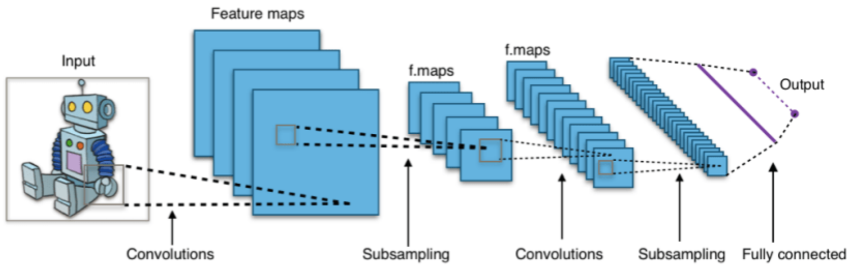
Imitation Learning

Jannik Zürn

Terminology

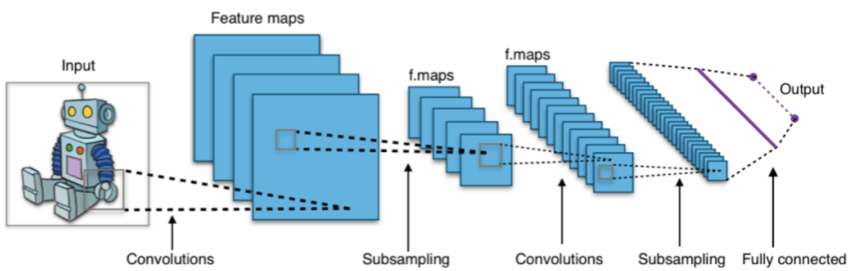


0.6	Cat
0.0	Dog
0.0	Car
0.4	Liquid

$\mathbf{x}$

$$f(y|\mathbf{x})$$

y



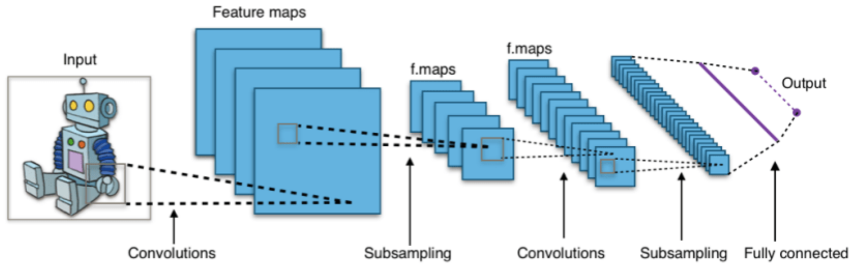
0.1	Left
0.1	Right
0.8	Straight

$\mathbf{o}_t$

$$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$$

$\mathbf{a}_t$

Mapping observations to actions



0.1	**Left**
0.1	**Right**
0.8	**Straight**

$\mathbf{o}_t$

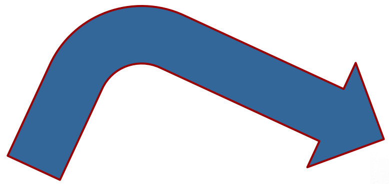
$$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$$

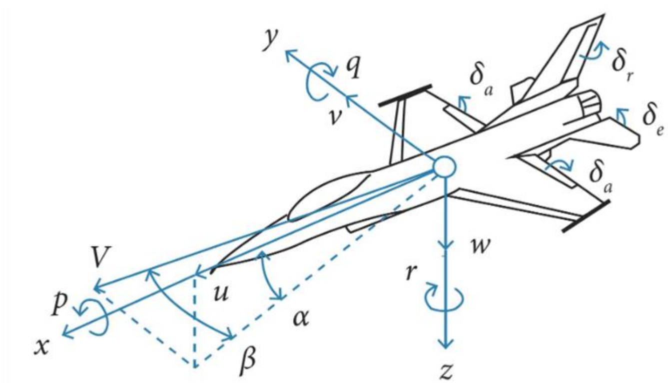
$\mathbf{a}_t$

$$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t) - \text{policy (partially observed)}$$
$$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) - \text{policy (fully observed)}$$

$\mathbf{a}_t$ – action
$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation

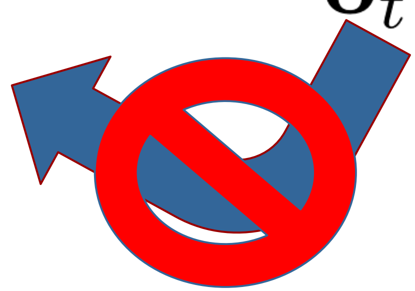
States and observations



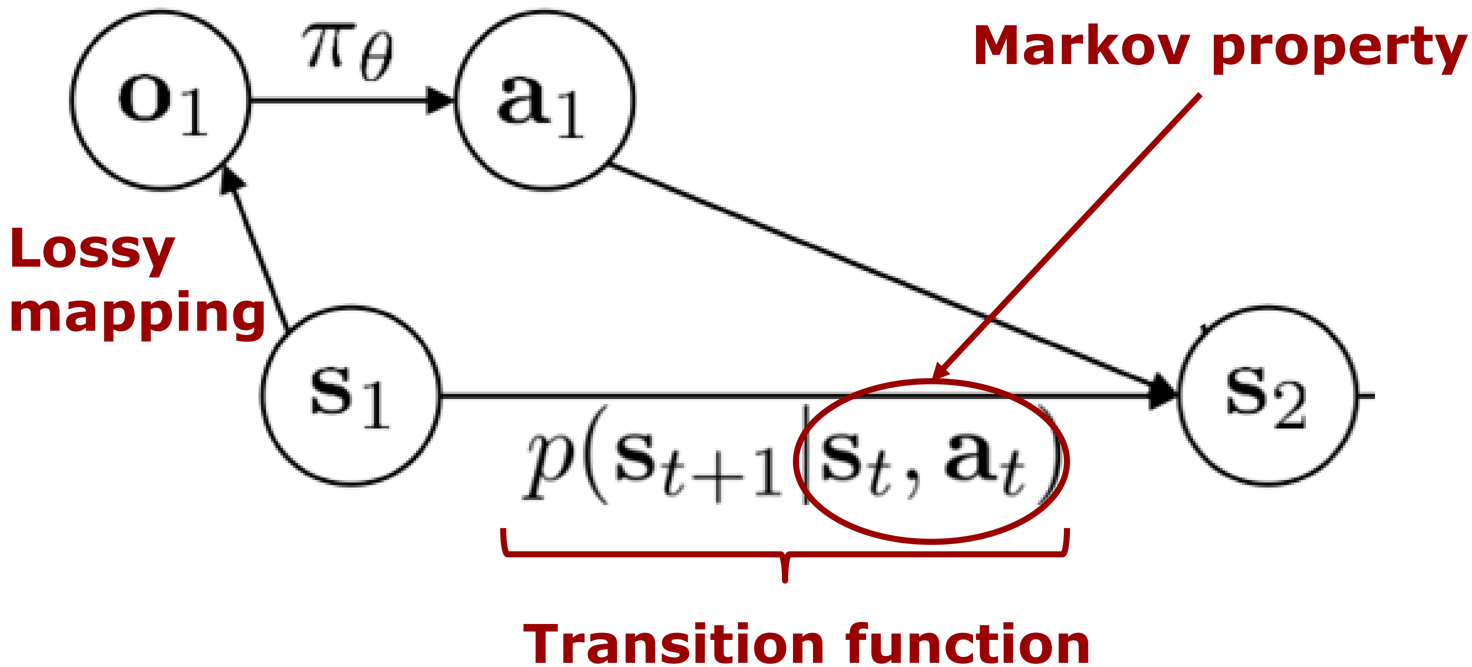


$\mathbf{s}_t$ – state

$\mathbf{o}_t$ – observation

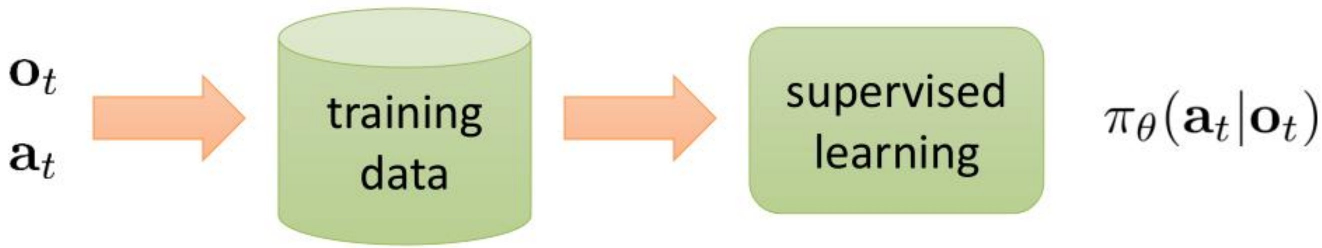


States and observations



[1]

Where to get labels?

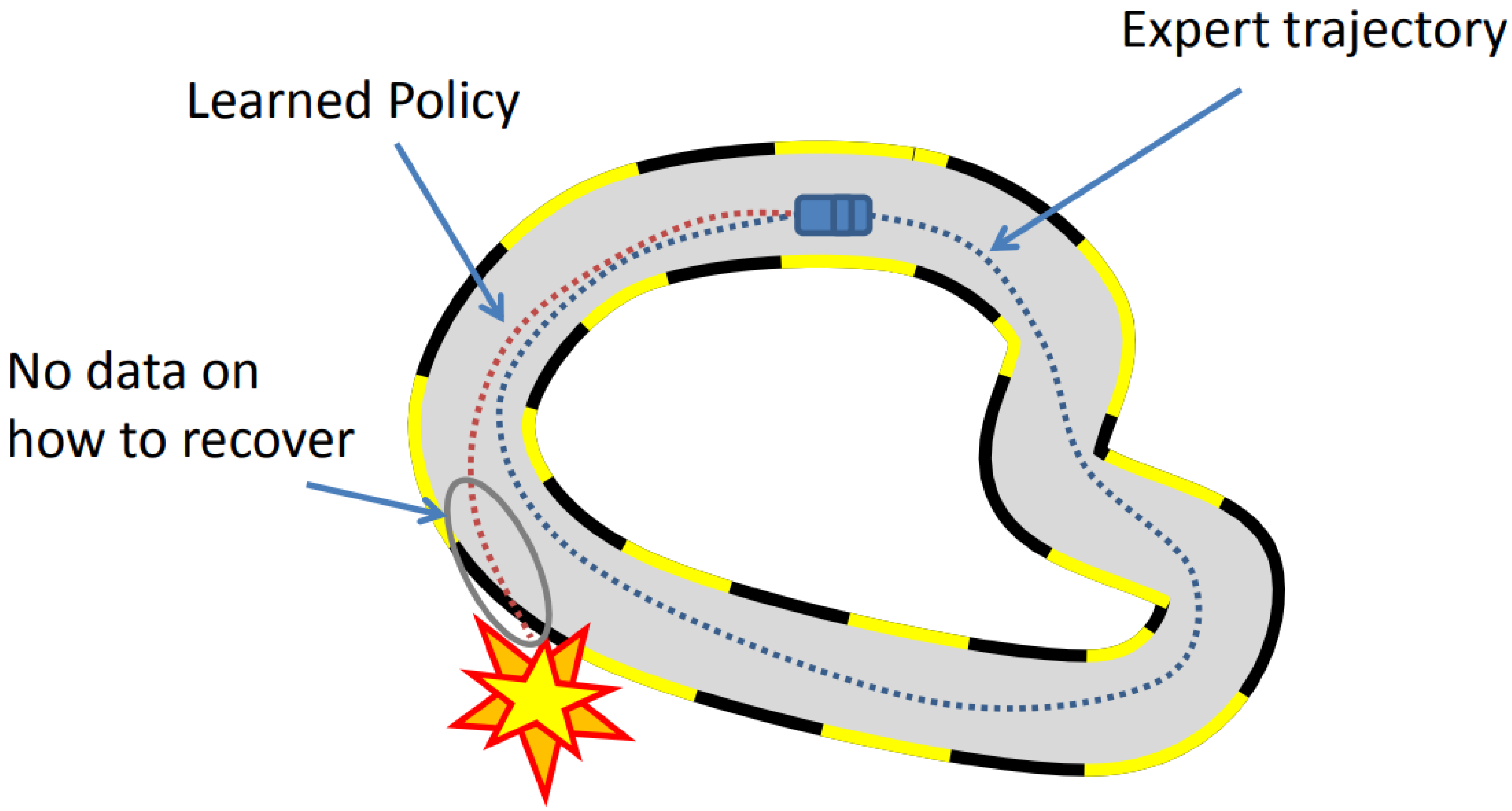


Behavior Cloning

Problems?

- Noisy labels (expert makes mistakes)
- Have to observe a lot of data from expert
- **Data distribution mismatch**:
  Observations that are not in training data set

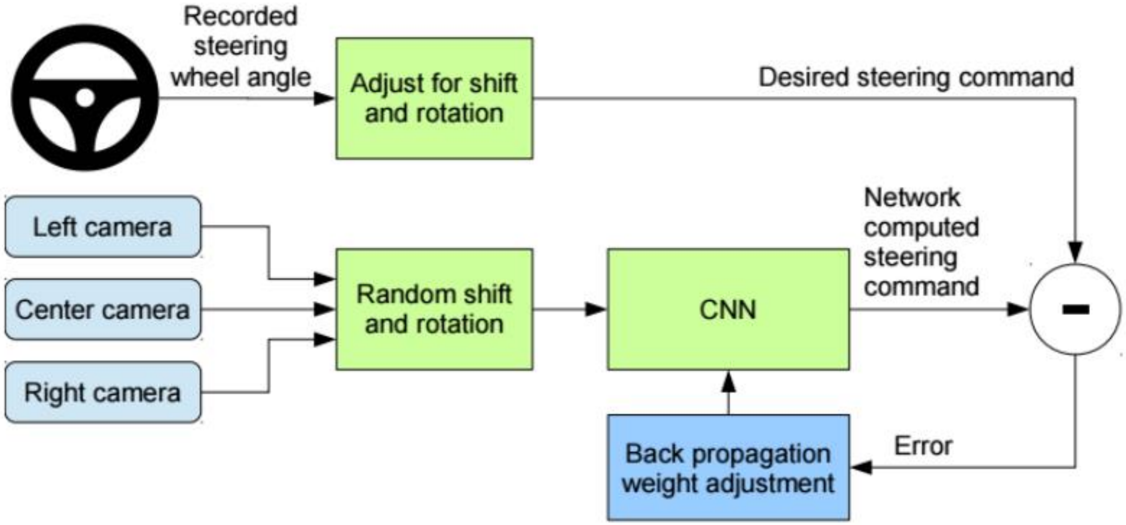
Data Distribution Mismatch

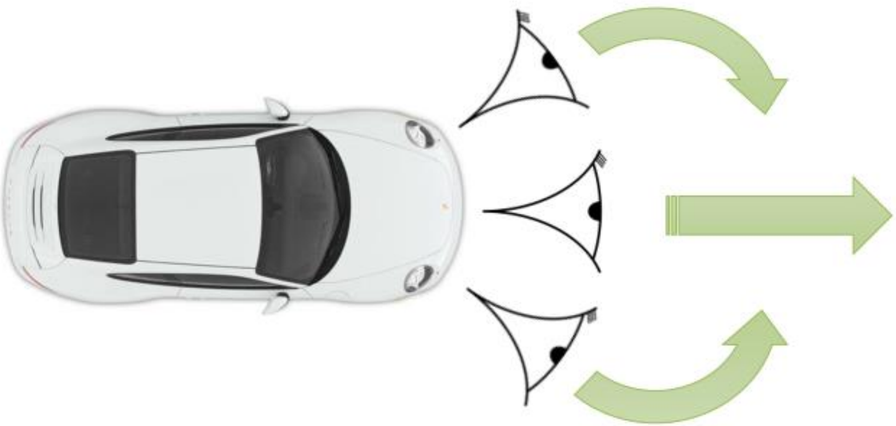


Imitation Driving

[Bojarski et al. 2016]

Imitation Driving - why does it work?





[Bojarski et al. 2016]

Dataset Aggregation

Problem: Distribution drift of $p_{\pi_\theta}(\mathbf{o}_t)$

Solution: Move $p_{\pi_\theta}(\mathbf{o}_t)$ closer to $p_{data}(\mathbf{o}_t)$

How? $\rightarrow$ Add samples of $p_{\pi_\theta}(\mathbf{o}_t)$ with annotations from expert to dataset

DAgger algorithm (Dataset Aggregation)

Train $p_{\pi_\theta}(\mathbf{o}_t)$ from expert data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, ..., \mathbf{o}_N, \mathbf{a}_N\}$

Run $p_{\pi_\theta}(\mathbf{o}_t)$ to get new data $\mathcal{D}_\pi = \{\mathbf{o}_1, ..., \mathbf{o}_M\}$

Ask **expert** to label $\mathcal{D}_\pi$ with actions

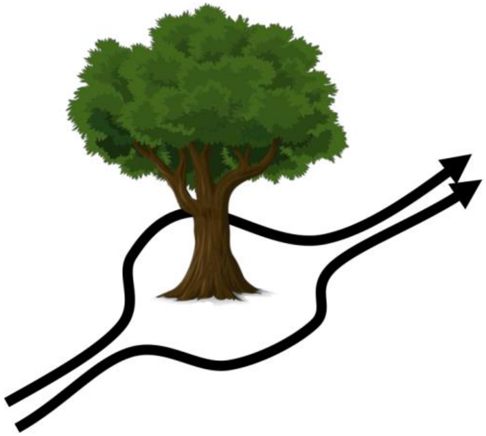
Aggregate $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

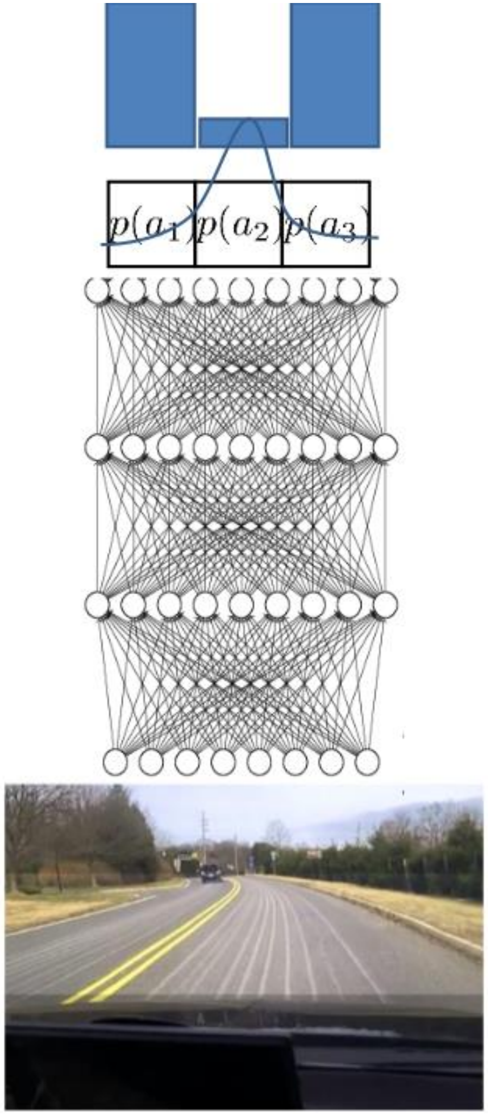
Failing to fit the expert

Reasons for failing:

- **Multimodal** expert behavior
- **Non-Markovian** expert behavior

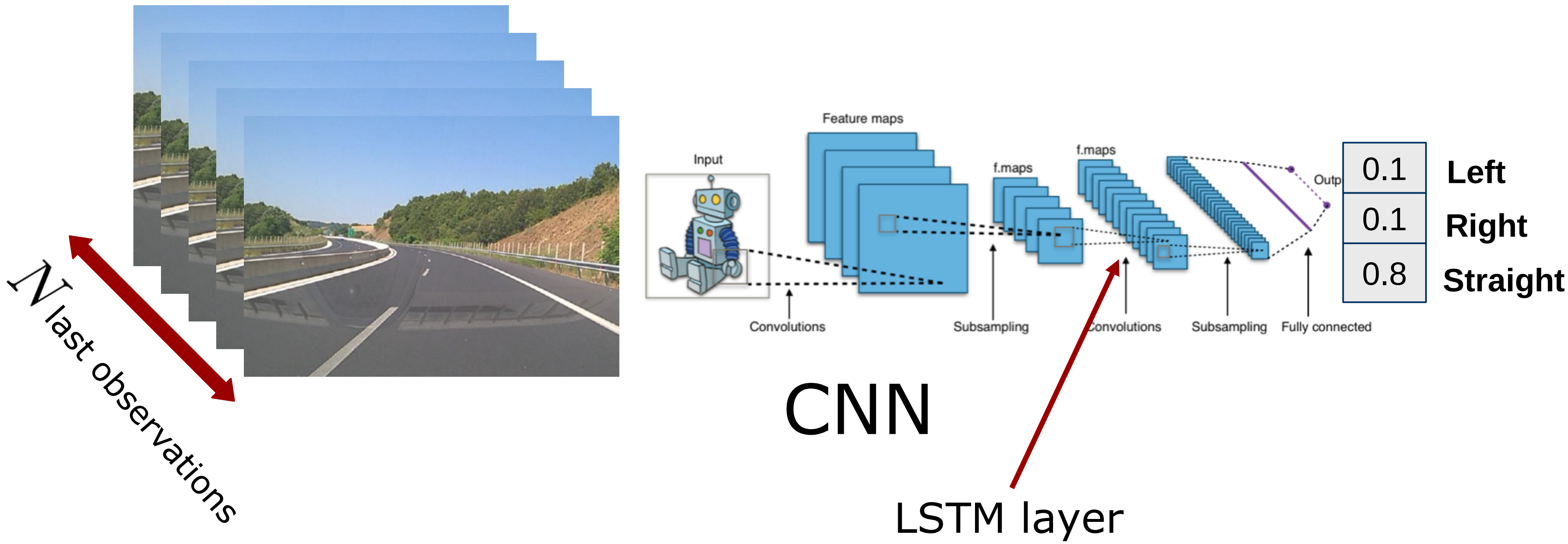
Multimodal expert behavior





[1]

Non-Markovian expert behavior



DAgger

Problems:

- Repeatedly **query expert**
- Execute an unsafe/**partially trained policy**

Non-human experts

- For training: learn classifier from expert brute force algorithm (i.e. exhaustive search)
- Query expert algorithm using DAgger
- For inference: use learned policy instead of slow expert

Cost function for Imitation

Reward r:

$$r(\mathbf{s}, \mathbf{a}) = \log p(\mathbf{a} = \pi^*(\mathbf{s})|\mathbf{s})$$

Loss:

$$\text{loss}(\mathbf{s}, \mathbf{a}) = -r(\mathbf{s}, \mathbf{a})$$

(cross-entropy loss)

References

- **[1]** UC Berkeley Deep RL course by Sergey Levine, lecture 1
- **[2]** A. Giusti et al.: A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots (2016)
- **[3]** M. Bojarski et al.: End to End Learning for Self-Driving Cars (2016)

Deep Reinforcement Learning

Niklas Wetzel

Outline

- Recap reinforcement learning (RL) concepts
- Value function approximation
- Deep Q-Networks

Components of RL: Recap

- Markov Decision Process (MDP)
- Value-functions, action-value functions
- Policy estimation (prediction), policy improvement, policy iteration
- Monte-Carlo estimation, temporal difference (TD) learning

Recap: Markov Decision Process

- Defined via $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$
  - state space, action space, dynamics, rewards, discount factor
  - Dynamics function defined by probabilities

$$p(s', r | s, a) := \mathbb{P}(S_{t+1} = s', R_t = r | S_t = s, A_t = a)$$

  - Notation: We use $\mathbb{P}_\pi, \mathbb{E}_\pi$, if $A_t \sim \pi$ for all t
- Goal: To maximize the expected return

Markov Decision Process Terminology

- Policy: $\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$
- Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$
- State-value function: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$
- Action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

How to estimate the value functions

- Monte Carlo: Average over samples (unbiased)

$$\mathbb{E}_\pi[G_t | S_t = s, A_t = a] \approx \frac{1}{N} \sum_{n=1}^{N} G_t^{(n)}$$

- Bootstrapping via TD targets (biased)

$$\mathbb{E}_\pi[G_t | S_t = s] \approx R_{t+1} + \gamma v_\pi(S_{t+1})$$

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve large problems, e.g.

- Backgammon: 10^{20} states
- Chess: 10^{47} states
- Go: 10^{170} states
- Helicopter: continuous state space

How can we scale up methods for prediction and control?

Value Function Problems

- Till now value function is treated like lookup table
  - Every state has an entry $v_\pi(s)$
  - Or every state-action pair (s,a) has an entry $q_\pi(s,a)$
- Problem with large MDPs:
  - Too many states and/or actions to store in memory
  - Too slow to learn the value of each states

Value Function Approximation

Solution for large MDPs:

- Estimate value functions with function approximation
- Generalize from seen states to unseen states

$$V_\phi(s) \approx v_\pi(s)$$

$$Q_\phi(s, a) \approx q_\pi(s, a)$$

- Update parameters ϕ using MC or TD learning

Function Approximator Types?

We choose ***neural networks*** as function approximators, but other choices are possible, e.g.

- Linear combination of features
- Decision trees
- Nearest neighbour methods
- Fourier / wavelet bases
- …

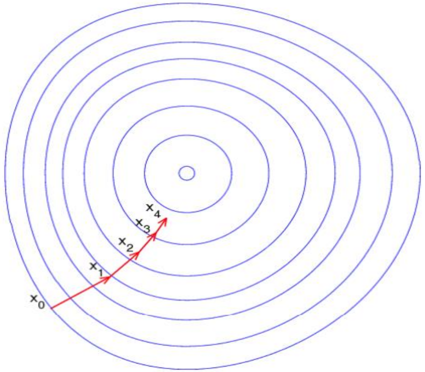
We require a training method that is suitable for *non-stationary*, *non-i.i.d.* data

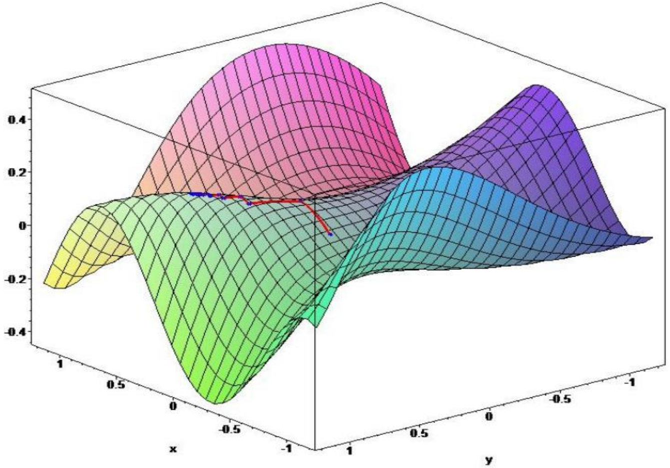
Gradient Descent

- Task: Find a local minimum of differentiable function J(ϕ)
- Adjust ϕ in direction of the negative gradient

$$\Delta\phi = -\ \alpha\ 0.5\mathrm{grad}(J),$$

where α is the step-size





[Source: UCL Course on RL, D. Silver]

Value Function Approx. by GD

- Goal: find parameters ϕ minimizing the mean-squared error between approximate value function $V_\phi(s)$ and true value function $v_\pi(s)$

$$J(\phi) = \mathbb{E}_\pi\left[\left(V_\phi(S) - v_\pi(S)\right)^2\right]$$

- Gradient descent (GD) finds a local minimum

$$\Delta\phi = \alpha\mathbb{E}_\pi\left[(v_\pi(S) - V_\phi(S))\nabla_\phi V_\phi(S)\right]$$

Value Function Approx. by SGD

- Stochastic gradient descent (SGD) samples states $S^{(1)},\ldots,S^{(N)}$ and estimates the gradient

$$\Delta\phi = \alpha \frac{1}{N} \sum_{n=1}^{N} \left[\left(V_\phi(S^{(n)}) - v_\pi(S^{(n)}) \right) \nabla_\phi V_\phi(S^{(n)}) \right]$$

- Expected update is equal to the full gradient update

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by a supervisor
- In RL there is no supervisor, only rewards
- In practise, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta\phi = (G_t - V_\phi(S_t))\nabla_\phi V_\phi(S_t)$$

  - For TD, the target is the TD target $R_{t+1} + \gamma V_\phi(S_{t+1})$

$$\Delta\phi = (R_{t+1} + \gamma V_\phi(S_{t+1}) - V_\phi(S_t))\nabla_\phi V_\phi(S_t)$$

MC with Value Function Approximation

- Return G_t is an unbiased sample of the true $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$(S_1, G_1), (S_2, G_2), ..., (S_T, G_T)$$

- MC guaranteed to converge to a local optimum using non-linear value function approximation

TD with Value Function Approximation

- TD target $R_{t+1} + \gamma V_\phi(S_{t+1})$ is a biased sample of the true $v_\pi(S_t)$
- Can still apply supervised learning to "training data" by substituting return samples with TD target samples

- Not guaranteed to converge to a local optimum using non-linear value function approximation

Action-Value Function Approximation

- Approximate the action-value function

$$Q_\phi(s,a) \approx q_\pi(s,a)$$

- Minimize mean-squared error between Q_ϕ(s,a) and the true action value function q_π(s,a)

$$J(\phi) = \mathbb{E}_\pi\left[(q_\pi(s,a) - Q_\phi(s,a))^2\right]$$

Action-Value Function Approx. by SGD

- Stochastic gradient descent (SGD) samples states $S^{(1)}, A^{(1)}, \ldots, S^{(n)}, A^{(n)}$ and estimates the gradient

$$\Delta\phi = \alpha \frac{1}{N}\sum_{n=1}^{N}[(Q_\phi(S^{(n)}, A^{(n)}) - q_\pi(S^{(n)}, A^{(n)}))\nabla_\phi Q_\phi(S^{(n)}, A^{(n)})]$$

- Expected update is equal to the full gradient update

TD-Control: Optimal Value Functions

- Optimal state-value function

$$v_*(s) := \max_\pi v_\pi(s)$$

- Optimal action-value function

$$q_*(s, a) := \max_\pi q_\pi(s, a)$$

- TD control goal: Estimate optimal value functions

Sarsa: On-policy TD-control

- Approximate q_π with Q_ϕ
- TD targets given by

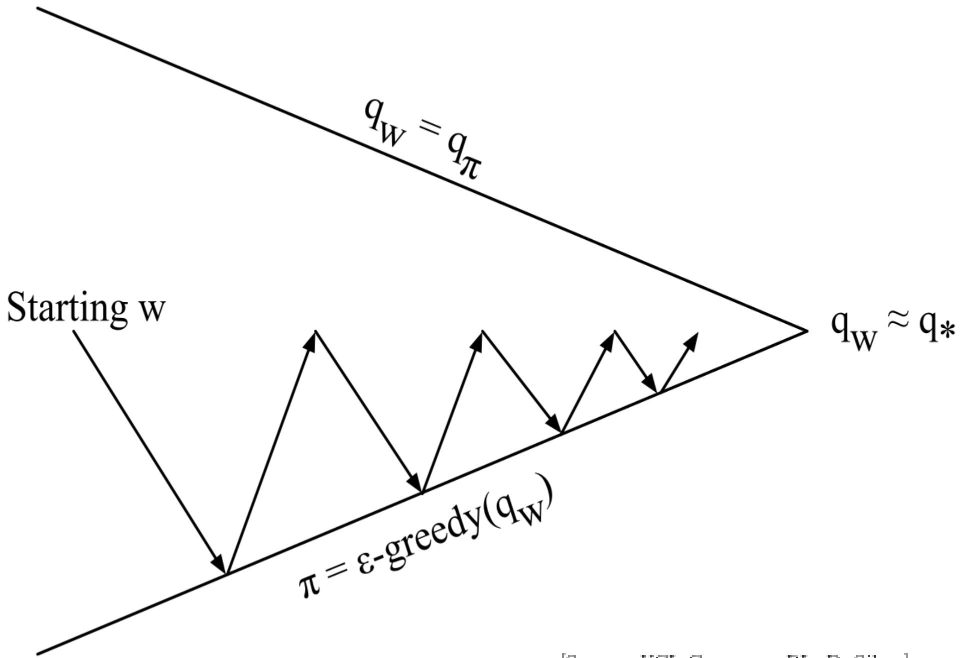
$$R_{t+1} + Q_\phi(S_{t+1}, A_{t+1})$$

- Policy improvement: Update behaviour policy π to be (ϵ-) greedy w.r.t. Q_ϕ

- Repeat till convergence: $\pi_k \rightarrow \pi_{k+1} \rightarrow \ldots \rightarrow \pi_*$

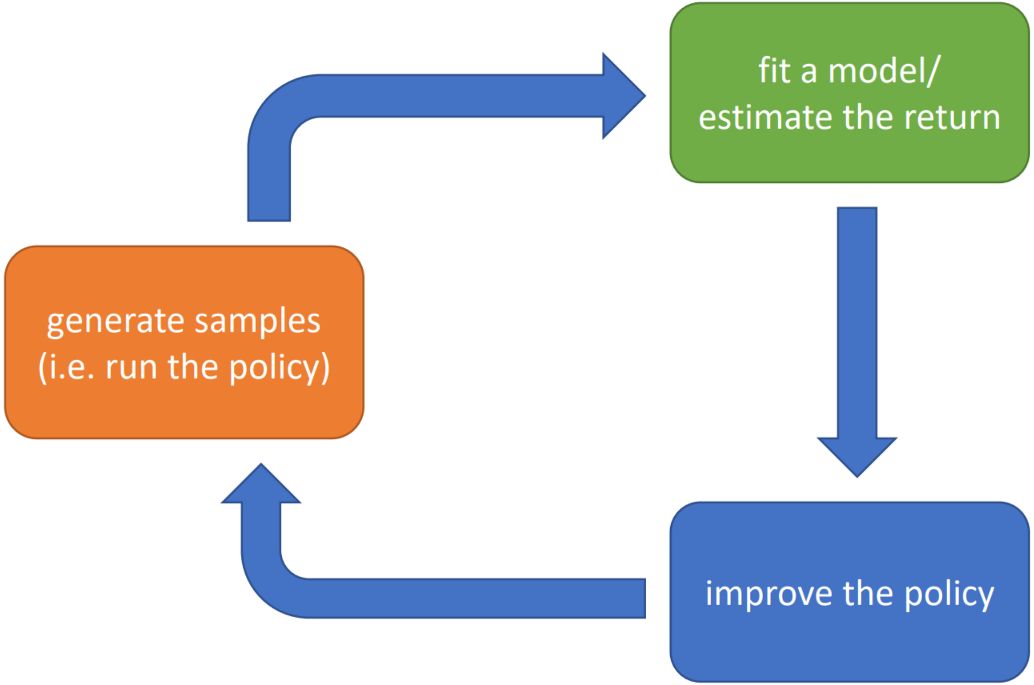
Control with Value Function Approximation

- Policy evaluation
  Approximate policy evaluation:
  $$Q_\phi \approx q_\pi$$
- Policy improvement
  Any, e.g. ϵ–greedy improvement



[Source: UCL Course on RL, D. Silver]

Policy iteration



[Source: CS 294-112: Deep RL, S. Levine]

Q-learning: Off-policy TD-control

- Directly approximate q_* with Q_ϕ
- TD target given by

$$R_{t+1} + \max_{a \in \mathcal{A}} Q_\phi(S_{t+1}, a)$$

- Advantage: off-policy

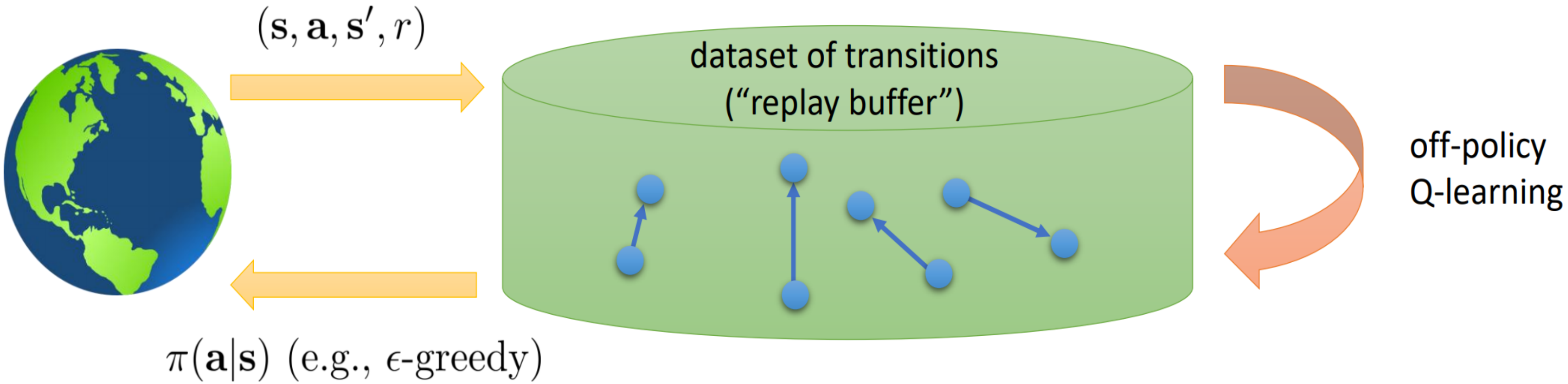
Q-learning with function approximator 1

- Q-iteration algorithm (online):
  1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
  2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
  3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

- Problem: Highly correlated states

Deep Q-learning

Idea: Decorrelate data with replay buffer



[Source: CS 294-112: Deep RL, S. Levine]

Q-learning with function approximator 2

- Q-iteration algorithm (offline):

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$

$K\times$

3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

- Careful: No SGD, since y_i depends on ϕ, but is 'viewed' as constant in 3.)

Q-learning with function approximator 3

- Idea: Fix TD targets for a given time period

- Q-iteration with replay buffer and target network:

  1. save target network parameters: $\phi' \leftarrow \phi$
  2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
  3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
  4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

  $N\times$

  $K\times$

- You will implement this in your RL exercise

Sources:

- CS 294-112: Deep Reinforcement Learning
  - Sergey Levine
- UCL Course on RL
  - David Silver
- Reinforcement Learning: An Introduction
  - R.Sutton, A.Barto