Reinforcement Learning

a gentle introduction & industrial application

Christian Hidber

Questions: slido.com #bigdata2018
Learning learning from children
The game: demo
The game: setup

Goal: maximize sum of rewards

Game engine

Actions

Learner

Game state

Step reward -1
The game: positive feedback

game engine

actions

learner

game state

Step Reward

100

Step reward
The game: negative feedback

actions

game engine

learner

Step reward

-10

Step Reward

game state
the learned stuff => policy

Policy

(game engine)

learner

Step Reward
-1

Step reward

actions

(game state)

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

game engine

actions

Policy

(rules learned, how to play the game)

learner

Step reward

-1

game state
policy improvement => learning

FOLIE 9
REINFORCEMENT LEARNING

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC
Reinforcement learning

Key idea:
continuously improve policy to increase total reward

Policy
(rules learned, how to play the game)

RL algorithm

Step Reward
100
Step reward

game state

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

State:  Image
Action from Policy:  Image
Reward:  -1
Next State:  Image

Step #1 1
Step #2 2  3  4  5  6  7
Episode 1: play with 1st policy (random)

State

Action from Policy

Reward

Next State

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

1. State
2. Action from Policy
3. Reward
4. Next State

Episode Over

Step #
Episode 1: improve 1st policy for state in step 3

State

Action from Policy

Episode Over

Episode Over

-50
Episode 1: improve 1st policy for state in step 2

REINFORCEMENT LEARNING

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

1 2 3 4 5 6 7

Step #
Episode 1: improve 1st policy for state in step 1

Policy (rules learned, how to play the game)

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

49 (= -1 +100 -50)

State

Action from Policy

Episode Over

1 2 3 4 5 6 7

Step #
Episode 2: play with 2nd policy

Policy (rules learned, how to play the game)

Already learned: go left is ok

State

Action from Policy

Reward

Next State

Step #
Episode 2: play with 2nd policy

---

1. State
2. Action from Policy
3. Reward
4. Next State

---

Already learned: go left is ok

---

Step #
Episode 2: play with 2nd policy

State

Action from Policy

Reward

Next State

Already learned: don’t go up

Step #
Episode 2: play with 2nd policy

Policy (rules learned, how to play the game)

State

Action from Policy

Reward

Next State

Step #
Episode 2: play with 2nd policy

Policy (rules learned, how to play the game)

Episode Over

Step #
Episode 2: improve 2nd policy for state in step 5

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: improve 2nd policy for state in step 4

Policy (rules learned, how to play the game)

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

-51 (= -1 - 50)

Episode Over
Episode 2: improve 2nd policy for state in step 3

Policy (rules learned, how to play the game)

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

49
(+100-1-50)

State

Action from Policy

Episode Over

1 2 3 4 5 6 7

Step #
Episode 2: improve 2nd policy for state in step 2

Policy (rules learned, how to play the game)

State

Action from Policy

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 2

Policy (rules learned, how to play the game)

State

Action from Policy

= some running average of old and new value

Future Reward

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

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

Episode Over
Episode 2 : improve 2nd policy for state in step 1

Policy
(rules learned, how to play the game)

148 (= -1 + 100 + 100 - 1 - 50)
Future Reward
(sum of all rewards from current state until ‘game over’)

State
Action from Policy

Episode Over

1 2 3 4 5 6 7
Step #
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 \((\text{state}_1, \text{action}_1, \text{reward}_1), (\text{state}_2, \text{action}_2, \text{reward}_2), \ldots\) 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.
      \[
      \begin{align*}
      & \text{action}_i \text{ becomes for state}_i \text{ more likely if FutureReward}_i \text{ is “high”} \\
      & \text{action}_i \text{ becomes for state}_i \text{ less likely if FutureReward}_i \text{ is “low”}
      \end{align*}
      \]

Policy (rules learned, how to play the game)
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 table[state,] s.t.
      - \(\text{action}_i\) becomes for state \(i\) more likely if \(\text{FutureReward}_i\) is “high”
      - \(\text{action}_i\) 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 `(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”
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

Initialize \( \text{table} \) with action probabilities for each state
Repeat
- play episode with policy given by \( \text{table} \)
- 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 \( \text{FutureReward} = \text{reward}_i + \gamma \cdot \text{FutureReward} \)
    - update \( \text{table[state]} \) as
      - action becomes more likely if \( \text{FutureReward} \) is "high"
      - action becomes less likely if \( \text{FutureReward} \) is "low"

Change to “play episode with policy given by NeuralNet”
Change to “update weights of NeuralNet”
Neural nets to the rescue

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

Instead of

\[
\text{Table}[\text{state}] = \text{action probabilities}
\]

Do

\[
\text{NeuralNet}( \text{state} ) \sim \text{action probabilities}
\]

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

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

\[ W = W + \text{????} \]

Encode state as vector

Use softmax
Policy Gradient Algorithm sketch

Initialize \textit{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 \( W \)

\[
W = W + \text{???} \quad \text{Increases out}_i
\]
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 $W$

  $W = W + \text{Learning rate} * \text{FutureReward}_i * \text{Grad}_{W} (\text{neuralNet}_W(\text{state}_i, \text{action}_i))$

  Increases $\text{out}_i$

Encode state as vector

Use softmax

Weights $W$
Policy Gradient

\[ W = \arg \max_W E_{\tau \sim p_W}[R(\tau)] \]

\[ \tau = (s_1, a_1, r_1), (s_2, a_2, r_2), ... \]

\[ R(\tau) = \sum_i r_i \]

\[ p_W = \text{episode probability given the policy defined by the NeuralNet with weights } W \]

\[ W_{k+1} = W_k + \alpha \cdot \nabla_W f(W_k) \]
Policy Gradient

\[ W = \operatorname{arg\, max}_W E_{\tau \sim p_W}[R(\tau)] \]

expected total reward playing with \( W \) = \( f(W) \)

\[ W_{k+1} = W_k + \alpha \cdot \nabla_W E_{\tau \sim p_W}[R(\tau)] \]

\( \nabla_W E_{\tau \sim p_W}[R(\tau)] = \nabla_W \int_{\tau} p_W(\tau) \cdot R(\tau) \]

\[
= \int_{\tau} p_W(\tau) \cdot \frac{\nabla_W p_W(\tau)}{p_W(\tau)} \cdot R(\tau) = \int_{\tau} p_W(\tau) \cdot \nabla_W \log p_W(\tau) \cdot R(\tau)
\]

\[ = E_{\tau \sim p_W}[\nabla_W \log p_W(\tau) \cdot R(\tau)] \]

good news

\[
\tau = (s_1, a_1, r_1), (s_2, a_2, r_2), \ldots
\]

\[ R(\tau) = \sum_i r_i \]

\( p_W = \) episode probability given the policy defined by the NeuralNet with weights \( W \)

\[ \frac{\delta f(x)}{f(x)} = \nabla_x \log f(x) \]
Policy Gradient

\[ W = \text{arg} \max_W E_{\tau \sim p_W} [R(\tau)] \]

this is trouble  "average" total reward playing with W

\[ W_{k+1} = W_k + \alpha \cdot \nabla_W E_{\tau \sim p_W} [R(\tau)] \]

this is the new trouble

\[ W_{k+1} = W_k + \alpha \cdot E_{\tau \sim p_W} [\nabla_W \log p_W(\tau) \cdot R(\tau)] \]

\[ \sum_i \nabla_W \text{NeuralNet}_W(s_i, a_i) \cdot \text{FutureReward}_i \]

\[ W = W + \alpha \cdot \text{Gradient}_W (\text{neuralNet}_W(s_i, a_i)) \cdot \text{FutureReward}_i \]

\[ \tau = (s_1, a_1, r_1), (s_2, a_2, r_2), ... \]

\[ R(\tau) = \sum_i r_i \]

\[ p_W = \text{episode probability given the policy defined by the NeuralNet with weights } W \]
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

actions

Policy
(rules learned, how to play the game)

RL algorithm

Step Reward
100
Step reward

game state

Turning the problem into a game

REINFORCEMENT LEARNING
Designing the Action-Space

<table>
<thead>
<tr>
<th>Snake game</th>
<th>Roof drainage systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Snake game arrows" /></td>
<td><img src="image2" alt="Roof drainage icons" /></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

Roof drainage systems

• What does a human expert look at?
• Can you switch the experts between 2 steps?
• Full state vs partial state
• Designing Features
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>100</td>
<td>100</td>
<td>-50</td>
<td>1000</td>
<td>-1</td>
</tr>
</tbody>
</table>

### Roof drainage systems

- Change Error Count: +/- 1 per Error
- Success: 100
- Step: -0.01

- 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
Turning the problem into a game

Policy (rules learned, how to play the game)

Game engine

Step reward

Game state

Actions

RL algorithm
Finding the dimensions with reinforcement learning: demo
Hydraulics Calculation Pipeline

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

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

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.
Resources

- Sutton & Barto: Reinforcement Learning, an introduction, 2nd edition, 2018: https://drive.google.com/file/d/1opPSz5AZ_kVa1uWOdOiveNiBFiEOHjkG/view
- http://karpathy.github.io/2016/05/31/rl/
- https://openai.com/