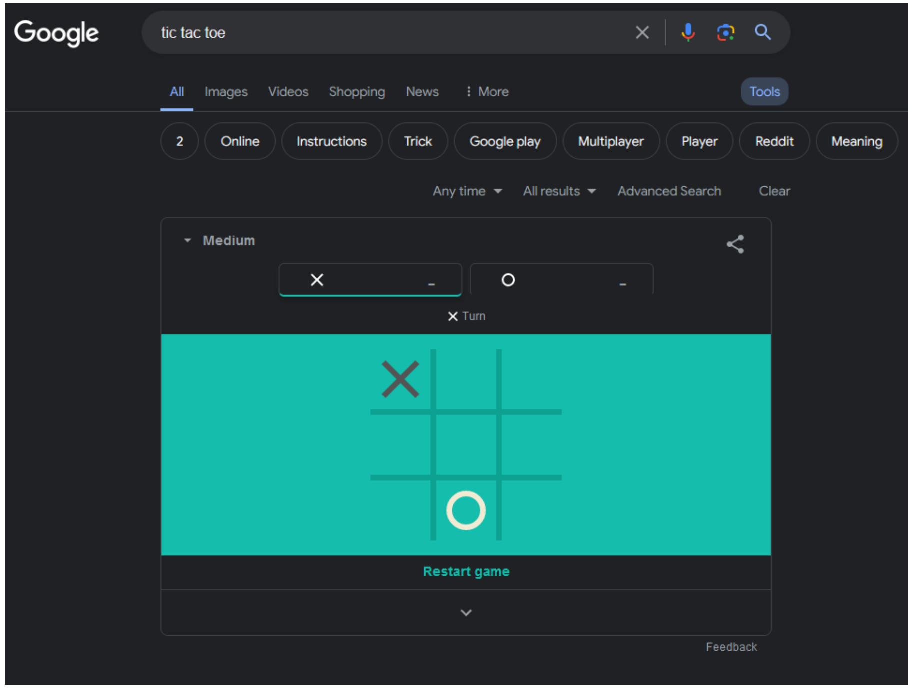
COMPSCI 389
Introduction to Machine Learning

MENACE

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

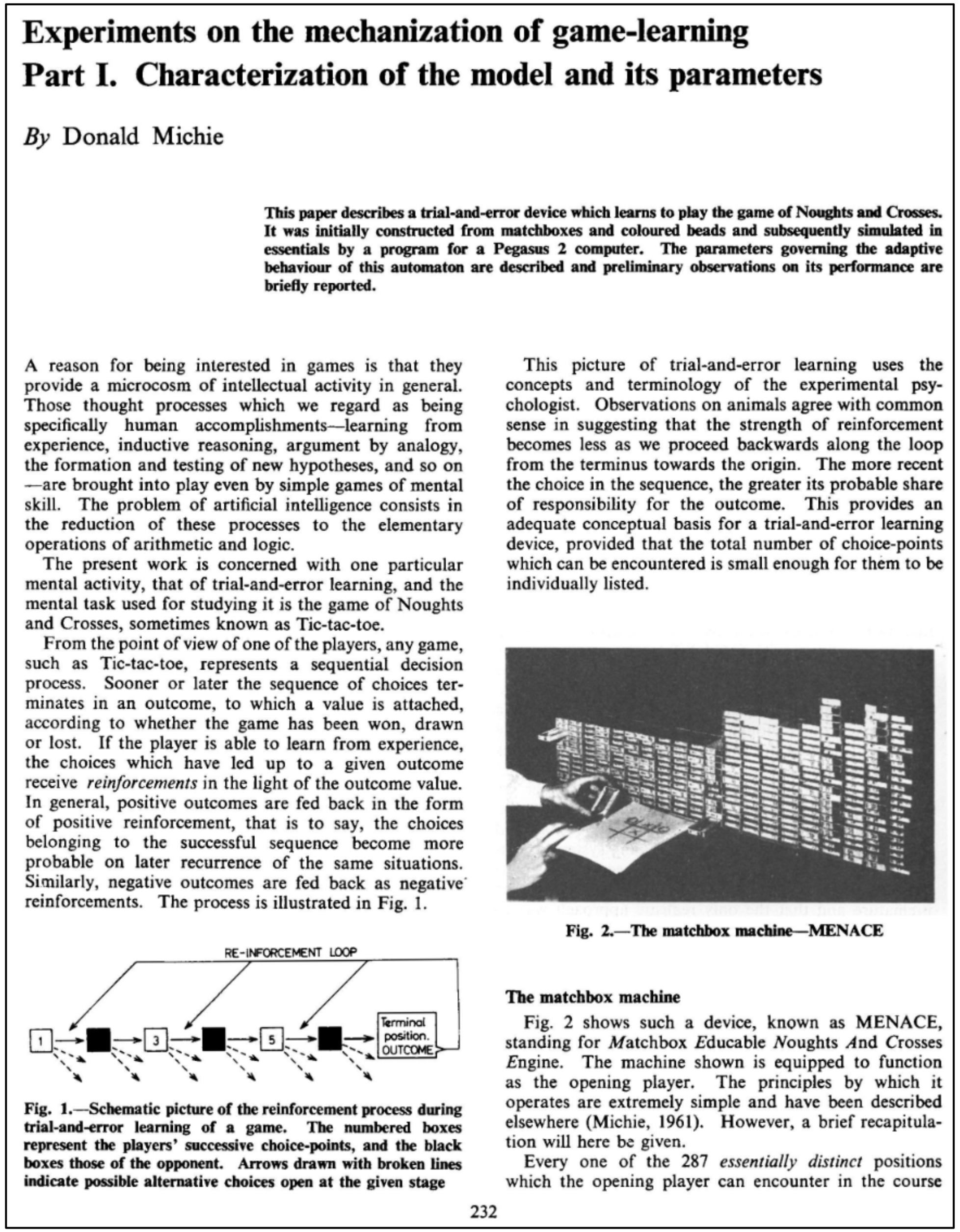
MENACE

- Machine Educable Noughts and Crosses Engine (MENACE)
- Designed by Donald Michie in 1961
- Learns to plays *noughts and crosses* (tic-tac-toe)



MENACE

- Machine Educable Noughts and Crosses Engine (MENACE)
- Designed by Donald Michie in 1961
- Learns to plays *noughts and crosses* (tic-tac-toe)
- One of the first RL algorithms!
- We will cover a *variant* of the original MENACE (details may differ)



Experiments on the mechanization of game-learning
Part I. Characterization of the model and its parameters

By Donald Michie

This paper describes a trial-and-error device which learns to play the game of Noughts and Crosses. It was initially constructed from matchboxes and coloured beads and subsequently simulated in essentials by a program for a Pegasus 2 computer. The parameters governing the adaptive behaviour of this automaton are described and preliminary observations on its performance are briefly reported.

A reason for being interested in games is that they provide a microcosm of intellectual activity in general. Those thought processes which we regard as being specifically human accomplishments—learning from experience, inductive reasoning, argument by analogy, the formation and testing of new hypotheses, and so on —are brought into play even by simple games of mental skill. The problem of artificial intelligence consists in the reduction of these processes to the elementary operations of arithmetic and logic.

The present work is concerned with one particular mental activity, that of trial-and-error learning, and the mental task used for studying it is the game of Noughts and Crosses, sometimes known as Tic-tac-toe.

From the point of view of one of the players, any game, such as Tic-tac-toe, represents a sequential decision process. Sooner or later the sequence of choices terminates in an outcome, to which a value is attached, according to whether the game has been won, drawn or lost. If the player is able to learn from experience, the choices which have led up to a given outcome receive *reinforcements* in the light of the outcome value. In general, positive outcomes are fed back in the form of positive reinforcement, that is to say, the choices belonging to the successful sequence become more probable on later recurrence of the same situations. Similarly, negative outcomes are fed back as negative reinforcements. The process is illustrated in Fig. 1.

Fig. 1.—Schematic picture of the reinforcement process during trial-and-error learning of a game. The numbered boxes represent the players' successive choice-points, and the black boxes those of the opponent. Arrows drawn with broken lines indicate possible alternative choices open at the given stage

This picture of trial-and-error learning uses the concepts and terminology of the experimental psychologist. Observations on animals agree with common sense in suggesting that the strength of reinforcement becomes less as we proceed backwards along the loop from the terminus towards the origin. The more recent the choice in the sequence, the greater its probable share of responsibility for the outcome. This provides an adequate conceptual basis for a trial-and-error learning device, provided that the total number of choice-points which can be encountered is small enough for them to be individually listed.

Fig. 2.—The matchbox machine—MENACE

The matchbox machine

Fig. 2 shows such a device, known as MENACE, standing for *M*atchbox *E*ducable *N*oughts *A*nd *C*rosses *E*ngine. The machine shown is equipped to function as the opening player. The principles by which it operates are extremely simple and have been described elsewhere (Michie, 1961). However, a brief recapitulation will here be given.

Every one of the 287 *essentially distinct* positions which the opening player can encounter in the course

232

How many game states are there?

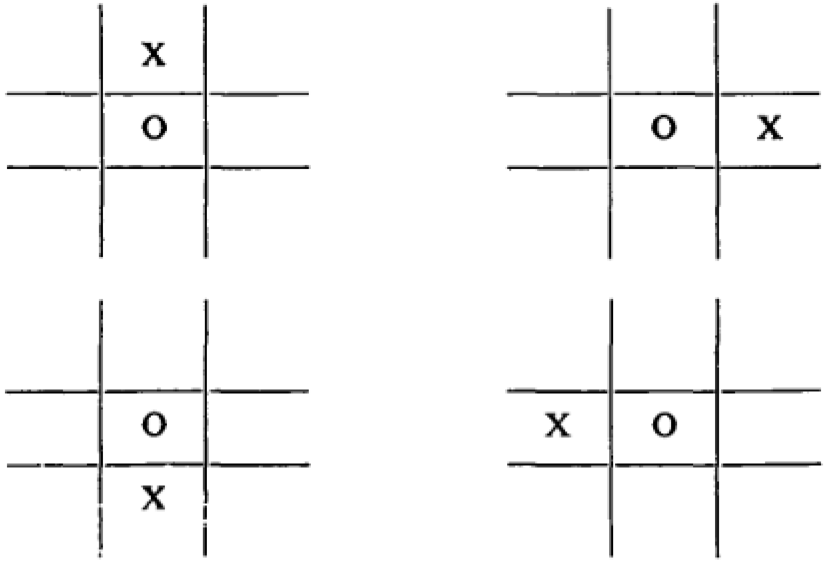


Fig. 3.—Four positions which are in reality variants of a single position

- With redundant states removed, there are only 304 game states!

Label matchboxes with every possible game state

Assign a color to each square

Table 1

The colour code used in the matchbox machine. The system of numbering the squares is that adopted for the subsequent computer simulation program

1 WHITE	2 LILAC	3 SILVER
8 BLACK	0 GOLD	4 GREEN
7 AMBER	6 RED	5 PINK

Load the matchboxes

- Load the matchboxes with several beads of each color corresponding to a legal move.

2024

2025

How to play

- When it is MENACE’s turn to make a move, find the matchbox corresponding to the current game state
- Randomly select a bead from inside
- MENACE takes the move corresponding to the color of the chosen bead
- Leave the matchbox open with the bead in front

Note: The real MENACE algorithm gave different rewards based on how many moves had been made

How MENACE learns

- If MENACE wins, for each move, return the bead and add three extra beads of the same color.
  - This makes it **more** likely for MENACE to select the chosen moves in the future.
- If MENACE loses, the beads are not returned to the boxes.
  - This makes it **less** likely for MENACE to select the chosen moves in the future.
- If it is a draw, then return the beads to the matchboxes along with one extra bead.
  - This makes it *slightly* **more** likely for MENACE to select the chosen moves in the future.

Results

- Michie played 220 games against MENACE over 16 hours.
- After 20 games MENACE could consistently draw (the result of optimal play)

How is this RL?

- The states are the possible board positions when it is MENACE's turn.
- The actions are the possible moves.
- The state transitions follow the rules of tic-tac-toe, *and include play by the human player*.
    - If the human player changes their strategy over time, then the **transition function** p of the MDP changes over time!
    - This is called a **non-stationary MDP**.
- Winning $(+3)$, losing (-1), and drawing $(+1)$ can be viewed as rewards
- The beads and matchboxes are one way of encoding a policy

Exploration Versus Exploitation

- Notice that MENACE does not always select the move that it thinks is best!
  - This is the move corresponding to the most frequent bead color in the current matchbox.
- The behavior of RL agents can be roughly classified as either **exploration** or **exploitation**.
- **Exploration:** The agent selects the action that it does *not* think is optimal in order to learn more about that action's outcome.
- **Exploitation**: The agent selects the action that it thinks is optimal to maximize the amount of reward it gets.

Exploration-Exploitation Trade-Off

- Both exploration and exploitation are necessary.
  - Without exploration, the agent will always select the same action in each state.
    - Without information about other actions, it can’t learn that they are better.
  - Without exploitation, the agent won’t maximize the amount of reward that it gets.
- RL agents balance this exploration-exploitation trade-off.

Algorithm 16.1: A simple RL algorithm inspired by MENACE.

```
for each episode do
    // Run one episode (play one game).

    // Learn from the outcome of the episode.

end
```

Remember, a policy π is parameterized by **policy parameters** θ. We write π_θ to denote the policy with parameters (weights) θ just like we wrote f_w in the supervised learning setting.

MENACE selected these actions by sampling them with probability proportional to the number of beads of each color in the matchbox for state S_t.

Algorithm 16.1: A simple RL algorithm inspired by MENACE.

for *each episode* **do**
    `// Run one episode (play one game).`
    for *each time t in the episode* **do**
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
    end
    `// Learn from the outcome of the episode.`

end

Algorithm 16.1: A simple RL algorithm inspired by MENACE.

for *each episode* **do**
    `// Run one episode (play one game).`
    for *each time* t *in the episode* **do**
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
    end
    `// Learn from the outcome of the episode.`
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is big* **then** — If MENACE won

    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is small* **then** — If MENACE lost

end

Algorithm 16.1: A simple RL algorithm inspired by MENACE.

for *each episode* **do**

  // Run one episode (play one game).

  for *each time t in the episode* **do**

    Agent observes state S_t;

    Agent selects action A_t according to the current policy, π_θ;

    Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;

  end

  // Learn from the outcome of the episode.

  if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is big* **then**

    for *each time t in the episode* **do**

      Make action A_t more likely in state S_t;

    end

  end

  if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is small* **then**

end

If MENACE won

MENACE added more beads of the color corresponding to action A_t, making it more likely in state S_t

If MENACE lost

Algorithm 16.1: A simple RL algorithm inspired by MENACE.

for *each episode* **do**
  `// Run one episode (play one game).`
  for *each time t in the episode* **do**
    Agent observes state S_t;
    Agent selects action A_t according to the current policy, π_θ;
    Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
  end
  `// Learn from the outcome of the episode.`
  if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is big* **then**
    for *each time t in the episode* **do**
      Make action A_t more likely in state S_t;
    end
  end
  if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is small* **then**
    for *each time t in the episode* **do**
      Make action A_t less likely in state S_t;
    end
  end
end

If MENACE won

If MENACE lost

MENACE removed one bead corresponding to action A_t, making it **less** likely in state S_t

How to: Make action A_t more likely in state S_t;

- Let f be a function that takes three inputs, x, y, and z, and outputs a real number
  - $f = \pi$
  - $x = S_t$
  - $y = A_t$
  - $z = \theta$
- How can we change z to increase $f(x, y, z)$?
- Recall from gradient descent lectures that the partial derivative $\frac{\partial f(x,y,z)}{\partial z}$ indicates how $f(x, y, z)$ changes a z changes.
  - If $\frac{\partial f(x,y,z)}{\partial z}$ is positive, increasing z increases $f(x, y, z)$
  - If $\frac{\partial f(x,y,z)}{\partial z}$ is negative, decreasing z increases $f(x, y, z)$
- **Solution**: Step in the direction of the partial derivative:

$$\theta \leftarrow \theta + \alpha \frac{\partial f(x, y, z)}{\partial z},$$

  - α is a hyperparameter called the **step size** or **learning rate.**
- How can we compute this derivative?
  - Reverse-mode automatic differentiation (backpropagation for neural networks)!

How to: Make action A_t ~~more~~ less likely in state S_t;

- Let f be a function that takes three inputs, x, y, and z, and outputs a real number
  - $f = \pi$
  - $x = S_t$
  - $y = A_t$
  - $z = \theta$
- How can we change z to ~~increase~~ decrease $f(x, y, z)$?
- Recall from gradient descent lectures that the partial derivative $\frac{\partial f(x,y,z)}{\partial z}$ indicates how $f(x, y, z)$ changes a z changes.
  - If $\frac{\partial f(x,y,z)}{\partial z}$ is positive, increasing z increases $f(x, y, z)$
  - If $\frac{\partial f(x,y,z)}{\partial z}$ is negative, decreasing z increases $f(x, y, z)$
- **Solution**: Step in the opposite direction of the partial derivative:

$$\theta \leftarrow \theta - \alpha \frac{\partial f(x, y, z)}{\partial z},$$

  - α is a hyperparameter called the **step size** or **learning rate.**
- How can we compute this derivative?
  - Reverse-mode automatic differentiation (backpropagation for neural networks)!

Algorithm 16.2: A simple RL algorithm inspired by MENACE, Version 2.0

for *each episode* **do**
    // Run one episode (play one game).
    for *each time t in the episode* **do**
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
    end
    // Learn from the outcome of the episode.
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is big* **then**
        for *each time t in the episode* **do**
            $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
        end
    end
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is small* **then**
        for *each time t in the episode* **do**
            $\forall i,\ \theta_i \leftarrow \theta_i - \alpha \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
        end
    end
end

Make action A_t more likely in state S_t;

Make action A_t less likely in state S_t;

- Notice that the only difference in the resulting update is the sign.
- In other problems, is a return (discounted sum of rewards) of − 1 big or small?
  - We don't know!
- Idea: **weight** the update by the return:

$$\forall i,\ \theta_i \leftarrow \theta_i + \alpha \left(\sum_{t'=0}^{\infty} \gamma^{t'} R_{t'} \right) \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}.$$

Algorithm 16.2: A simple RL algorithm inspired by MENACE, Version 2.0

for *each episode* **do**
    `// Run one episode (play one game).`
    for *each time t in the episode* **do**
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
    end
    `// Learn from the outcome of the episode.`
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is big* **then**
        for *each time t in the episode* **do**
            $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
        end
    end
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is small* **then**
        for *each time t in the episode* **do**
            $\forall i,\ \theta_i \leftarrow \theta_i - \alpha \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
        end
    end
end

Algorithm 16.2: A simple RL algorithm inspired by MENACE, Version 2.0

for *each episode* **do**
    // Run one episode (play one game).
    for *each time t in the episode* **do**
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
    end
    // Learn from the outcome of the episode.
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is big* **then**
        for *each time t in the episode* **do**
            $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
        end
    end
    if $\sum_{t=0}^{\infty} \gamma^t R_t$ *is small* **then**
        for *each time t in the episode* **do**
            $\forall i,\ \theta_i \leftarrow \theta_i - \alpha \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
        end
    end
end

Algorithm 16.3: A simple RL algorithm inspired by MENACE, Version 3.0

for *each episode* **do**
    // Run one episode (play one game).
    for *each time t in the episode* **do**
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
    end
    // Learn from the outcome of the episode.
    for *each time t in the episode* **do**
        $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \left(\sum_{t'=0}^{\infty} \gamma^{t'} R_{t'}\right) \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
    end
end

- Consider the iteration of the for-loop where $t = 5$:

$$\forall i,\ \theta_i \leftarrow \theta_i + \alpha \left(\sum_{t'=0}^{\infty} \gamma^{t'} R_{t'} \right) \frac{\partial \pi_\theta(S_5, A_5)}{\partial \theta_i}.$$

- Recall that the partial derivative indicates how to change θ_i to increase the probability of action A_5 in state S_5.
- The weight given to this direction is:
$R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \gamma^4 R_4 + \gamma^5 R_5 + \cdots$
- However, A_5 didn't influence R_1 through R_4!
- Idea: Change the weight to only consider rewards *after* A_t:

Algorithm 16.3: A simple RL algorithm inspired by MENACE, Version 3.0

for *each episode* **do**
  // Run one episode (play one game).
  for *each time t in the episode* **do**
    Agent observes state S_t;
    Agent selects action A_t according to the current policy, π_θ;
    Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
  end
  // Learn from the outcome of the episode.
  for *each time t in the episode* **do**
    $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \left(\sum_{t'=0}^{\infty} \gamma^{t'} R_{t'} \right) \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
  end
end

$$\sum_{t'=t}^{\infty} \gamma^{t'} R_{t'} = \gamma^t \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

Note: We can replace $\frac{\partial \pi_\theta(S_t,A_t)}{\partial \theta_i}$ with $\frac{\partial \ln(\pi_\theta(S_t,A_t))}{\partial \theta_i}$.
The ln() is monotonic, and so it doesn't change the meaning.

REINFORCE

Note: The actual REINFORCE algorithm sums up the changes to θ_i from the whole episode and then makes the changes. The pseudocode below changes each θ_i at time $t = 0$, and that change influences the derivative computed at subsequent times.

Algorithm 17.2: REINFORCE

for *each episode* **do**
  `// Run one episode (play one game).`
  for *each time t in the episode* **do**
    Agent observes state S_t;
    Agent selects action A_t according to the current policy, π_θ;
    Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
  end
  `// Learn from the outcome of the episode.`
  for *each time t in the episode* **do**
    $\forall i,\ \theta_i \leftarrow \theta_i + \alpha\gamma^t \left(\sum_{k=0}^{\infty} \gamma^k R_{t+k}\right) \frac{\partial \ln(\pi_\theta(S_t, A_t))}{\partial \theta_i}$;
  end
end

We inserted a ln() here

We use only the rewards after A_t to weight the partial derivative.

REINFORCE

- Proposed by Ronald Williams in the 1992 paper "Simple statistical gradient-following algorithms for connectionist reinforcement learning" *Machine Learning* 8 (1992), pp. 229–256.
- Recall the goal of finding a π^* such that:

$$\pi^* \in \arg\max_{\pi} \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right].$$

- REINFORCE is gradient ascent on the objective function:

$$J(\theta) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right],$$

where θ influences the actions A_t, and hence the distribution of rewards, R_t.

- REINFORCE remains one of the standard RL algorithms today!

End