Millennials.ai

Millennials.ai

Millennials.ai is a research based Artificial Intelligence (A.I.) company with a focus on middle large companies already using their data to efficiently execute their business proposition. We integrate custom made algorithms and machine learning into existing business processes to optimize the value of your business data.

About me

- MSc Artificial Intelligence student at the UvA
- AI developer at Millennials.ai
- Currently doing my thesis about *Compositionality in Reinforcement Learning* at the Institute for Logic Language and Computation (ILLC)

Training: Reinforcement Learning

17-05-2019

Content

- Introduction
  - (Machine) Learning
  - Background info
- Theory
  - MDP formalization
  - Policy evaluation
  - Policy iteration
  - Value iteration
- Hands on
  - Your turn!
- Practical applications & Takeaways
  - Showcasing the cool stuff

Introduction

Artificial Intelligence

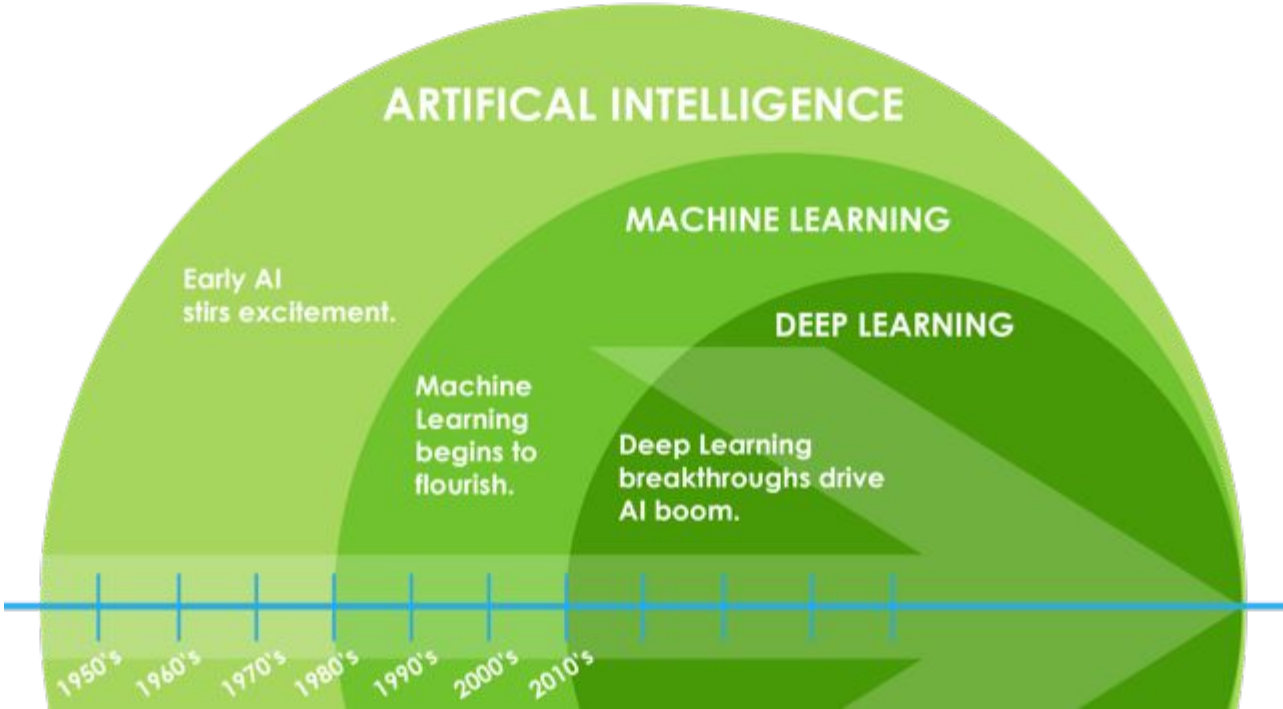
- Definition of artificial intelligence?

  “The study of agents that receive percepts from the environment and perform actions” - *Russel & Norvig*

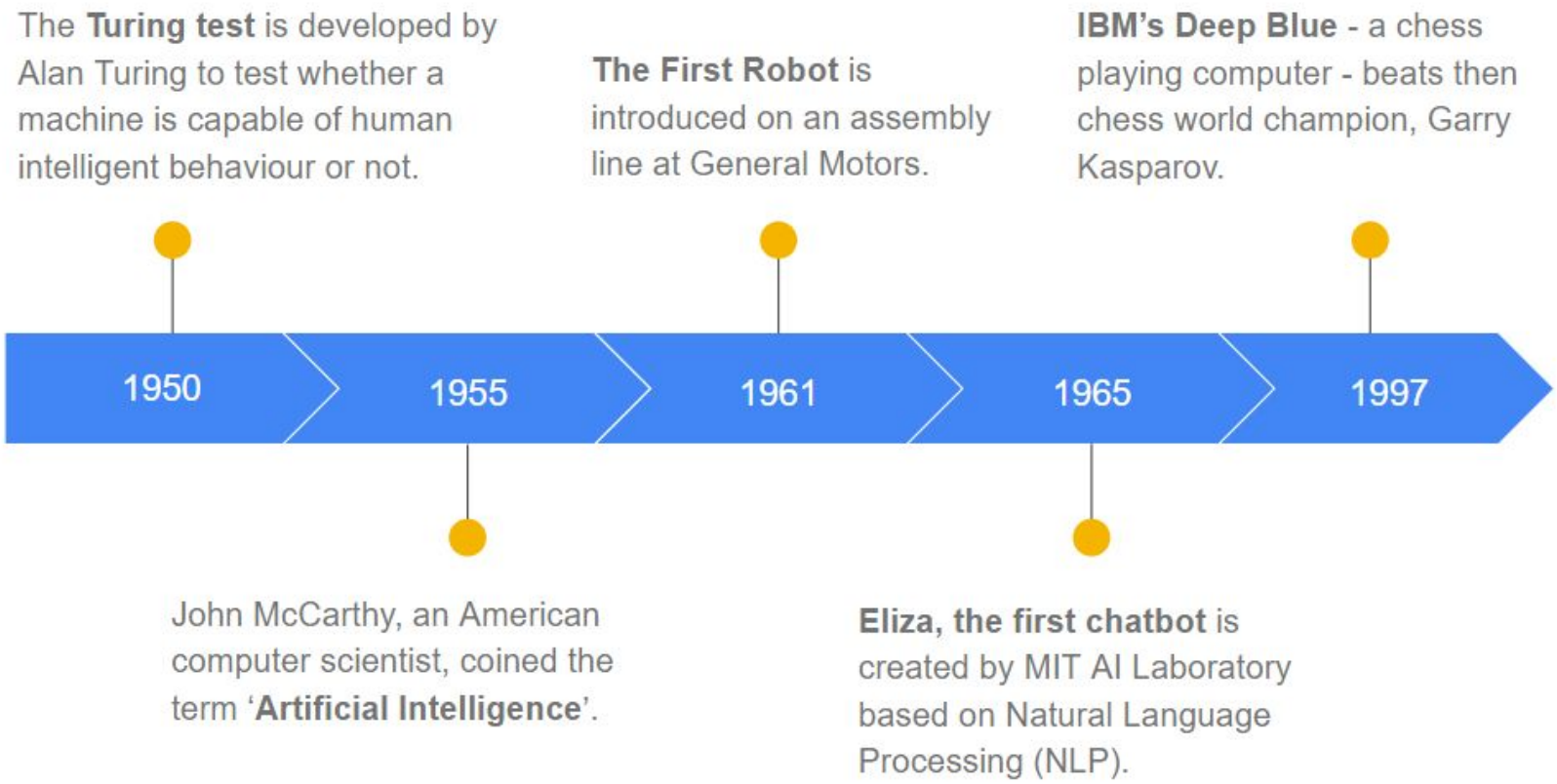
- Art of learning mappings
  - Function from input to labels -> y = f(x)
- What do we consider learning? Semantic debate!

Terminology

- First things first, lots of people seem to get this wrong!



Brief history of AI



More recent history

- 1998: Gradient-based learning with backpropagation, Yann LeCunn
- 2009: Launch of ImageNet Dataset & Challenge
  - Millions of annotated images of cats, dogs, cars, etc.
- 2011: AlexNet wins ImageNet Challenge by a large margin
  - Eight layer deep network
  - Trained using GPUs
  - Use of *relu* activations
- 2014: Facebook publishes DeepFace, a 97% accuracy face detector
- 2015: Deep Q-Learning achieves human-level control on 49 Atari games

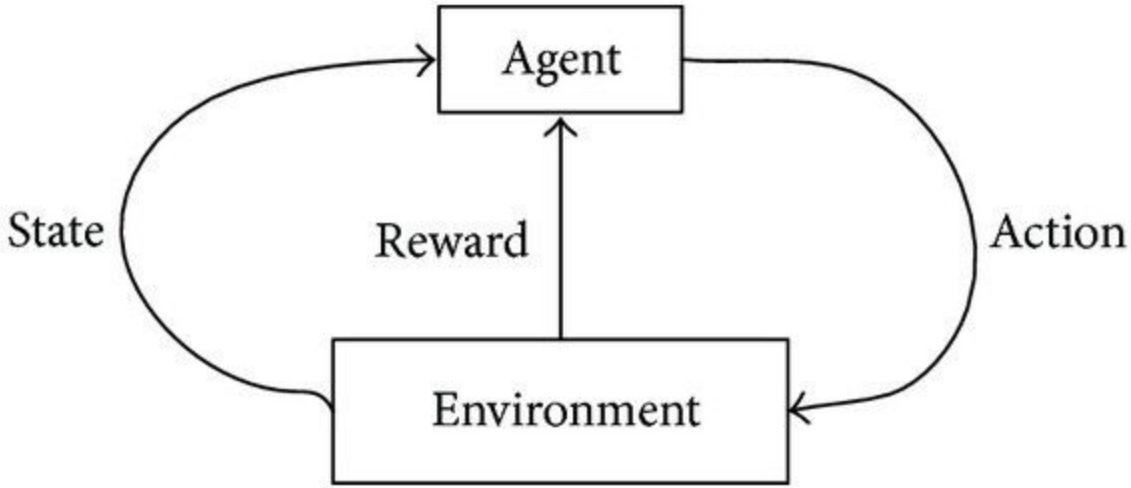
Types of learning

- Supervised: Learning to predict
- Unsupervised: Learning to structure
- Semi-supervised (*self training*): Learning to induce
- Reinforcement: Learning to do!

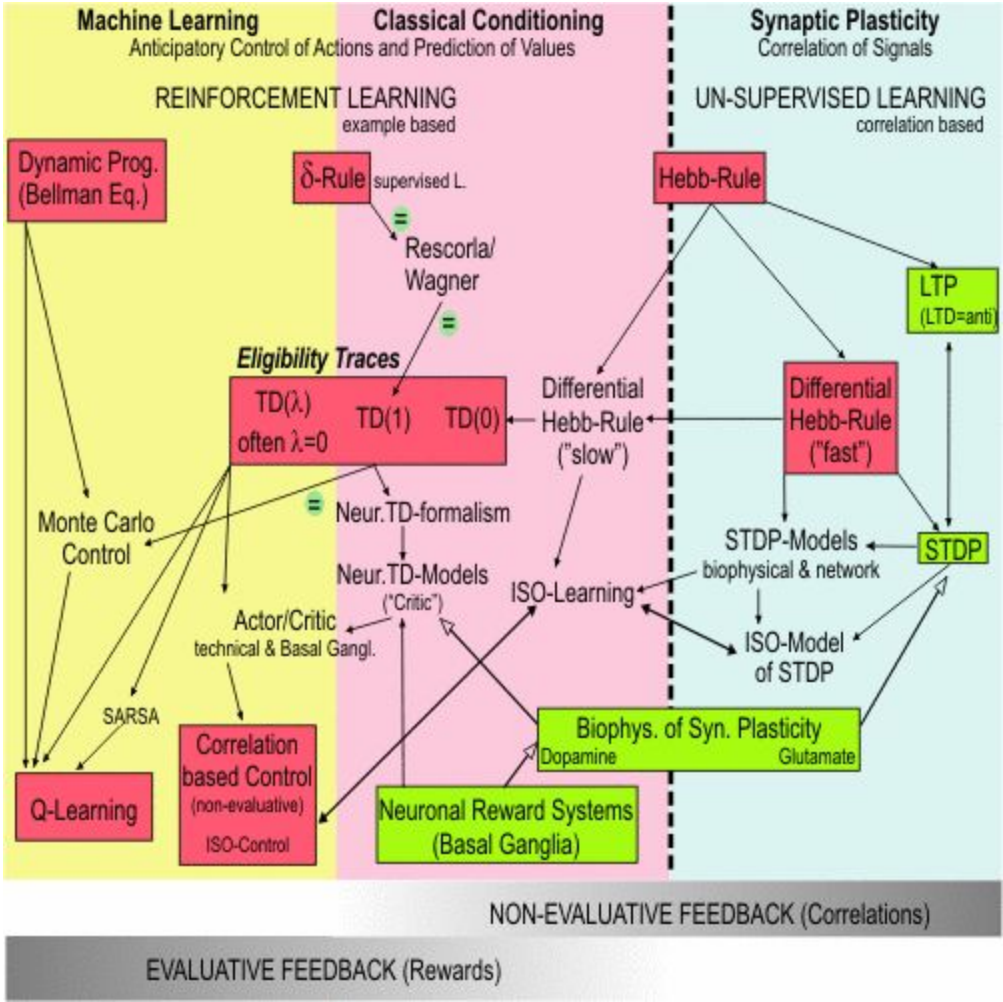
We'll see the difference between RL and supervised learning later

Reinforcement Learning in a nutshell

- Agent should learn to “maximize future reward”
- Talk about states, actions, reward
- Learning from negative or positive feedback
- In the end: no direct human dataset -> better than human performance?



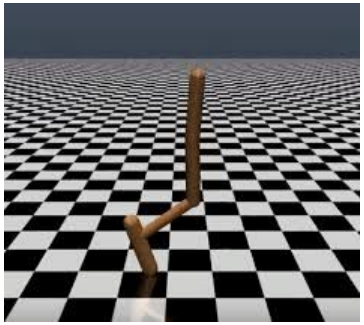
RL is difficult to position as a single research area



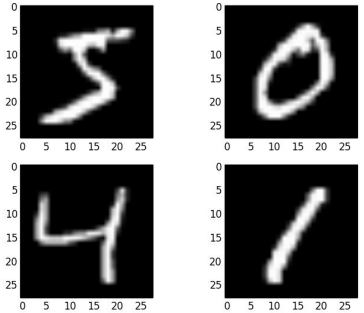
Reinforcement Learning vs other types

- Sequential decision making
- Actions influence your future *observations*
- In supervised learning, actions (predictions) only influence your future actions
  - This is learning!
  - No i.i.d. assumptions
- Individual states are hard (impossible?) to label

Reinforcement Learning vs Supervised Learning



- Learning to do
- Become better than humans
- Data depends on current strategy
- Data is sequential



- Learning to predict
- Dataset annotated by humans
- Data doesn’t depend on learned behavior*
- Data is assumed to be i.i.d.

Why is RL hard?

- Credit assignment
  - If reward is good -> What action in the sequence was good? The last? The first?
  - If reward is bad -> there might exist even worse actions than the ones we picked
- Data is non-iid and depends on the actions chosen
  - Get stuck in “bad” states
  - Now, we learn to deal well with bad states but cannot reach good ones

Some more definitions

Markov Decision Processes (MDPs): a model for sequential decision processes

(discounted) return: cumulative reward, discounted when we have a continuous task.

Trajectories & Episode: sequence of states, actions and rewards. If we have natural termination states, we split trajectories into episodes

Policy

Value function

Learning methods in RL

- Tabular
- Model-based
- Model free
- Q-learning
- DQN (we'll see some examples later)
- Policy gradient (enabled Neural networks)
- More sophisticated methods

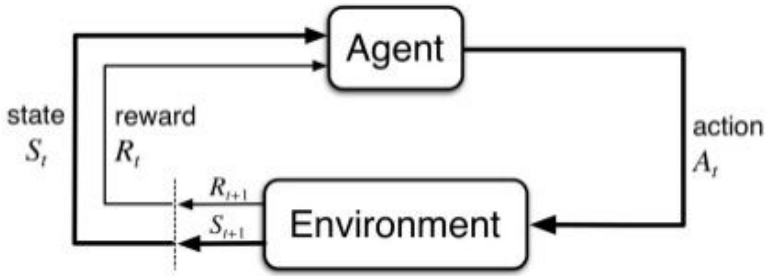
Learning with gradient updates

- Basis for neural networks, generally applies to all NN architectures
- However, hard to find right set of *hyperparameters*
- Even more so in RL!
- We will not go into detail today about these methods, but we'll see some applications
- Today: tabular methods, this means we're keeping track of all values in a big table, and allows us to use dynamic programming

RL Theory

MDP formalization

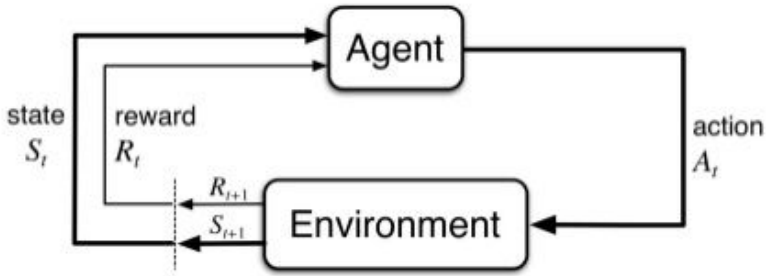
- Since RL is based on sequences of actions, observations and rewards, we call this a Process
- Markov Decision Processes are a formulation of sequential decision making
- Actions impact not only immediate reward, but next states, and future rewards as well
- Group sequences in episodes
- At the end, observe **reward**



$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

MDP formalization

- Since RL is based on sequences of actions, observations and rewards, we call this a Process
- Markov Decision Processes are a formulation of sequential decision making
- Actions impact not only immediate reward, but next states, and future rewards as well
- Group sequences in episodes
- At the end, observe **reward**



$$S_0, A_0, \quad , S_1, A_1, \quad S_2, A_2, R_3, \ldots$$

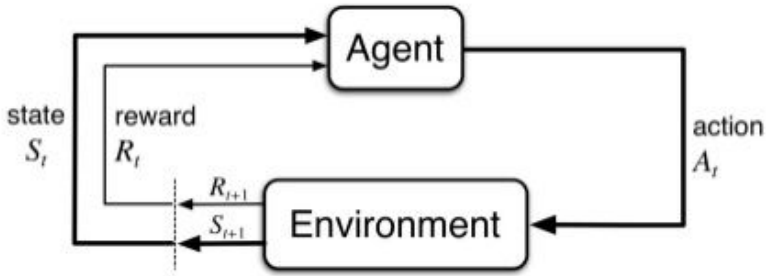
MDP formalization

- **MDP = (S, A, T, R, γ)**
- **S**: set of states
- **A**: set of actions
- **T**: Transition probabilities
  - From state s, we take action a, what is the probability we end up in state s’
- **R**: rewards
  - Numerical value to reward the agent upon completion of an objective
- **γ**: discount factor
  - Reduce reward the longer it takes to achieve
  - Solves the problem of cyclic action set

MDP formalization

Assumptions:

- Next state only depends on current state (markov assumption)
- Reward is only dependent on state, action and next state
- Discrete time steps
- Environment is fully observable
  - No hidden information
- Environment is stationary
  - Same action from same state will always have same effect



$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

MDP?

Are the following processes (vanilla) MDPs?

- Chess
- Autonomous driving
- Robot in a maze (camera on robot)
- Robot in a maze (camera above robot)

Examples that are not, can often be cast into MDPs, or some additional definition can be made to accomodate (POMDP, Continuous MDPs)

Big picture: how to learn policies

- Learn value functions V(s) or action-value functions Q(s,a)
  - How good is the current state s?
  - How good is the action a in state s?
- Use these functions to improve on our policy π(s)
  - Dictates what action we pick in state s
- Employ dynamic programming:

simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner

- Introduced by Bellman in the 1950s

Big picture: how to learn policies

- A good policy π(s) should get high expected returns
- When do we prefer one policy π over another policy π'?

$$\textbf{When } V_{\pi}(s) > V_{\pi'}(s)$$

- Similarly, we can do the same for state-action pairs

$$\textbf{When } Q_{\pi}(s,a) > Q_{\pi'}(s,a)$$

- We can use these Q values for finding the best action in state s:

$$a^{*} = \max_{a} Q_{\pi}(s,a)$$

Policy evaluation

How do we come up with $\mathbf{V_\pi(s)}$? Iterative solution, with MDP dynamics as given:

$$
\begin{aligned}
&\text{Loop:} \\
&\quad \Delta \leftarrow 0 \\
&\quad \text{Loop for each } s \in \mathcal{S}: \\
&\qquad v \leftarrow V(s) \\
&\qquad V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right] \\
&\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|) \\
&\text{until } \Delta < \theta
\end{aligned}
$$

Policy improvement

Exploit $\mathbf{V_\pi(s)}$ to come up with the best actions in all states.

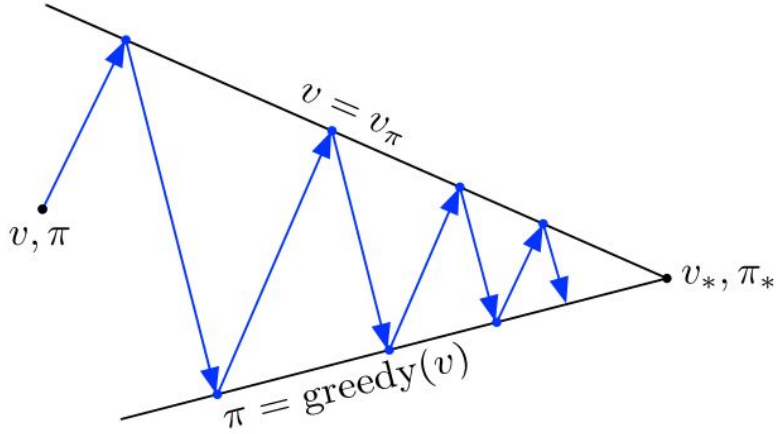
- Pick the action that looks best in the short term, according to $\mathbf{V_\pi(s)}$
- Greedy action selection!

$$
\begin{aligned}
&\textit{policy-stable} \leftarrow \textit{true} \\
&\text{For each } s \in \mathcal{S}: \\
&\quad \textit{old-action} \leftarrow \pi(s) \\
&\quad \pi(s) \leftarrow \arg\max_a \textstyle\sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big] \\
&\quad \text{If } \textit{old-action} \neq \pi(s), \text{ then } \textit{policy-stable} \leftarrow \textit{false} \\
&\text{If } \textit{policy-stable}, \text{ then stop and return } V \approx v_* \text{ and } \pi \approx \pi_*; \text{ else go to 2}
\end{aligned}
$$

Policy iteration

Consists of two iteration processes

- Policy evaluation:
  - Make the value function consistent with the current policy
- Policy improvement
  - Making the policy greedy with respect to the current value function



$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

Value iteration

- Drawback of policy iteration: evaluate policy at every iteration is computationally expensive
- Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement
- Algorithm on next slide

Value iteration

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Hands on

Hands on: your turn

Implement policy iteration & value iteration yourself on a toy problem

Don’t hesitate to ask questions or consult the documentation

Good luck!

https://git.io/fjlph

Practical Applications

Examples we'll go over

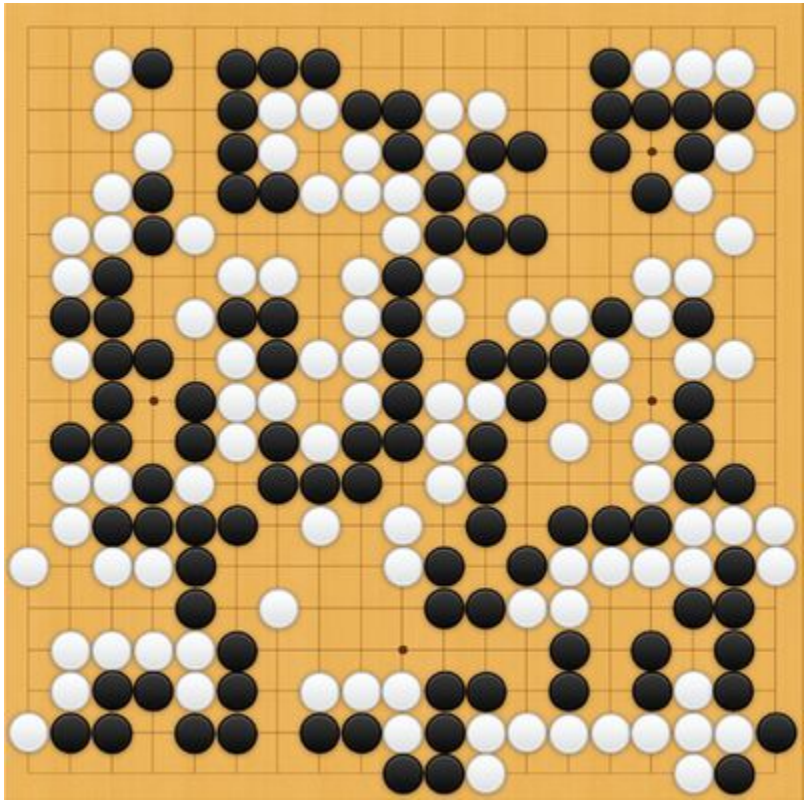
- AlphaGo
  - The DeepBlue of our age, outperforming the human world champion
- AlphaGo Zero
  - Purely RL based Go-playing AI, outperforming the original in merely days of training
- RTS games (Starcraft 2 / Dota 2)
  - Moving into massively scalable RL
- Robotics
  - The cool stuff
- Traffic Light Control
  - https://pure.uva.nl/ws/files/10793554/vanderpol_oliehoek_nipsmalic2016.pdf
- Display Advertising
  - https://arxiv.org/pdf/1802.09756.pdf

Chess?

- Chess is a combinatory explosive game
- Hard to search and plan ahead on “good” actions, since there are many states
- 1996: IBM develops Deep Blue to beat the World Champion
- However, it was not **RL**
- Combination of heuristics and brute force using massively parallel computational power
- “Think ahead” 6-8 steps.

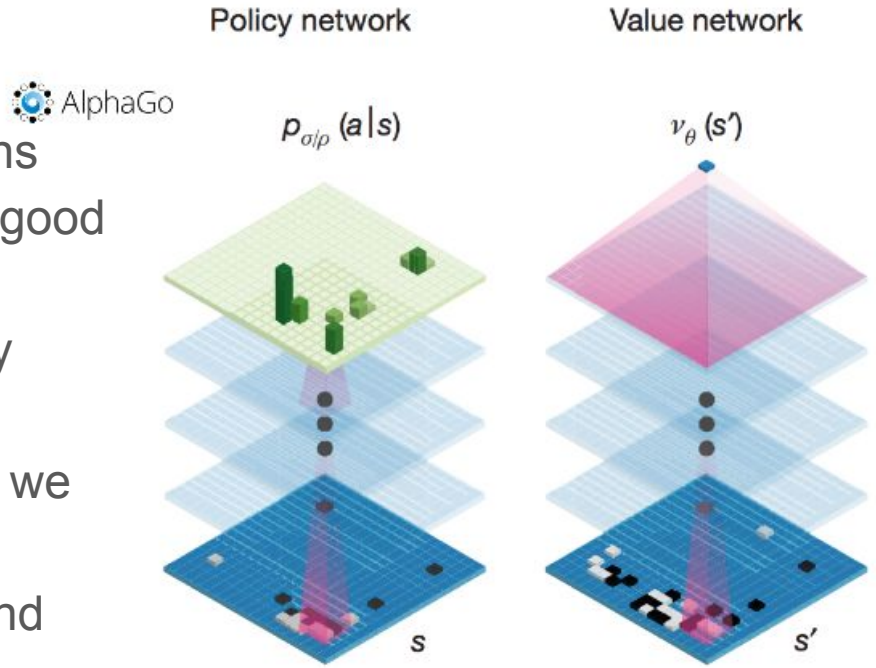
AlphaGo

- Chinese game “Go”, even computationally more difficult to enumerate all states due to large *branching factor*
- Board is 19x19, turn based between two players placing white and black stones
- Goal is to capture an area larger than the opponent.
- Monte Carlo Tree Search had already shown good performance

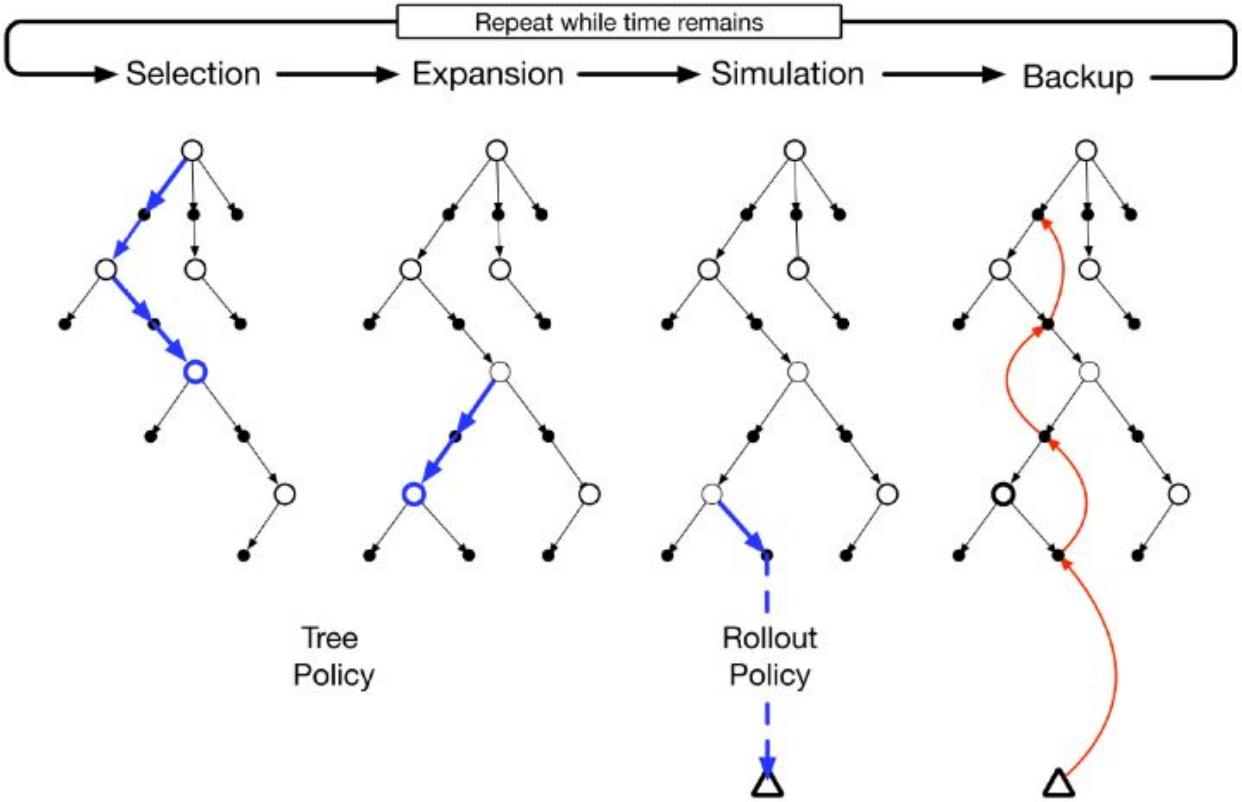


AlphaGo

- Difficult to evaluate “good” board positions
- MCTS executes a simple idea in finding good actions: just try (rollout) games from the current position according to some policy (next slide)
- When the games end, we know whether we win or lose
- Combine MCTS with Neural Networks and win everything!

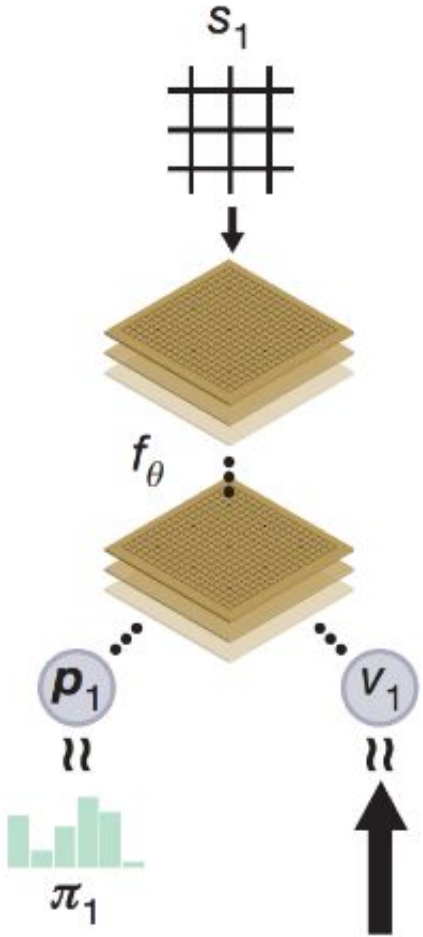


AlphaGo


Repeat while time remains
Selection
Expansion
Simulation
Backup
Tree Policy
Rollout Policy

AlphaGo Zero

- AlphaGo took months to train
- Initialized the neural networks by supervised learning
- Data gathered by humans, bad!
- How to outperform humans? Stop relying on them
- AlphaGo Zero -> purely trained through self-play
- Squash two networks from AlphaGo into one:
  Policy and value network now share the same network
- Intuition: Guided tree search by network.
- Result? Trainable agent in 3(!) days,



AlphaStar & OpenAI Five

- Once classical games are beaten, where do you go?
- Video games!
- More difficult setting because of multiple factors:
  - Partial observation
  - Multi-agent setting
  - Changing environment
  - Lack of human experts
  - Continuous states
  - Long term horizons
  - ..

DOTA 2

AlphaStar & OpenAI Five

- Main approach?
- Parallelize training & train a lot!
- Use general learning strategies:

Proximal Policy Optimization (PPO)

	OPENAI 1V1 BOT	OPENAI FIVE
CPUs	60,000 CPU cores on Azure	128,000 preemptible CPU cores on GCP
GPUs	256 K80 GPUs on Azure	256 P100 GPUs on GCP
Experience collected	~300 years per day	~180 years per day (~900 years per day counting each hero separately)

Did I say a lot? I mean a lot!

AlphaStar & OpenAI Five

- These big players take the theoretical properties of the algorithms for granted, and use them on huge models.
- Moves the problem to an engineering point-of-view:
  How do I get as many training iterations as possible
- Shows how general these methods are
- How to see what these agents learn? Can we learn from the computer again?
- Strive for explainable and generalizable models (XAI)

Robotics (Boston Dynamics)

- What happens if we put some of these powerful algorithms to work in the real world?
- https://youtu.be/LikxFZZO2sk
- Basically, a combination of fancy sensors, powerful hardware and sophisticated learning methods.
- Very secretive about their methods, so hard to properly analyze

Robotics (Object manipulation)

- Application of OpenAI Five approach to some other task
- Train in simulation a lot, only finetune in the real world
- Again, gathering of enough data is the real problem
- Found policies correspond to human-like movements

- https://youtu.be/jwSbzNHGflM

Learning Dexterous In-Hand Manipulation

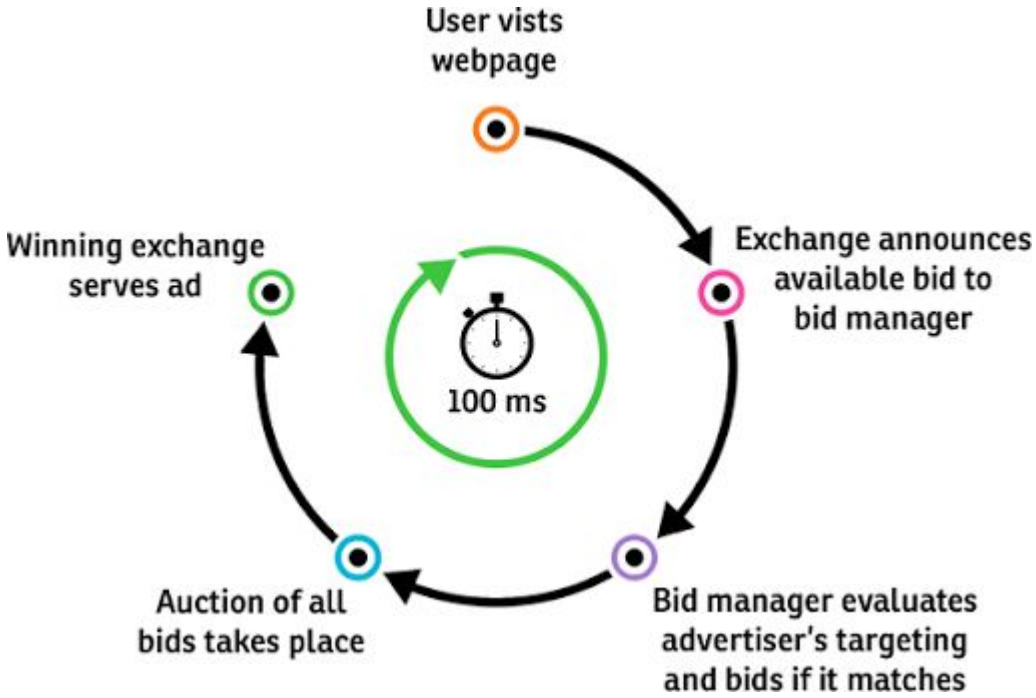
OpenAI; Marcin Andrychowicz, Bowen Baker, Maciek Chociej,
Rafał Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron,
Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor,
Josh Tobin, Peter Welinder, Lilian Weng, Wojciech Zaremba

Traffic Light Control

- Very practical problem, but hard to properly optimize using traditional methods
- Number of states in a traffic control grid are huge (like the games before), but there is no easy way to “reset.” In the worst care we end up having a grid-locked traffic jam
- Again, like more real-world AI relies on simulation for training
- Scale to multi-agent setting to allow for communication between traffic lights

Display advertising

- Use policies to find the best ad for a user/context pair
- Move away from A/B testing, and instead make incremental improvements using RL methods
- Use user clicks as feedback and reward signal.



Takeaway & more resources

Conclusion

- You’ve seen:
- A positioning of RL in the AI research area
- Implementation of tabular RL methods
  - MDP formalization
  - Policy evaluation
  - Policy improvement
  - Value iteration
- Real world examples of RL

Conclusion

- Hard to apply RL in the real world
- Thriving research area, but little examples in business
- Implications are huge: looking for general learning algorithms
- Need to explain these models and their decisions
  - Extensive testing
  - What caused certain actions
  - Accountability
  - How to deal with rapid automatization of tasks done by humans?
  - Ethics!
- Large gaps to fill -> good place for more experiments

Biggest players

- OpenAI
- NVIDIA
- (Google) DeepMind
- Facebook AI Research
- MILA
- Stanford
- *.. many more*

NVIDIA

DeepMind

Mila

Where to find interesting RL content

- Reinforcement Learning: an Introduction by Sutton & Barto (2018)
- arxiv.org
  - Most research is put on here, also by bigger companies
- Blog posts by the big three: DeepMind, OpenAI & FAIR
- Browse GitHub repos!

More recommended literature:

Artificial Intelligence: A Modern Approach by Russel & Norvig (2009)