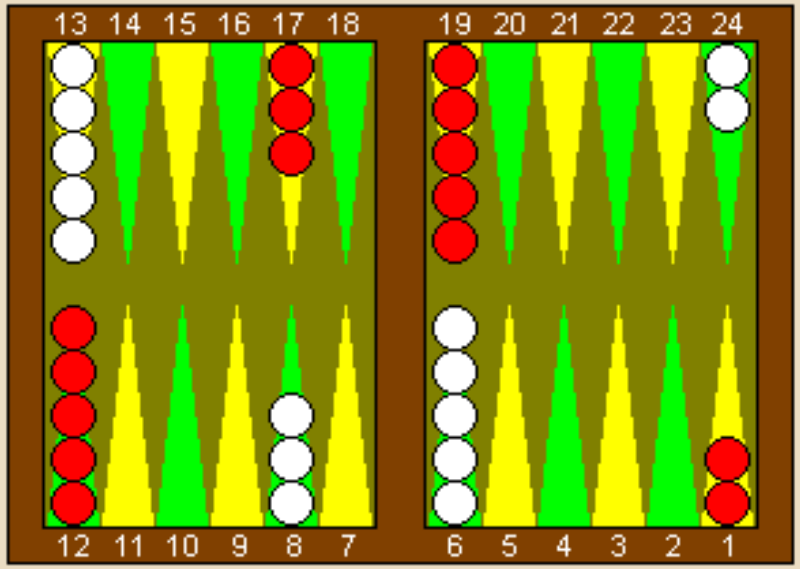
Move	Estimate	Rollout
13-9, 6-5 13-9, 24-23	−0.014 +0.005	−0.040 +0.005



White is to play 4 - 4

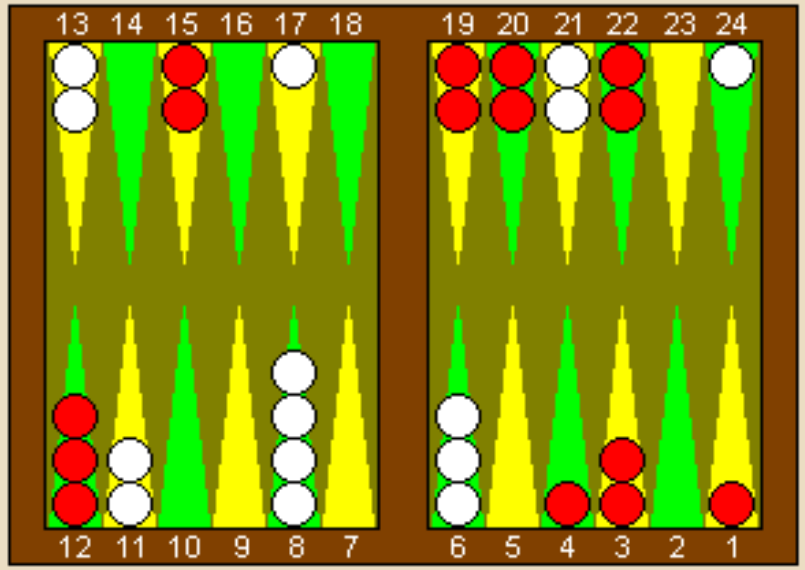
Move	Estimate	Rollout
8-4*, 8-4, 11-7, 11-7 8-4*, 8-4, 21-17, 21-17	+0.184 +0.238	+0.139 +0.221

Temporal Difference Learning and TD-Gammon [Tes95]



White is to play 4 - 1

Move	Estimate	Rollout
13-9, 6-5 13-9, 24-23	−0.014 +0.005	−0.040 +0.005



White is to play 4 - 4

Move	Estimate	Rollout
8-4*, 8-4, 11-7, 11-7 8-4*, 8-4, 21-17, 21-17	+0.184 +0.238	+0.139 +0.221

Playing Atari with Reinforcement Learning [MKS13]

- Context-free

Playing Atari with Reinforcement Learning [MKS13]

- Context-free
  - Input only from sensory data

Playing Atari with Reinforcement Learning [MKS13]

- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)

Playing Atari with Reinforcement Learning [MKS13]

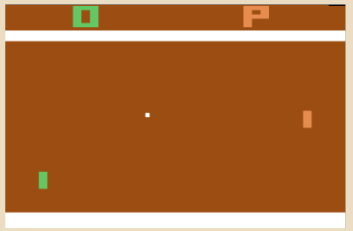
- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal

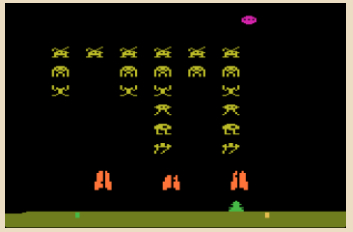
Playing Atari with Reinforcement Learning [MKS13]

- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.

Playing Atari with Reinforcement Learning [MKS13]

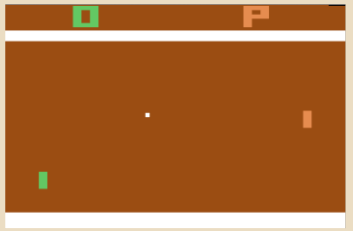
- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.
  - 7 Atari games, implemented in the Arcade Learning Environment

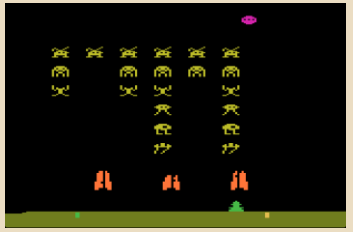




Playing Atari with Reinforcement Learning [MKS13]

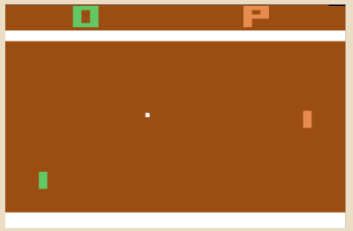
- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.
  - 7 Atari games, implemented in the Arcade Learning Environment
- Using RL and DL together seems problematic

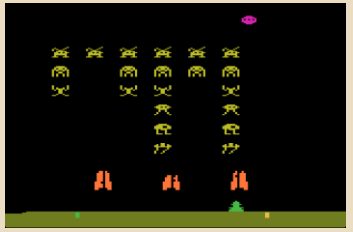




Playing Atari with Reinforcement Learning [MKS13]

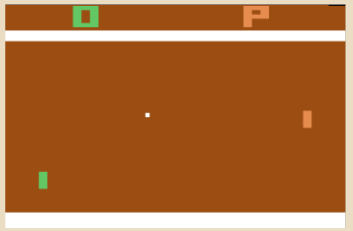
- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.
  - 7 Atari games, implemented in the Arcade Learning Environment
- Using RL and DL together seems problematic
  - Highly correlated states vs data samples are independent

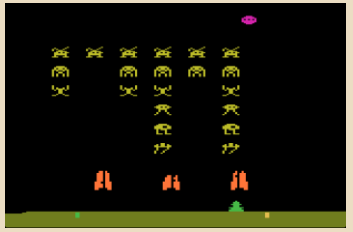




Playing Atari with Reinforcement Learning [MKS13]

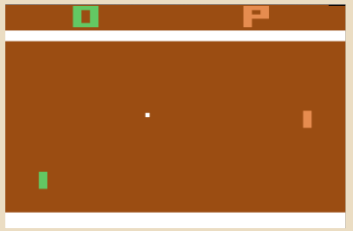
- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.
  - 7 Atari games, implemented in the Arcade Learning Environment
- Using RL and DL together seems problematic
  - Highly correlated states vs data samples are independent
  - Distribution changes vs fixed underliying distribution

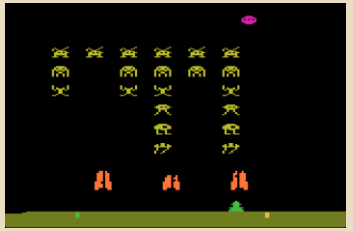




Playing Atari with Reinforcement Learning [MKS13]

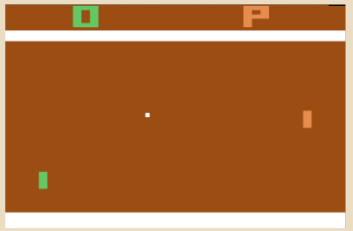
- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.
  - 7 Atari games, implemented in the Arcade Learning Environment
- Using RL and DL together seems problematic
  - Highly correlated states vs data samples are independent
  - Distribution changes vs fixed underliying distribution
- Experience replay mechanism

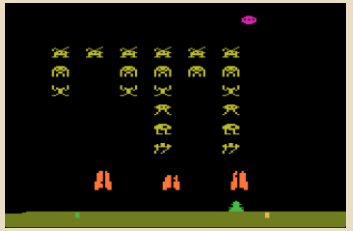




Playing Atari with Reinforcement Learning [MKS13]

- Context-free
  - Input only from sensory data
  - Deep Q-network (DQN)
- Main goal
  - Successfully learn to play as much games as possible.
  - 7 Atari games, implemented in the Arcade Learning Environment
- Using RL and DL together seems problematic
  - Highly correlated states vs data samples are independent
  - Distribution changes vs fixed underliying distribution
- Experience replay mechanism
  - Randomly sample previous transition = smooth the distribution





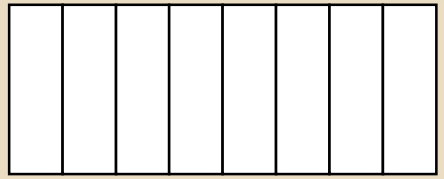
Algorithm 1 Deep Q-learning with Experience Replay

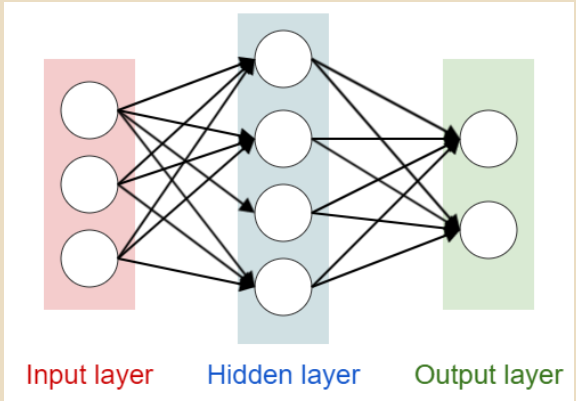
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for

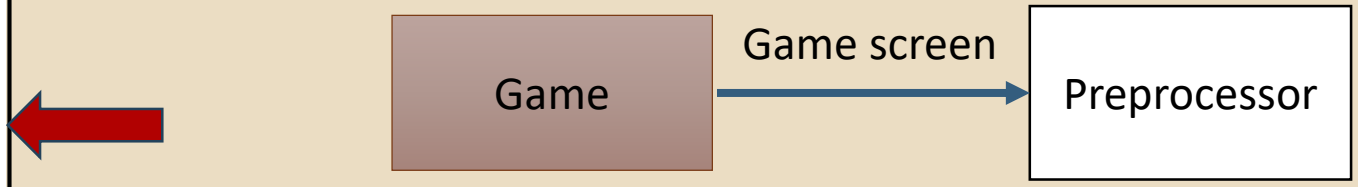
Replay memory





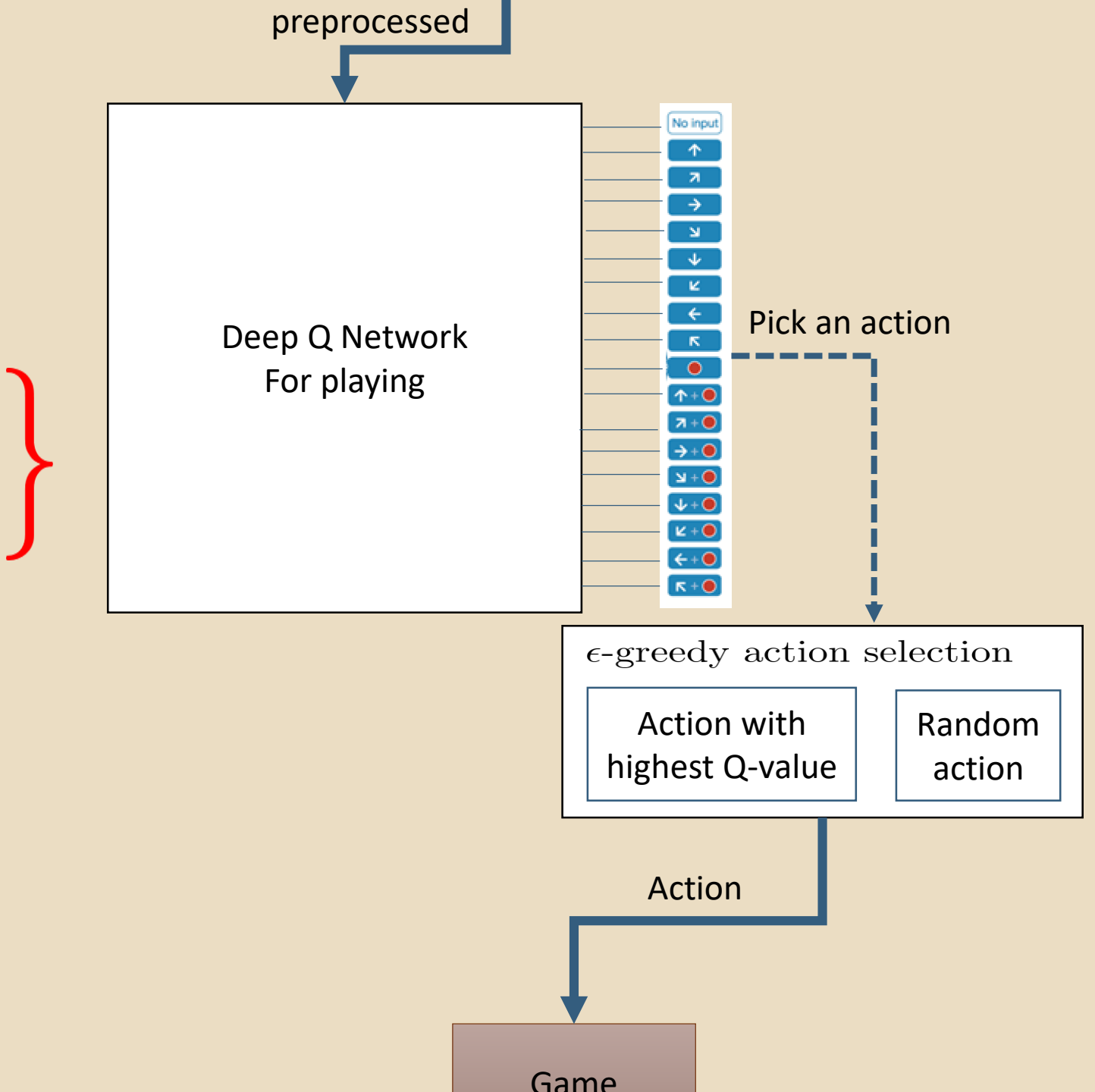
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for



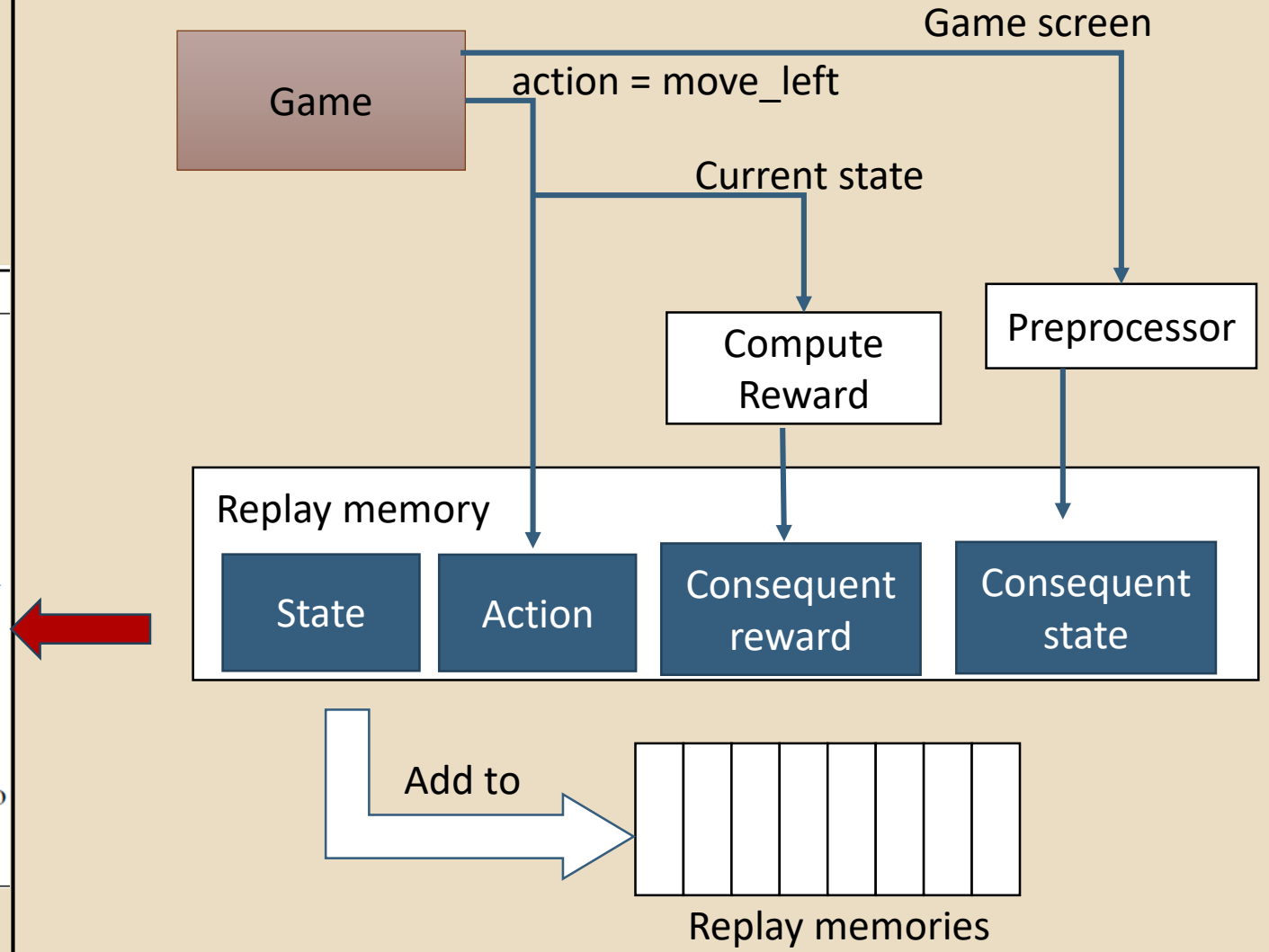
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for



Algorithm 1 Deep Q-learning with Experience Replay

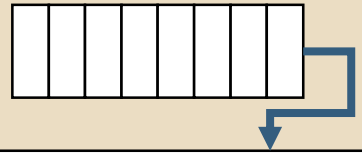
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for

For each replay memory «j»



Replay memory

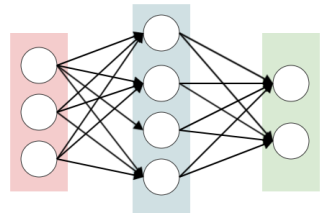
State | Action | Consequent reward | Consequent state

Smooth training distribution

If terminal memory -> current reward
Otherwise; update w.r.t. Max Q value of A

Compute loss
and Update network

Gradient descent



Playing Atari with Reinforcement Learning [MKS13]

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690

Average total reward

Playing Atari with Reinforcement Learning [MKS13]

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690

Average total reward

HNeat Best [8]	3616	52	106	19	1800	920	**1720**
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Neuro-evolution approach comparison

Conclusion

- Summary
- Some problems in articles
  - Evaluation
  - Why did TD-Gammon work? [PoB97]
- What to expect next?
  - Faster, stronger, less HW requirements
  - Sandbox learning

Thank you for listening!

Questions?

References

[AmS10] Amato, C. and Shani, G., High-level reinforcement learning in strategy games. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, 2010, pages 75-82.

[LYY12] Liao, Y., Yi, K. and Yang, Z., Cs229 nal report reinforcement learning to play mario. Technical Report, Technical report, Stanford University, 2012.

[Tes95] Tesauro, G., Temporal dierence learning and td-gammon. Communications of the ACM, 38,3(1995), pages 58-68.

[MKS13] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602