Reinforcement Learning for Control

Dr. Gerhard Paaß
Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
Sankt Augustin

TensorFlow

Fraunhofer
BIG DATA AI

Course Overview

1. Intro to Deep Learning	Recent successes, Machine Learning, Deep Learning & types
2. Intro to Tensorflow	Basics of Tensorflow, logistic regression
3. Building Blocks of Deep Learning	Steps in Deep Learning, basic components
4. Unsupervised Learning	Embeddings for meaning representation, Word2Vec, BERT
5. Image Recognition	Analyze Images: CNN, Vision Transformer
6. Generating Text Sequences	Text Sequences: Predict new words, RNN, GPT
7. Sequence-to-Sequence and Dialog Models	Transformer Translator and Dialog models
8. Reinforcement Learning for Control	Games and Robots: Multistep control
9. Generative Models	Generate new images: GAN and Large Language Models

: link to background material, : link to images used in lecture, G. : Terms that may be asked in the exam

Reinforcement Learning

Agenda

1. Introduction
2. Q-Learning
3. CartPole Example in Stable Baselines3
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

Introduction

- Supervised Learning: input x_t ➔ corresponding output y_t
- **Dynamic Control**: direct output **not available**
  - Chess: output (win / loose) given after many moves
  - Video game: output (next level / death) only given after many actions
  - Car driving: output (crash / reach destination)
  - Robot control
  - Portfolio management: buy / sell stock
- Need to compute optimal **actions** sparse and time-delayed **rewards**
  ➔ different learning approach
- Psychology:
  - Learning occurs by later "reinforcement"
    Free Book [Sutton 2018]

Chess pieces with colorful bokeh by Mukumbura CC BY-SA 2.0

Atari videogame Screenshot from simulator

©metamorworks - stock.adobe.com

Andy Hill / public domain

©metamorworks - stock.adobe.com

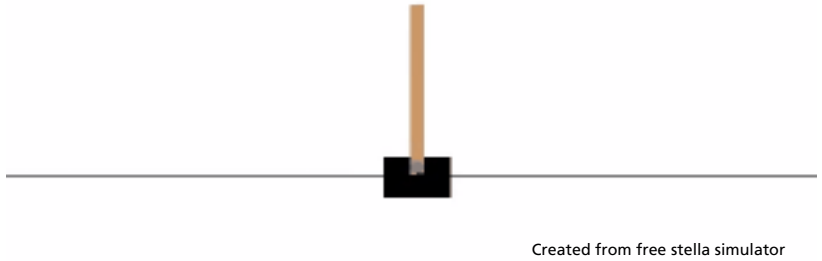
Reinforcement Learning

G. dynamic control G. reinforcement learning applications

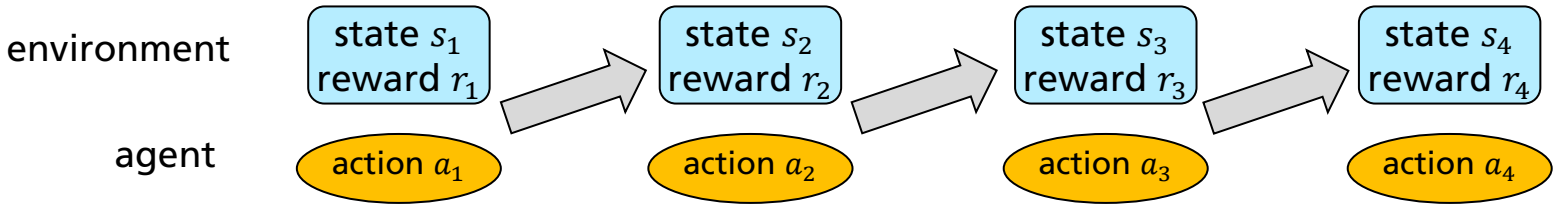
Fraunhofer BIG DATA AI

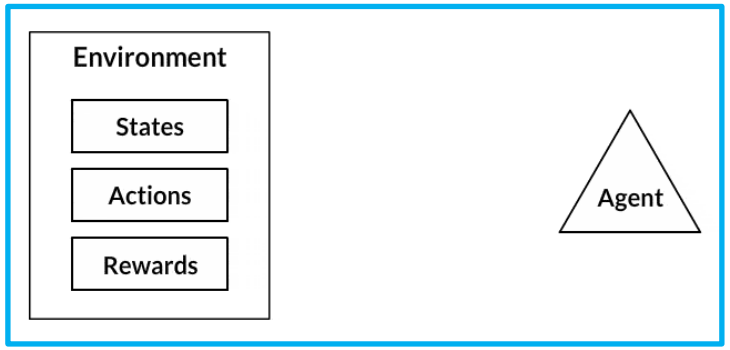
Mathematical Formulation

- Example: Pole Balancing



- Describe **environment** at time t by **states**: $s_t \in S$
  e.g. position of cart, angle of pole, velocities
- The agent can take possible **actions** at time t by: $a_t \in A$
  e.g. shift cart left / right
- State transition:
  when agent acts the environment changes ➔ new state, reward
  e.g. reward = -10 if outside, else reward = 1





G. reinforcement learning G. reward

Mathematical Formulation

- **Episode**: the series $(s_1, r_1, a_1), (s_2, r_2, a_2), \ldots, (s_n, r_n, a_n)$ until environment reaches a terminal state
- **Policy**: A map assigning an action to each state $\pi: s \to a$

depends on actions

- **Target**: Maximize future reward $\max_{\pi}(r_1 + r_2 + \cdots + r_n)$
  - E.g. for pole balancing a wrong move may lead to a loss many steps later
  - generalization: may be stochastic
- The equal consideration of all rewards is problematic
  - With infinite time steps the expression may diverge
  - optimizing for a very far rewards may be not relevant at present
  - the policy does **not** try to get rewards **fast**

➔ Future rewards should be taken into account with a **lower weight**

G. policy G. episode

Deterministic and Stochastic Environment

Deterministic environment

- both the next state and the next reward are deterministic functions. $f(s_{t+1}, r_{t+1}|s_t, a_t)$
- If the agent chooses a_t in state s_t yields always the same next state s_{t+1} and the same reward value r_{t+1}

Stochastic environment

- both the next state and the next reward may be **random** to some extent $p(s_{t+1}, r_{t+1}|s_t, a_t)$
- If the agent chooses a_t in state s_t he may get different next states s_{t+1} and rewards r_{t+1}

- Deterministic environments are easier to solve, e.g. construct a graph of all subsequent actions and states.
- That is not possible for the stochastic environment

We consider the deterministic case

G. deterministic environment G. stochastic environment

Reinforcement Learning

Agenda

1. Introduction
2. Q-Learning
3. CartPole Example in Tensorflow
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

Q-Function

- Lower weight for rewards in the far future: **discounted future reward**
  - $R_t = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \cdots + \gamma^{n-t} * r_n$
  - $= r_t + \gamma R_{t+1}$ where $\gamma < 1.0$

    discount factor γ
  - e.g.: $\gamma = 0.9$: $\gamma^{20} = 0.121, \gamma^{50} = 0.005\,, \gamma^{100} = 0{,}000026$

- Good strategy
  Choose an action that **maximizes** the **discounted future rewards**

- **Q-function** : assume you know the **optimal policy** π^* for all future states
  - **maximum** discounted future reward when action a_t is performed in state s_t

$$Q(s_t, a_t) = \max_{\pi^*} (r_{t+1} + \gamma * r_{t+2} + \cdots + \gamma^{n-t-1} * r_n)$$

Apply optimal policy π^* for all future states

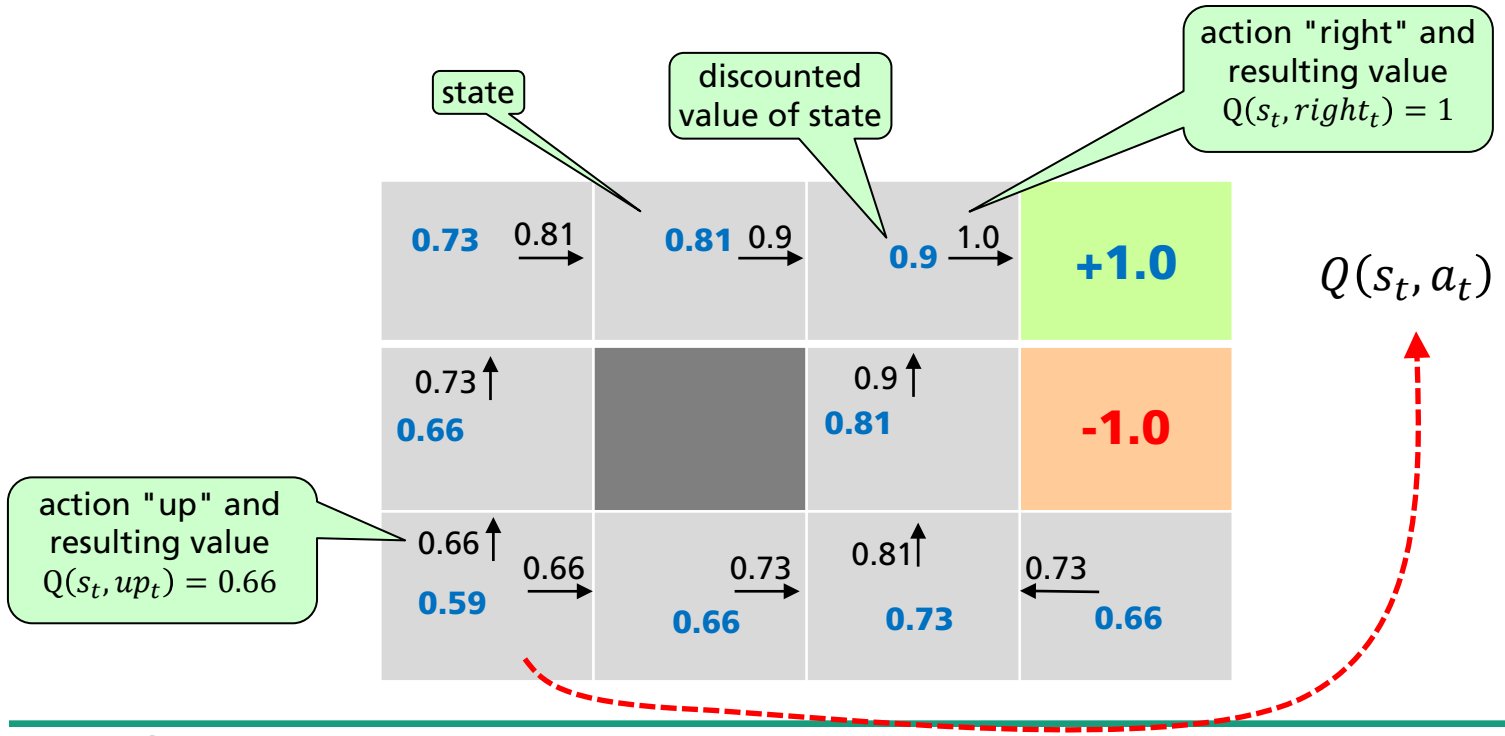
reward resulting from s_t, a_t

Q-function: discounted future reward an agent in state s_t can receive for action a_t

G. discounted future reward G. Q-function

Example: Deterministic Robot Navigation

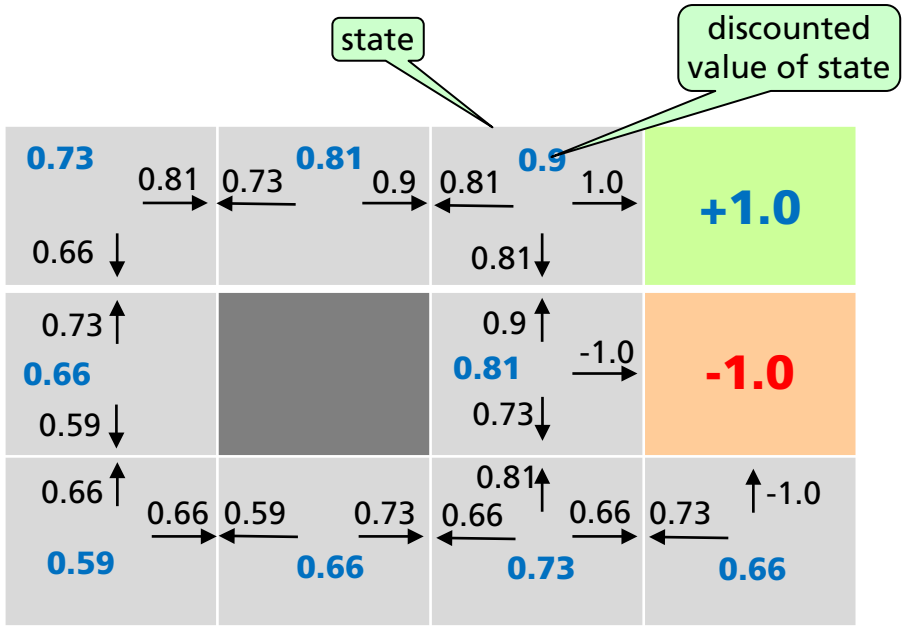
- Actions: UP, DOWN, LEFT, RIGHT
- Rewards: +1 at [4,3], -1 at [4,2], 0 else
  - $R_t = r_t + 0.9 * R_{t+1}$
- **Q-table** $Q(s_t, a_t)$: discounted future reward



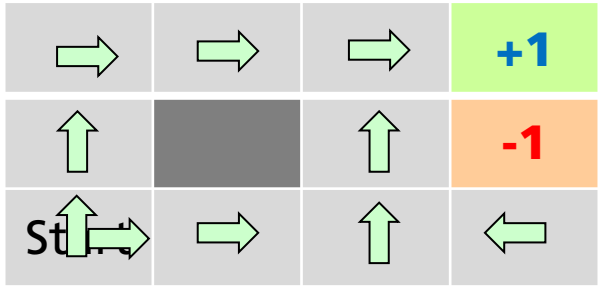
$$Q(s_t, a_t) = \max_{\pi^*} (r_{t+1} + \gamma * r_{t+2} + \cdots + \gamma^{n-t-1} * r_n)$$
$$= 0 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \gamma^4 0 + \gamma^5 1.0$$
$$= 0.59$$

Example: Deterministic Robot Navigation

- Actions: UP, DOWN, LEFT, RIGHT
- Rewards: +1 at [4,3], -1 at [4,2], 0 else
  - $R_t = r_t + 0.9 * R_{t+1}$
- **Q-table** $Q(s_t, a_t)$: discounted future reward



- Optimal strategy: select action with maximal Q-value



Bellman Equation

- **Q-function**

$$Q(s_t, a_t) = \max_{\pi^*} \left(r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + \cdots + \gamma^{n-1} * r_{t+n}\right)$$

π^* is the (unkown) best policy

- Reformulation: assume we know $(s_t, a_t, r_{t+1}, s_{t+1})$

$$Q(s_t, a_t) = r_{t+1} + \gamma * \max_{\pi^*} \left(r_{t+2} + \gamma^1 * r_{t+3} + \cdots + \gamma^{n-2} * r_{t+n}\right)$$

γ is factored out

same

$$Q(s_t, a_t) = r_{t+1} + \gamma * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

π^* selects best a_{t+1}

Establishes a relation between different (s_t, a_t) of the Q-function

historical photo of American mathematician Richard E. Bellman Fair use

[Bellman: Dynamic Programming 1957]

- **Bellman equation**: recursive relation
- If we know $Q(s_t, a_t)$: How to find best action ?
  ➔ in state s_t take the action a such that $Q(s_t, a) = \max_a Q(s_t, a)$

$Q(s_t, a_t)$ defines Optimal Policy

Approximate by Neural Network

- Bellman equation: $Q(s_t, a_t) = r_{t+1} + \gamma * \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1})$

  (discount factor)

- **Q-Function**
  - Problem: the number of states often is extremely large (e.g. chess $\sim 10^{46}$)

- Approximate $Q(s_t, a_t)$ by a **neural network** $\hat{Q}(s_t, a_t; w)$
  - good representation of functions, small number of parameters w
  - Can detect inputs (s_t, a_t) with similar Q-values: generalization
  - ➔ **deep Q-network**

Deep Q-Network Training

- Generate episode: $(s_1, r_1, a_1), (s_2, r_2, a_2), \ldots, (s_n, r_n, a_n)$
  - $\hat{Q}(s, a; w)$ current model with weights w
  - for each $(s_t, a_t, r_{t+1}, s_{t+1})$

reward for s_1, a_1

Bellman equation

$$Q(s_t, a_t) = r_{t+1} + \gamma * \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1})$$

difference should be 0

$$loss_t = \left(\hat{Q}(s_t, a_t; w) - \left[r_{t+1} + \gamma * \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}; w)\right]\right)^2$$

compute by predicting neural network

- sum losses for all t
  - compute gradient of loss sum
  - Update w in model $\hat{Q}(s, a; w)$ by stochastic gradient

[Watkins 1989]

- Deep Q network training
  - good exploitation of information
  - works also for stochastic environments

Deep Q-Network Training Tricks

- unstable procedure as $\hat{Q}(s_t, a; w)$ values …
  - are constantly changing, highly correlated

- **Replay memory**
  - $\hat{Q}(s_t, a; w)$ generates episodes $(s_1, r_1, a_1), \ldots, (s_n, r_n, a_n)$ ➔ store
  - randomly sample a batch of transitions $< s_t, a_t, r_{t+1}, s_{t+1} >$
  - compute gradient and update $\hat{Q}(s_t, a; w)$ by stochastic gradient

Reduces correlation

- **Exploration**
  - Need to explore all possible states ➔ find best policy
  - If replay memory is generated with best actual policy ➔ never visit large areas of the state space
  - Generate replay memory with **ϵ-greedy strategy** with probability ϵ choose random action otherwise choose best actual action ➔ ϵ may decrease with time

Reinforcement Learning

Agenda

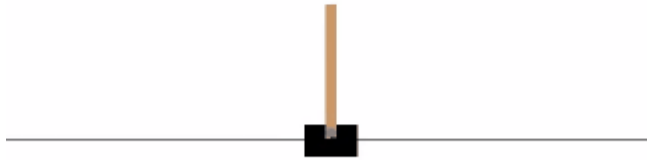
1. Introduction
2. Q-Learning
3. CartPole Example in Stable Baselines3
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

CartPole

- Goal: balance a pole connected with one joint on top of a moving cart.
- Actions
  - 0: push cart to left
  - 1: push cart to right
- State vector

Feature	Min	Max
Cart position	-2.4	2.4
Cart velocity	-Inf	Inf
Pole angle	~ -41.8°	-41.8°
Pole Velocity At Tip	-Inf	Inf

- Reward: 1 for every step taken
- Starting States: random values in [-0.05,0.05]



- Termination
  - |Pole angle| greater 12 deg.
  - |Cart position| greater 2.4
  - Episode length > 200
- Model of the environment
  - provided in gym
    https://gym.openai.com/envs/CartPole-v0/

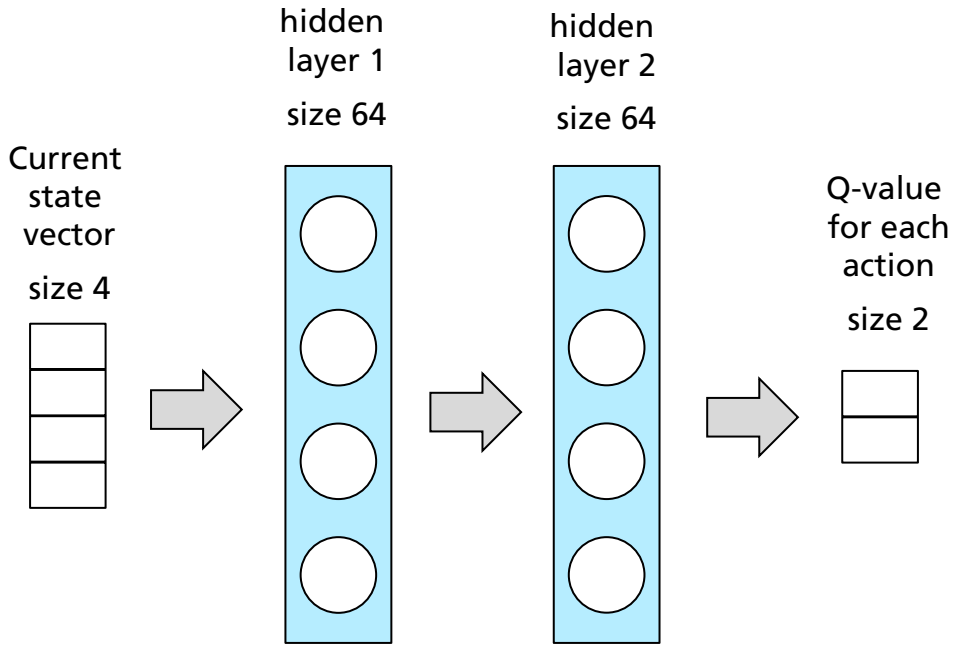
Implementation in StableBaselines3: Q-Value Network

- StabeBaselines3: RL-algorithms with the same interface ➔ Similar to scikit.learn for data mining

```
env = gym.make("CartPole-v1", render_mode="rgb_array")  # generates environment answer
model = DQN("MlpPolicy", env, verbose=1)                # Deep Q-Network
model.policy
```

- Policy is computed with an MLP

```
DQNPolicy(
    (q_net): QNetwork(
        (features_extractor): FlattenExtractor(
            (flatten): Flatten(start_dim=1, end_dim=-1) )
    (q_net): Sequential(
        (0): Linear(in_features=4, out_features=64, bias=True)
        (1): ReLU()
        (2): Linear(in_features=64, out_features=64, bias=True)
        (3): ReLU()
        (4): Linear(in_features=64, out_features=2, bias=True)
        )
    )
....
```



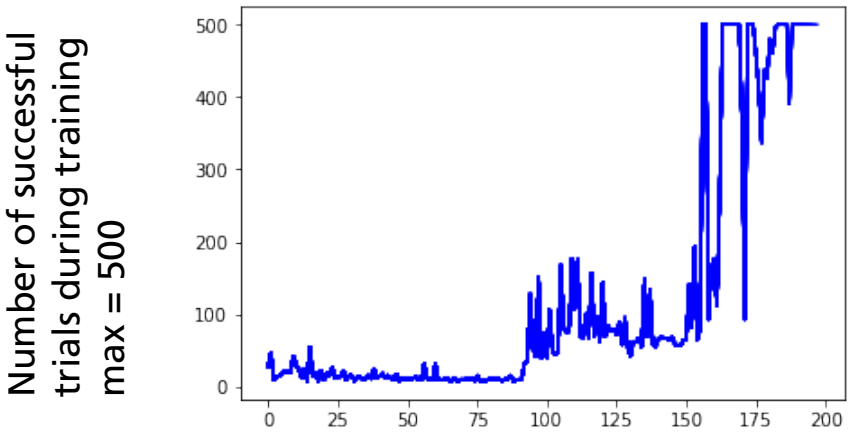
Generate observations from current model

- A standard 3-layer network is used
  - input: length of state vector, output: length of action vector, hidden sizes: 64
- Tricks to stabilize learning
  - replay buffer to store parts of episodes
  - gradient clipping to avoid large steps
  - Two networks of same architecture are used.
    - Primary network for Q-value computation,
    - Target Network for action selection, parameters copied with delay ➔ reduces bias.
- Training

```
model.learn(total_timesteps=200_000)
```

- Application of trained model

```
obs, info = env.reset()
while True:
    action, _states = model.predict(obs)
    obs, reward, done, truncated, info = env.step(action)
```



Reinforcement Learning

Agenda

1. Introduction
2. Q-Learning
3. CartPole Example in Tensorflow
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

Q-Learning: Atari Games

- Location of sumarine, position & direction of sharks too specific ➔ use pixels of video frames
- **Input**: State s
  4 video frames 210 x 160 pixels, 128 colors
- **Output**: Q-values for all actions $Q(s, a_i, ; w)$
  - 3 convolutional layers
  - 2 fully connected layers
  - Targets computed as above

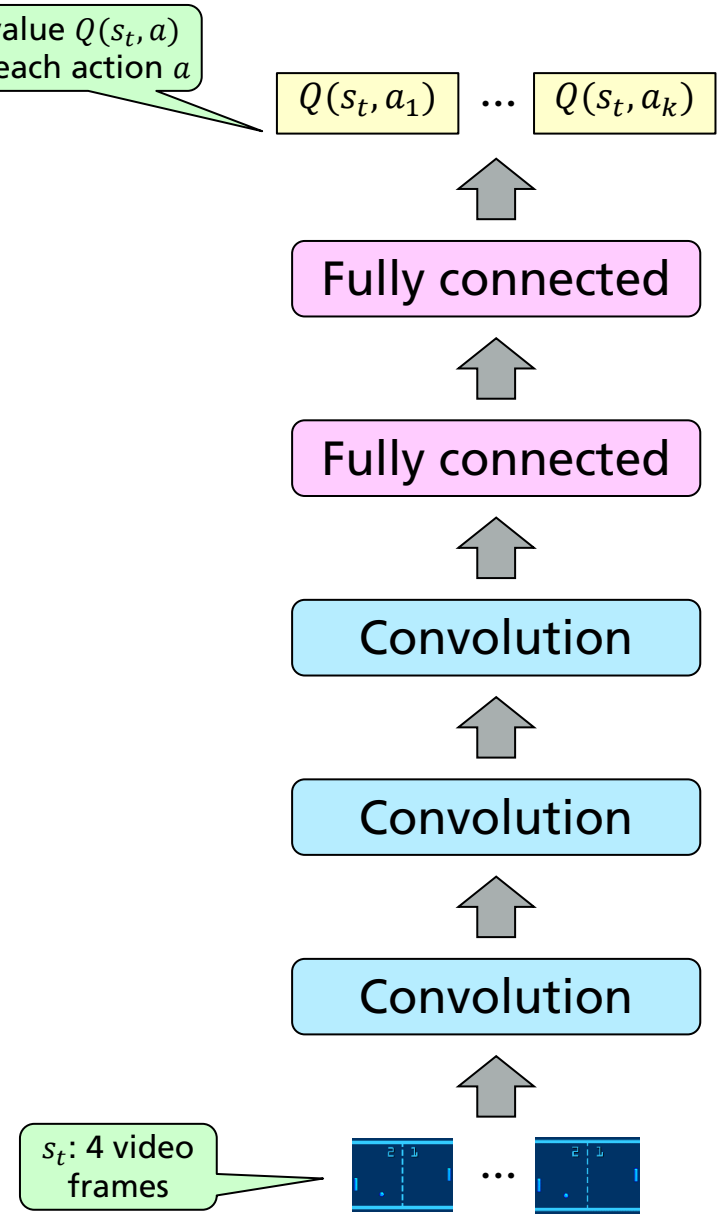
Training

- 50 million frames
- 38 days of game play

videos

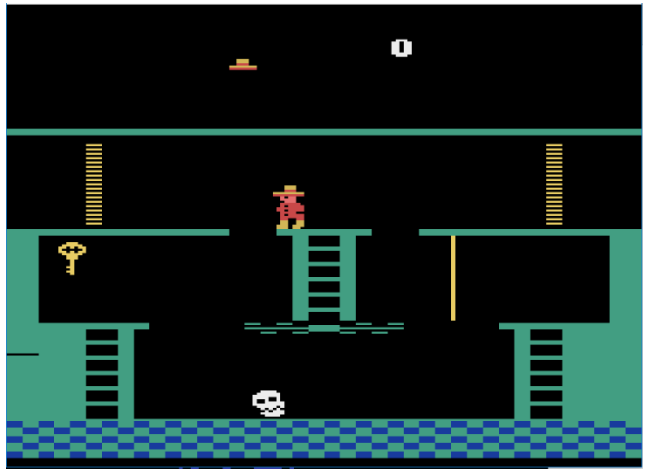
- Seaquest
- Space invaders

Atari videogame Screenshot from simulator



Atari Games: Results

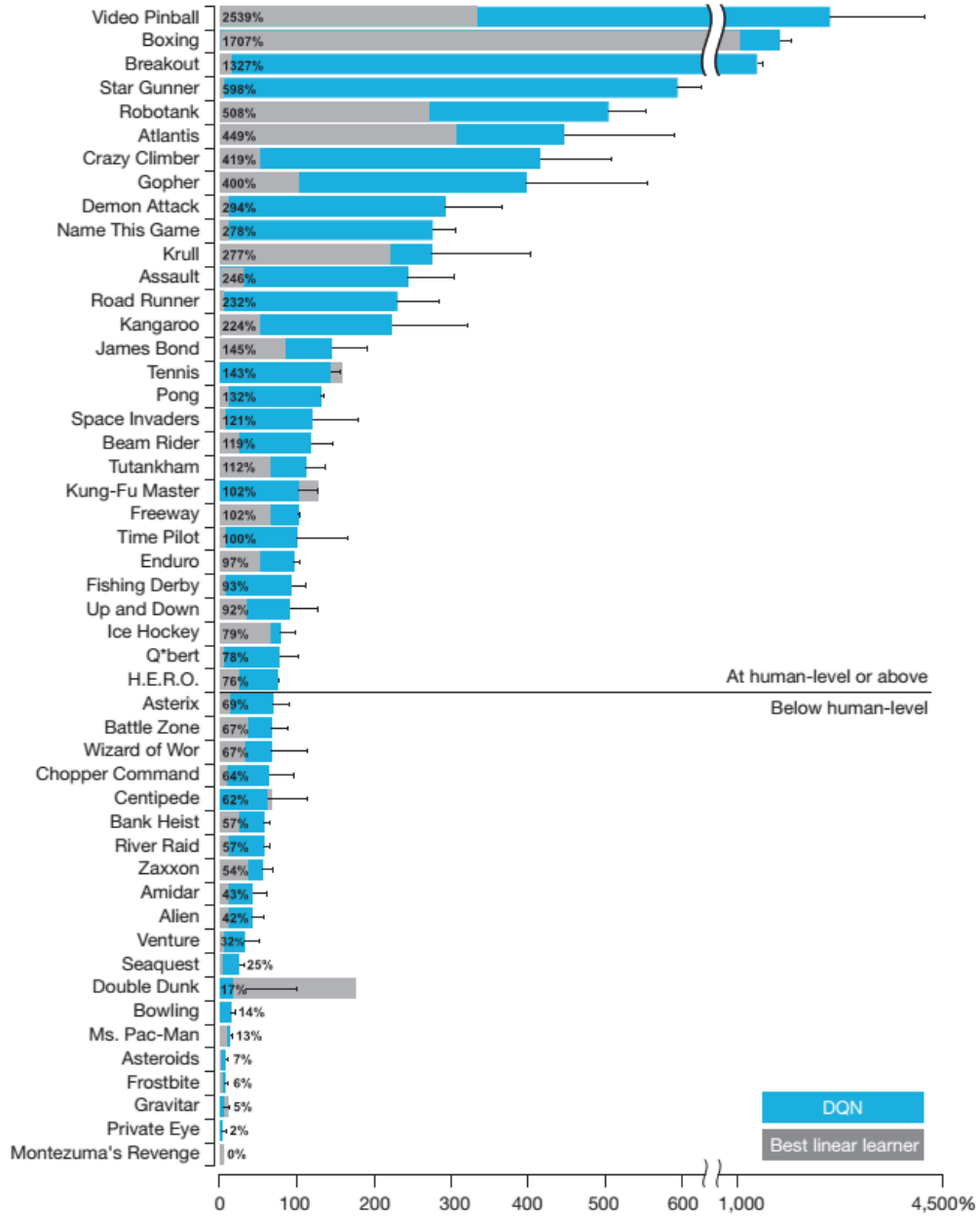
- 49 Atari 2600 games [Mnih et al. 2015]
- Percentage: a professional human game tester has 100%
  - Blue: Deep Q-network, Grey: best other learners
- Same architecture and hyperparameters for all games
- Achieves human-level performance for many games
- Improved results in [van Hasselt et al. 2016]
- even solved Montezuma's revenge video



eigener screenshot vom public domain stella emulator

Need better exploration

- Remember previously visited states
- first return to a promising state, then explore from it
- solve simulated environments



Reinforcement Learning

Agenda

1. Introduction
2. Q-Learning
3. CartPole Example in Tensorflow
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

Example: Rock-Paper-Scissors

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock

4th UK Rock Paper Scissors Championships by james bamber / CC0

- Consider policies for iterated rock-paper-scissors
  - A deterministic policy is always suboptimal
  - A uniform **stochastic policy is optimal** (i.e. Nash equilibrium)

Random policies are necessary: Mixed Strategy

G. stochastic policy

Policy-Based Reinforcement Learning

- **Stochastic policy:** $\pi\,(a|s)$ selects an **action** a for given **state** s according to a probability distribution

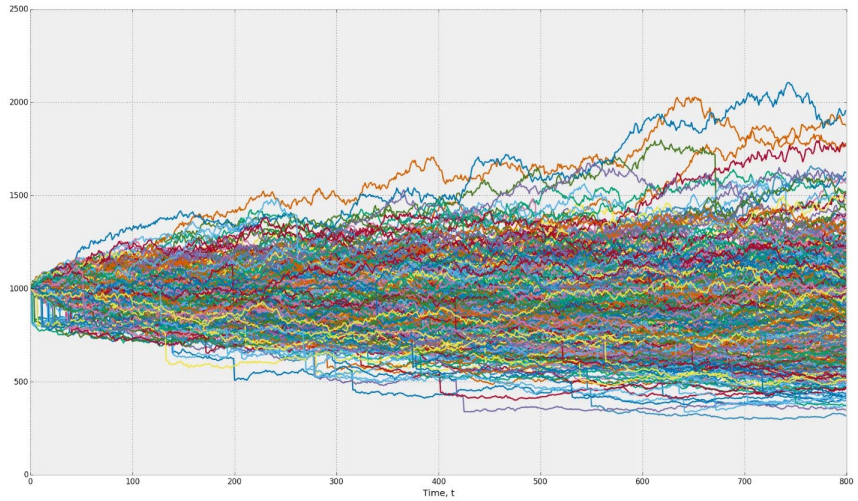
Policy model

- Compute stochastic policy by a neural network $\hat{\pi}(a|s;w)$ with parameter w

- For $k = 1 \ldots K$
  - Generate Data: starting state s_1
    1. Policy model $\hat{\pi}(a_t|s_t;w)$ randomly generates an action a_t.
    2. Environment randomly generates the next state s_{t+1} and reward r_{t+1} by $p(s_{t+1}, r_{t+1}|s_t, a_t)$
    3. continue with step 1.
  - yields an **episode** $\tau_k = (s_1, r_1, a_1), (s_2, r_2, a_2), \ldots, (s_n, r_n, a_n)$
  - compute sum $R(\tau_k)$ of discounted rewards

  ➔ Generates an **episode distribution of a stochastic policy**

- mean value of discounted rewards $V(w) = \frac{1}{K}[R(\tau_1) + \cdots + R(\tau_K)]$
- for large K: mean value of discounted rewards $V(w)$ is criterion for maximization ➔ **expected value of a policy**



G. Episode Distribution of a Stochastic Policy

G. Expected Val-ue of a Policy

Policy Gradient: Algorithm REINFORCE

- Initialize policy parameters w arbitrarily
- For $i = 1$ to $numIter$
  - Generate episode $\tau^{(i)} = (s_1, r_1, a_1), (s_2, r_2, a_2), \ldots, (s_n, r_n, a_n)$ by $\pi(a_t|s_t; w)$
    - For each (s_t, r_t, a_t) update w using gradient $\partial V(w)/\partial w$ and learning rate α
- Return w

updated parameter w

complicated derivation

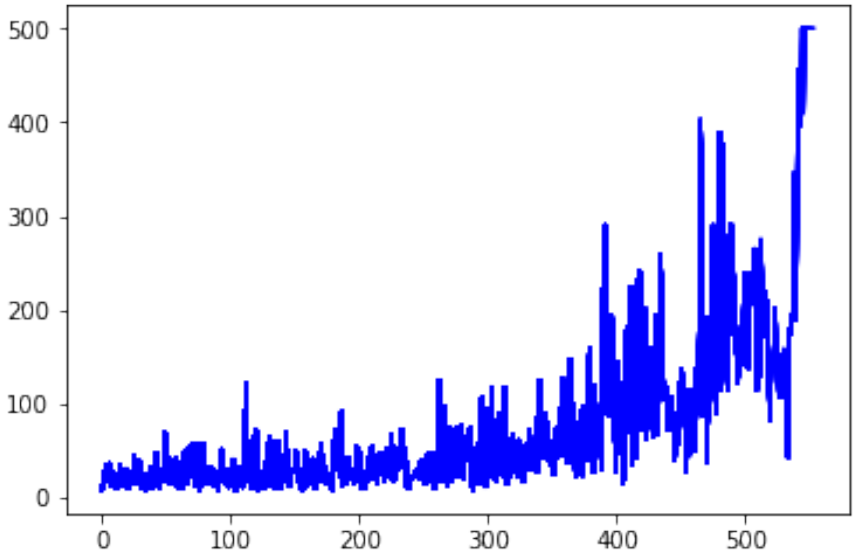
- **Policy gradient optimization** maximizes expected value $V(w)$ of discounted reward
  ➔ use gradient $\partial V(w)/\partial w$
- **Advantages:**
  - Better convergence properties, effective in high-dimensional or continuous action spaces
  - Can learn **stochastic** policies
- **Disadvantages**:
  - Typically converges to a local rather than global optimum
  - Evaluating a policy is often inefficient and has high variance

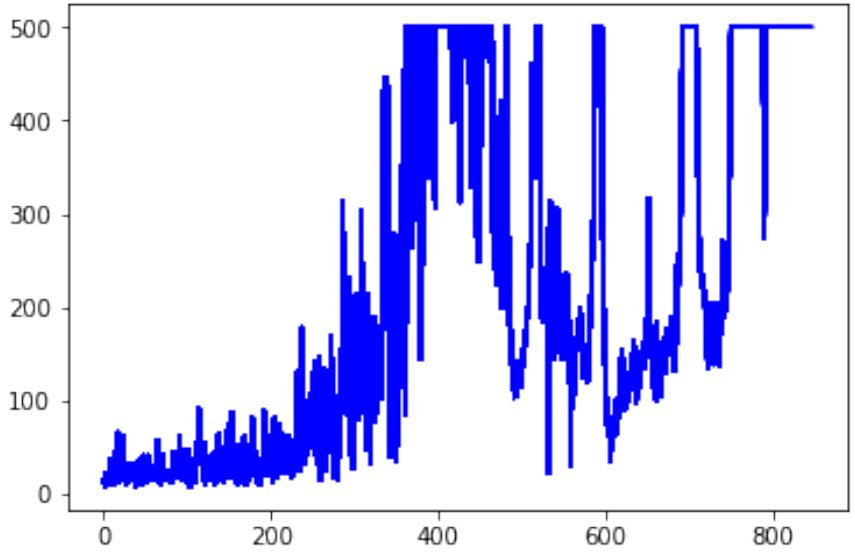
Results

Model summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 24)                120
_________________________________________________________________
dense_20 (Dense)             (None, 24)                600
_________________________________________________________________
dense_21 (Dense)             (None, 2)                 50
=================================================================
Total params: 770
Trainable params: 770
Non-trainable params: 0
```

Number of successful trials during training
max = 500

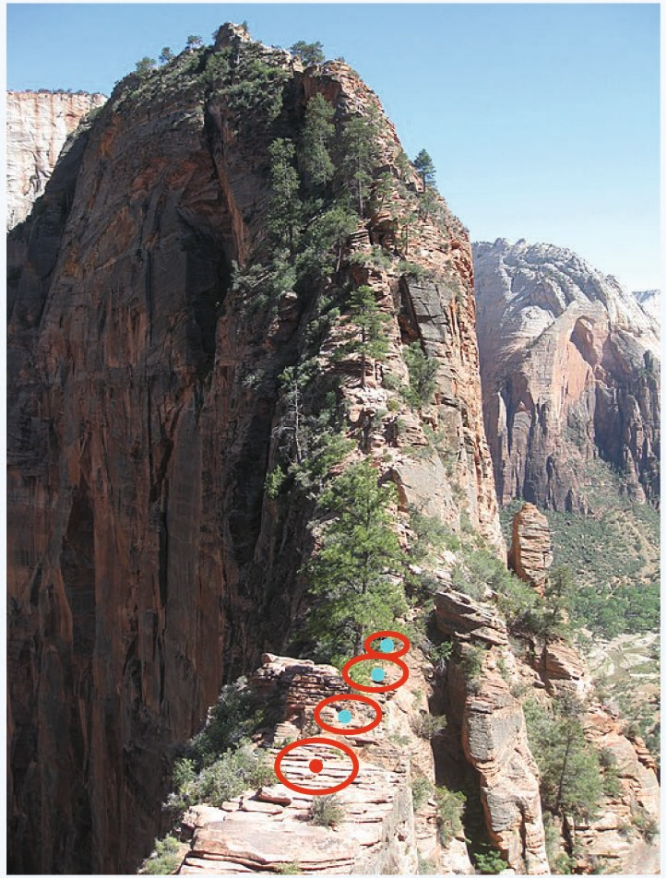




PPO Proximal Policy Optimization

Policy gradient optimization

- gradient: log probability of actions $\log(\pi_w(a|s))$
  to trace the impact of actions
- a different objective is $r(w) = \log \pi_w(a|s) \,/\, \log \pi_{w,old}(a|s)$,
  where $\pi_{w,old}(a|s)$ is the action probability under the
  previous policy ➔ use as a surrogate target
- if $r(w)$ is very large: **Clipping** ➔ reduce step to some value
  try to update policies conservatively:
  use **Kullback-Liebler divergence** as a measure for difference



PPO Proximal Policy Optimization

- turn the KL divergence from a constraint to a penalty term,
  similar to for example to L1, L2 weight penalty

PPO: a very effective and stable approach

Reinforcement Learning

Agenda

1. Introduction
2. Q-Learning
3. CartPole Example in Tensorflow
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

Reinforcement Learning Tips and Tricks

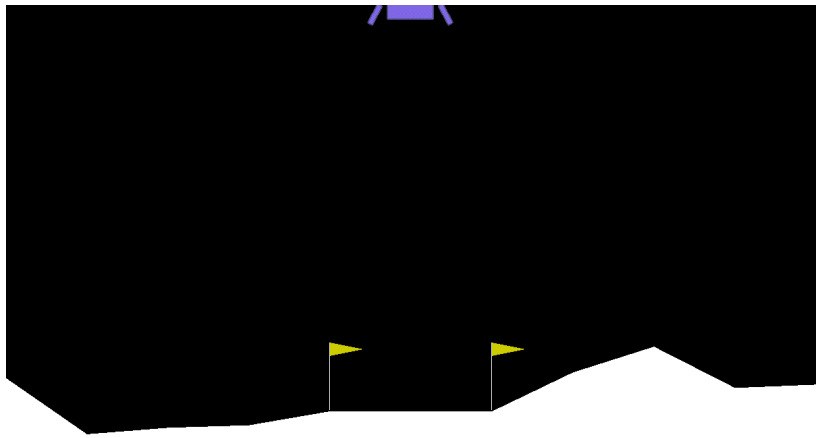
Difference to other ML Methods

- Data to train the agent is collected through interactions with the environment
  ➔ can lead to vicious circle: if the agent collects poor quality data (e.g., trajectories with no rewards), then it will not improve and continue to collect bad trajectories.
  - Results may vary with seeds. Always try several runs
  - **Hyperparameters** are important: tuning may be necessary ➔ look at RL zoo
  - Always **normalize** inputs to $[-1,1]$
- Model-free algorithms (without an internal model of the env.) as in stableBaselines3,
  ➔ usually ***sample inefficient*** ➔ require many interaction (often millions)
  - Usually very many training steps
  - Reward engeneering often necessary to get a good reward function
  - training instability: use PPO or similar approaches
- **Evaluation**: use extra test environment without exploration noise
  - use deterministic =True even for stochastic algos (e.g. PPO) ➔ usually better performance

https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html

Which Reinforcement Algorithm?

- **Discrete action** (single process)
  - DQN with (double DQN, prioritized replay, …)
  - DQN is usually slower to train but is the most sample efficient (because of its replay buffer).
- Discrete action (several parallel processes)
  - PPO or AC2
- **Continuous action** (single process)
  - SAC, TD3, TQC
- Continuous action (several parallel processes)
  - PPO, TRPO, AC2
- Creating a **custom environment**: simulation model
  - **normalize** your **observation space**
  - **normalize** your **action space** and make it symmetric when continuous.
    A good practice is to rescale your actions to lie in [-1, 1].
  - start with shaped reward (i.e. **informative reward**) and **simplified** version of your problem
  - **debug** with random actions. Check if the environment works and follows the gym interface:



from Gym LunarLander

https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html

Applications: Games

Go (igo) board with stones by Dilaudid / CC BY-SA 3.0

- Required: know the rules / dynamics of the environment
  - **AlphaGo** learns to play Go using
    - Value networks to evaluate board positions & Policy network to select moves
    - Monte Carlo tree search to find its moves. NN improves strength of tree search.
    - combination of supervised learning from human expert games, and reinforcement learning from games of self-play.

- OpenAI Five defeats **Dota2** world champions
  - 2 teams of 5 characters with different capabilities fight against each other
  - each team occupying and defending their own separate base on the map.
- Each character is controlled by a separate **PPO** agent:
  ➔ OpenAI defeated the world champion team 2:0

Link to video

Applications: End-to-end Driving

- existing dataset of stable human collected driving data
  - VISTA, a photorealistic, scalable, data-driven **simulator** for synthesizing a continuum of new perceptual inputs locally around an existing data
- An end-to-end **reinforcement learning** pipeline for training autonomous lane-stable controllers
  - only visual inputs and sparse reward signals, without explicit supervision using ground truth human control labels
- Experimental validation: agents trained in VISTA can be deployed directly in the real-world and achieve more robust recovery compared to previous state-of-the-art imitation learning models
- Amini et al. 2020: Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation

©metamorworks - stock.adobe.com

Applications: Robotics

- Application of RL in Advanced Robots
  - Robotic manufacturing
  - Predictive maintenance
  - Robotic Surgery
  - Military robotics
  - Agriculture robot
  - Service robotics

Industrial Robots

Robotic surgery

Agriculture robot

- Application of RL in Advanced Transportation Systems

Drones

Ship navigation

Aeronautical industry

Taxi services

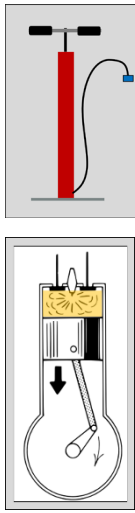
all AI generated

Application: RLHF Reinforcement learning with Human Feedback

- **Task:**
  Adapt GPT such that the desired answer is given for a prompt
- Instruction tuning **InstructGPT**
  [Ouyang et al. 2022]

- InstructGPT 1.3B outputs are
  - preferred to 175B GPT-3 85% of the time, and
  - preferred 71% when using few-shot prompts with GPT-3.
- For QA InstructGPT hallucinates information 21% of the time vs 41% in GPT-3

1. Train a **reward model** for ranking answers
   1. Use GPT-3 model to generate several answers for a prompt.
      Example prompt: Explain the heat pump to a 6 year old.
      A: When you press a bicycle air pump firmly …
      B: An internal combustion engine is a heat engine…
      C: Friction is a force acting between particles …
   2. Ask humans to rank the answers according to quality: A > C > B
   3. Train a reward model to rank the answers with a score



2. Train a stepwise model by **RL** to reproduce the ranking
   1. A new prompt is sampled from the training set.
      Example prompt: Explain how the airplanes were invented.
   2. The model generates a token at each time step.
      People had looked at how the birds fly …
   3. The model is updated using the proximal policy algorithm based on the reward given to the entire answer.

Do you know what time it is?

- 0.1 Yes
- 0.9 Quarter to twelve

standard approach used by ChatGPT

Reinforcement Learning

Agenda

1. Introduction
2. Q-Learning
3. CartPole Example in Tensorflow
4. Atari 2600 Games
5. Policy Optimization
6. Applications
7. Summary

Summary

- **Reinforcement learning** for problems with delayed rewards
  - Use deep neural networks as function approximators
  - Generate new data with algorithm

Chess pieces with colorful bokeh by Mukumbura CC BY-SA 2.0

Main approaches

- **Q-function** is the maximal future value of a state-action pair
  - relatively fast learning
  - Human performance in video game control

©metamorworks - stock.adobe.com

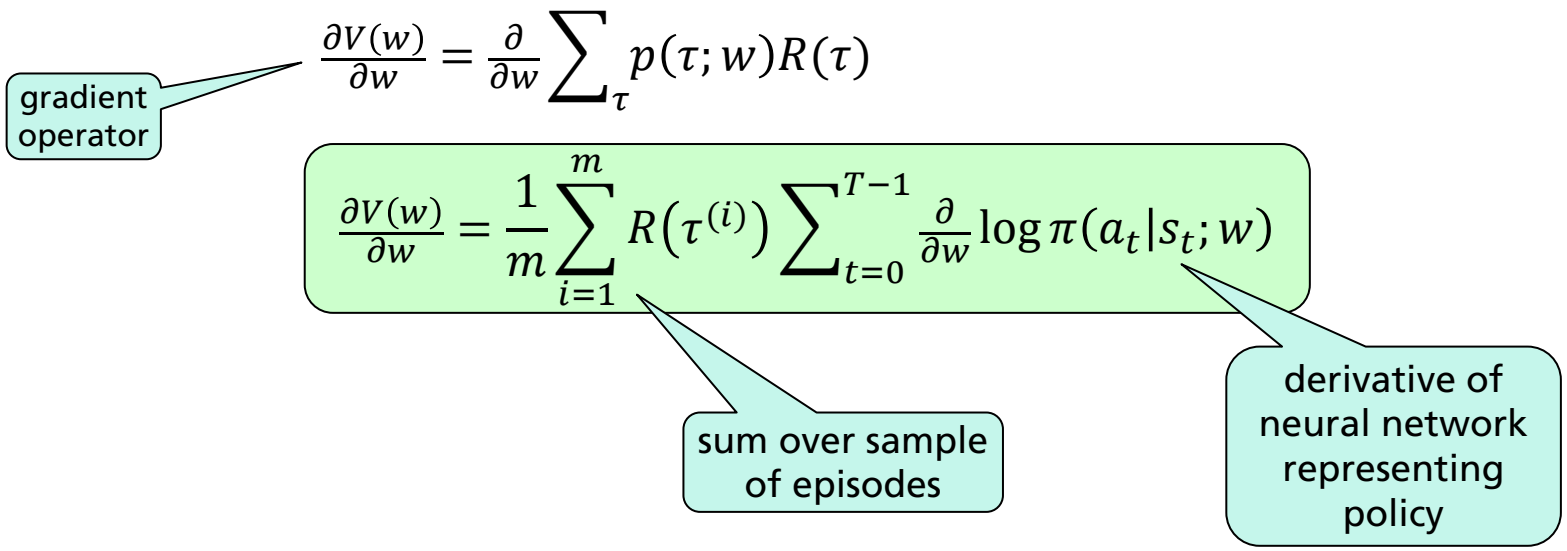
- **Policy Optimization** directly approximates the policy function
  - can represent stochastic policies
  - relatively high variance during learning
  - applicable to robot “end-to-end” learning from the RGB pixels of the camera to motor torques.
- Need to include better search, memory, and planning modules

Disclaimer

Responsible contact: **Dr. Gerhard Paaß**, Fraunhofer IAIS, Sankt Augustin
E-mail: gerhard.paass@iais.fraunhofer.de

Policy Gradient: Calculating the Derivative

- Take the gradient of the expected value of a policy with respect to w:



- Details of derivation in appendix

Policy Gradient: Algorithm REINFORCE

- Initialize policy parameters w arbitrarily
- For $i = 1$ to N
  - Generate episode $\tau^{(i)} = (s_1, r_1, a_1), (s_2, r_2, a_2), \dots, (s_n, r_n, a_n)$ by $\pi_w(a_t|s_t)$
    - For $t = 1$ to n do

$$w \leftarrow w + \alpha R\left(\tau^{(i)}\right) \sum_{t=0}^{T-1} \frac{\partial}{\partial w} \log \pi(a_t|s_t; w)$$

Learning rate

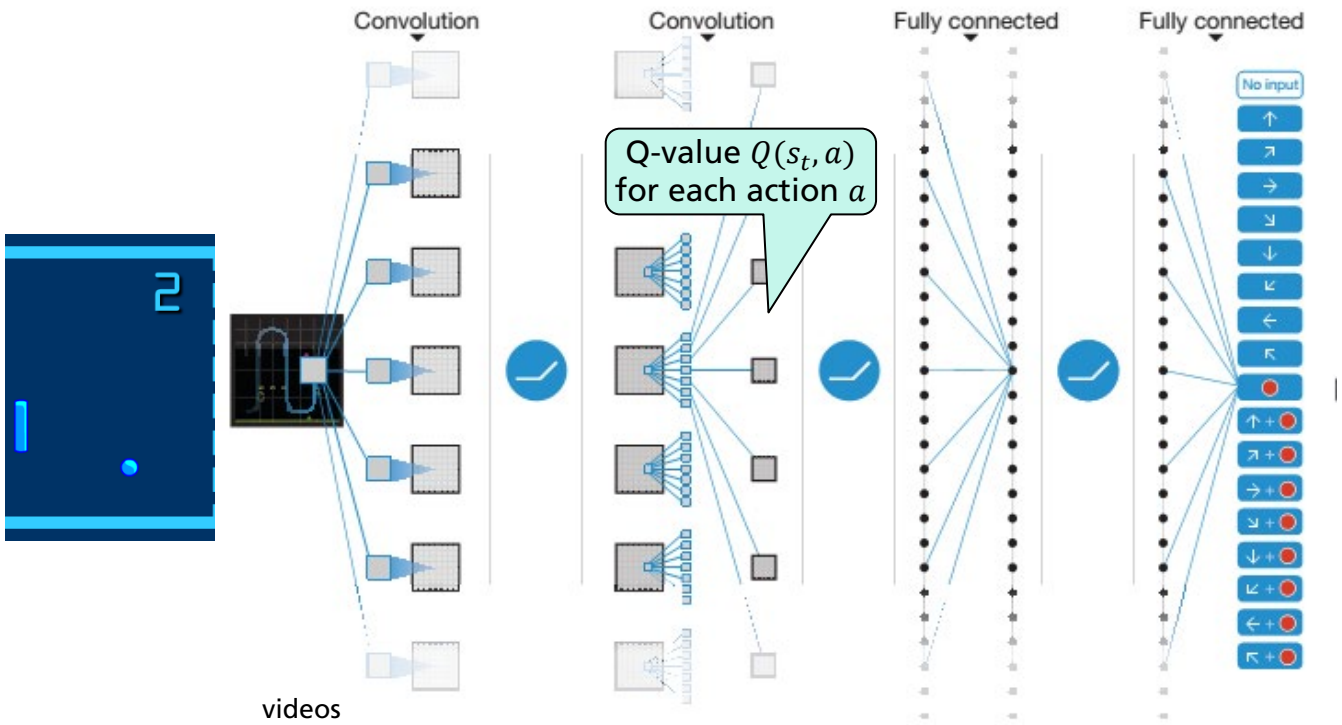
Neural network

- Return w

- **Advantages** of policy-based RL
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn **stochastic** policies
- **Disadvantages**:
  - Typically converges to a local rather than global optimum
  - Evaluating a policy is often inefficient and has high variance

Q-Learning: Atari Games

- Location of paddle, position & direction of ball too specific ➔ use pixels of video frames



videos

- Seaquest
  https://www.youtube.com/watch?v=v7T5El42QGE
- Space invaders
  https://www.youtube.com/watch?v=iqXKQf2BOSE