Reinforcement Learning

a gentle introduction & industrial application

Christian Hidber
Learning learning from children
The game: demo
The game: setup

Goal: maximize sum of rewards

Game engine

Actions

Learner

Game state

Step reward

Step Reward

-1

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC
The game: positive feedback

- actions
- game engine
- learner
- game state

Step Reward
100
Step reward
The game: negative feedback

Game engine

Actions

Learner

Game state

Step reward -10
the learned stuff => policy

Policy (rules learned, how to play the game)

learner

actions

Step Reward
-1

Step reward

game state

(game engine)

actions

Step Reward
-1

Step reward

game state
policy improvement => learning

Policy

(actions)

game engine

(game state)

learner

Step reward

-1

(rules learned, how to play the game)
policy improvement => learning
Reinforcement learning

Key idea: continuously improve policy to increase total reward

Policy

(actions)

Step reward

100

Game state

RL algorithm

(game engine)

(rules learned, how to play the game)
Episode 1: play with 1st policy (random)

Policy (rules learned, how to play the game)

State

Action from Policy

Reward

Next State

-1

Step #
Episode 1: play with 1st policy (random)

State

Action from Policy

Reward

Next State

100
Episode 1: play with 1st policy (random)

State: Pac-Man and ghost

Action from Policy: Up

Reward: -50

Next State: Pac-Man and ghost

Episode Over
Episode 1: improve 1st policy for state in step 3

Policy
(rules learned, how to play the game)
Episode 1: improve 1st policy for state in step 2

Policy (rules learned, how to play the game)

State

Action from Policy

Future Reward

(sum of all rewards from current state until ‘game over’)
Episode 1: improve 1st policy for state in step 1

Policy
(rules learned, how to play the game)

State

Action from Policy

Future Reward
(sum of all rewards from current state until 'game over')

Episode Over

Step #
Episode 2: play with 2nd policy

Already learned: go left is ok

State | Action from Policy | Reward | Next State
---|---|---|---

Step #
1 2 3 4 5 6 7
Episode 2: play with 2nd policy

Already learned: go left is ok

State Action from Policy Reward Next State

100
Episode 2: play with 2nd policy

Policy
(rules learned, how to play the game)

Already learned: don’t go up

State
Action from Policy
Reward
Next State

Step #
Episode 2: play with 2nd policy

Policy (rules learned, how to play the game)
Episode 2: play with 2nd policy

Policy (rules learned, how to play the game)

State

Action from Policy

Reward

Next State

Episode Over

Episode Over
Episode 2: improve 2nd policy for state in step 5

Policy (rules learned, how to play the game)

Future Reward
(sum of all rewards from current state until 'game over')

Episode Over

-50
(=-50)

State

Action from Policy

Episode Over

1 2 3 4 5 6 7 Step #
Episode 2: improve 2nd policy for state in step 4

Policy (rules learned, how to play the game)

State

Action from Policy

Future Reward
(sum of all rewards from current state until 'game over')

Episode Over

-51 (= -1-50)

1 2 3 4 5 6 7 Step #
Episode 2: improve 2nd policy for state in step 3

FOLIE 24
REINFORCEMENT LEARNING
Episode 2: improve 2nd policy for state in step 2

Policy (rules learned, how to play the game)

Future Reward
(sum of all rewards from current state until 'game over')

Episode Over

149 (=+100+100-1-50)
Episode 2: improve 2nd policy for state in step 2

Policy (rules learned, how to play the game)

State

Action from Policy

= some running average of old and new value

149 (=+100+100-1-50)

Future Reward (sum of all rewards from current state until 'game over')
Episode 2: improve 2nd policy for state in step 1

Future Reward

\[ 148 = (-1 + 100 + 100 - 1 - 50) \]

(sum of all rewards from current state until ‘game over’)
So far.....

A policy is a map from states to action probabilities
…updated by the reinforcement learning algorithm

A policy is a map from states to action probabilities.
...updated by the reinforcement learning algorithm

a policy is a map from states to action probabilities
After many, many episodes, for each state...
Algorithm sketch

Initialize table with random action probabilities for each state

Repeat

play episode with policy given by table

Record (state_1, action_1, reward_1), (state_2, action_2, reward_2), … for episode

For each step i

    compute FutureReward_i = reward_i + reward_{i+1} + …

    update table[state_i] s.t.

    • action_i becomes for state_i more likely if FutureReward_i is “high”
    • action_i becomes for state_i less likely if FutureReward_i is “low”
Algorithm sketch

Initialize table with random action probabilities for each state

Repeat

play episode with policy given by table

Record \((state_1, action_1, reward_1), (state_2, action_2, reward_2), \ldots\) for episode

For each step \(i\)

compute \(\text{FutureReward}_i = \text{reward}_i + \text{reward}_{i+1} + \ldots\)

update \(table[state_i]\) s.t.

- \(action_i\) becomes for state \(i\) more likely if \(\text{FutureReward}_i\) is “high”
- \(action_i\) becomes for state \(i\) less likely if \(\text{FutureReward}_i\) is “low”
Algorithm sketch

Initialize table with random action probabilities for each state
Repeat
    play episode with policy given by table
    Record \((\text{state}_1, \text{action}_1, \text{reward}_1), (\text{state}_2, \text{action}_2, \text{reward}_2), \ldots\) for episode
    For each step \(i\)
        compute \(\text{FutureReward}_i = \text{reward}_i + \text{reward}_{i+1} + \ldots\)
        update \(\text{table}[\text{state}_i] \) s.t.
        \begin{itemize}
        \item \(\text{action}_i\) becomes for state \(i\) more likely if FutureReward \(i\) is “high”
        \item \(\text{action}_i\) becomes for state \(i\) less likely if FutureReward \(i\) is “low”
        \end{itemize}
The game: demo
The bad news: nice idea, but...
The bad news: nice idea, but…

too many states… too many actions

• Too much memory needed
• Too much time
The solution

Idea:
Replace lookup table with a neural network that approximates the action probabilities contained in the table

Instead of

Table[state] = action probabilities

Do

NeuralNet(state) ~ action probabilities

Change to “play episode with policy given by NeuralNet”

Change to “update weights of NeuralNet”

Initialize table with action probabilities for each state
Repeat
  play episode with policy given by table
  Record (state, action, reward), (state, action, reward), … for episode
  For each step i
    compute FutureReward = reward + reward + …
    update table[state] =
      \[ \begin{array}{ll}
      \text{action, becomes for state, more likely if FutureReward is “high”} \\
      \text{action, becomes for state, less likely if FutureReward is “low”}
\end{array} \]
Neural nets to the rescue

Idea:
Replace lookup table with a neural network that approximates the action probabilities contained in the table

Instead of

Table[state] = action probabilities

Do

NeuralNet(state) \sim action probabilities

Encode state as vector

Apply neural network with “the right” weights

Use softmax
Policy Gradient Algorithm sketch

Initialize neuralNet with random weights W

Repeat

play episode(s) with policy given by weights W

Record (state\_1, action\_1, reward\_1), (state\_2, action\_2, reward\_2), … for episode(s)

For each step i

compute FutureReward\_i = reward\_i + reward\_i+1 + …

Update weights W

W = W + ????

Encode state as vector

Use softmax
Policy Gradient Algorithm sketch

Initialize neuralNet with random weights W
Repeat
  play episode(s) with policy given by weights W
  Record (state₁, action₁, reward₁), (state₂, action₂, reward₂), … for episode(s)
  For each step i
    compute FutureRewardᵢ = rewardᵢ + rewardᵢ₊₁ + …
  Update weights W
    W = W + ????

Encode state as vector

Weights W

Use softmax
Policy Gradient Algorithm sketch

Initialize neuralNet with random weights $W$

Repeat

play episode(s) with policy given by weights $W$

Record $(\text{state}_1, \text{action}_1, \text{reward}_1), (\text{state}_2, \text{action}_2, \text{reward}_2), \ldots$ for episode(s)

For each step $i$

compute $\text{FutureReward}_i = \text{reward}_i + \text{reward}_{i+1} + \ldots$

Update weights $W$

$$W = W + \text{???}$$

Increases $\text{out}_i$

Encode state as vector

Use softmax

Weights $W$
Policy Gradient Algorithm sketch

Initialize neuralNet with random weights W

Repeat
  play episode(s) with policy given by weights W
  Record (state₁, action₁, reward₁), (state₂, action₂, reward₂), … for episode(s)
  For each step i
    compute FutureRewardᵢ = rewardᵢ + rewardᵢ₊₁ + …
    Update weights W
    \[ W = W + \alpha \times \text{FutureReward}_i \times \frac{\text{Gradient}_W(\text{neuralNet}_W(\text{state}_i, \text{action}_i))}{\text{Grad}_W(\text{neuralNet}_W(\text{state}_i, \text{action}_i))} \]
    Learning rate Increases outᵢ

Encode state as vector

Use softmax

Weights W
What for? The real world

no feasible, deterministic algorithm
What for?

- Traditional Heuristics
- Classic Machine Learning
- Reinforcement Learning

Automatic solution found in 93.4%
The challenges

- manage the water level on the roof
- control & steer the water flow
- find the right dimensions
- save & reliable
Finding the right dimensions
Finding the „right“ dimensions: demo
What if…

• Collapsing pipes
• Collapsing roofs
• Clogged pipes
• Façade damages
Turning the problem into a game
### Designing the Action-Space

<table>
<thead>
<tr>
<th>Snake game</th>
<th>Roof drainage systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Snake game actions" /></td>
<td><img src="image" alt="Roof drainage actions" /></td>
</tr>
</tbody>
</table>

- What actions would a human expert like to have?
- Are these actions sufficient?
- Would more / other actions be helpful?
- Can we drop any actions?
Designing the State-Space

Snake game

- What does a human expert look at?
- Can you switch the experts between 2 steps?
- Full state vs partial state
- Designing Features

Roof drainage systems
Designing the Reward Function

### Snake game

<table>
<thead>
<tr>
<th>Step Reward</th>
<th>Fruit</th>
<th>Death</th>
<th>Success</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>100</strong></td>
<td>100</td>
<td>-50</td>
<td>1000</td>
<td>-1</td>
</tr>
</tbody>
</table>

- How would you rate the result of an expert?
- As simple as possible
- Positive feedback during the game
- Beware of “surprising policies”
- Game over if TotalReward too low

### Roof drainage systems

- Change Error Count: +/- 1 per Error
- Success: 100
- Step: -0.01
Turning the problem into a game

- Policy (rules learned, how to play the game)
- Game engine
- RL algorithm
- Game state
- Step reward
- Actions
Finding the dimensions with reinforcement learning: demo
Hydraulics Calculation Pipeline

- **Traditional Heuristics**: Automatic solution found in 93.4%
- **Classic Machine Learning**: Find a solution in 70.7% of the remaining 6.6%
- **Reinforcement Learning**: Automatic solution found in 98.1%
Wrap Up

• Turning the problem into a game
• Continuous policy improvement
• No training dataset
• Complements supervised learning
Thank you!

Christin.Hidber@bSquare.ch
W +41 44 260 54 00
M +41 76 558 41 48
https://www.linkedin.com/in/christian-hidber/

About Geberit

The globally operating Geberit Group is a European leader in the field of sanitary products. Geberit operates with a strong local presence in most European countries, providing unique added value when it comes to sanitary technology and bathroom ceramics.

The production network encompasses 30 production facilities, of which 6 are located overseas. The Group is headquartered in Rapperswil-Jona, Switzerland. With around 12,000 employees in around 50 countries, Geberit generated net sales of CHF 2.9 billion in 2017. The Geberit shares are listed on the SIX Swiss Exchange and have been included in the SMI (Swiss Market Index) since 2012.