Deep Q-Learning from Demonstrations (DQfD)

Bryan Chan & Chandripal Budnarain

Markov Decision Process (MDP)

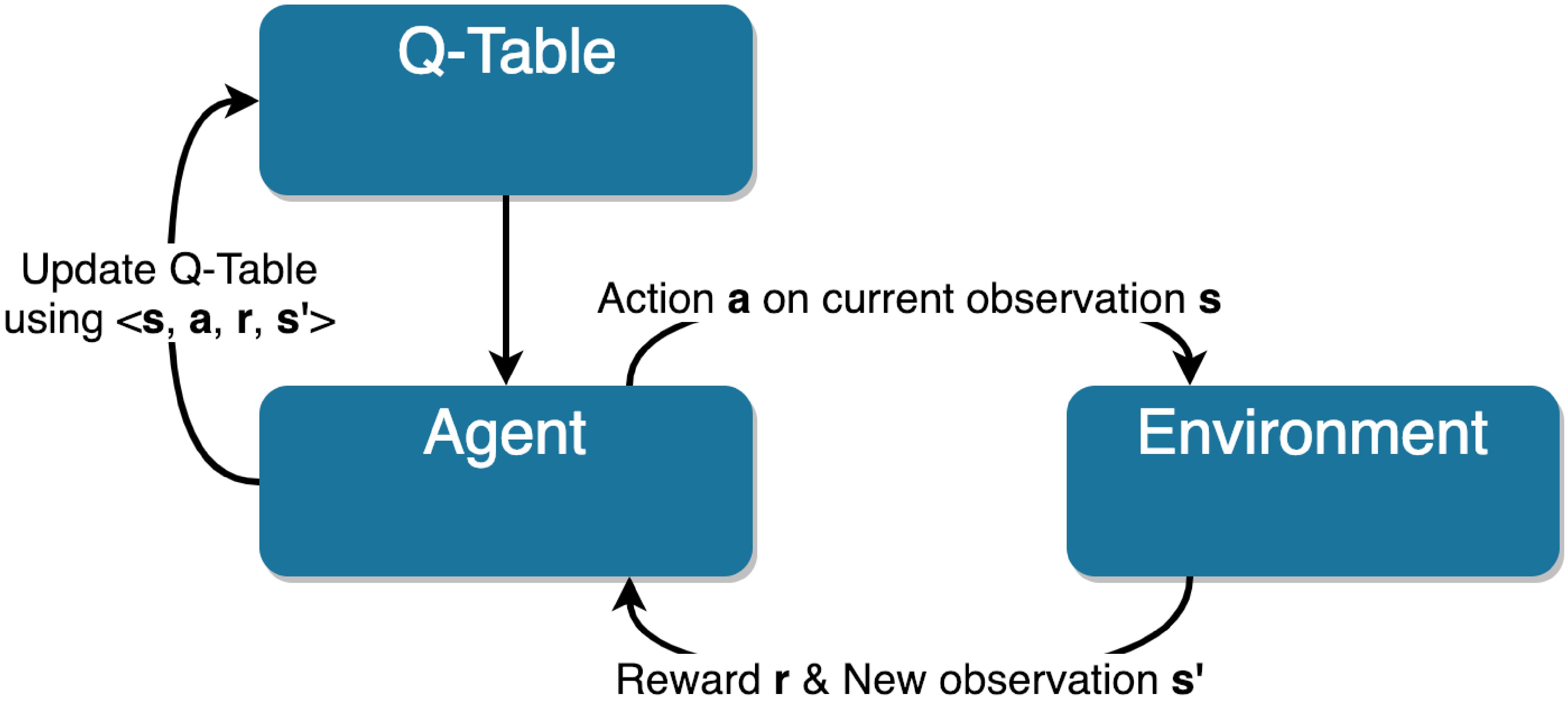
- A MDP is a tuple $\langle S, A, P, R, \gamma \rangle$
  - S: A finite set of states
  - A: A finite set of actions
  - P: A state transition function
  - R: A reward function
  - γ: Discount factor

- Want to find a policy $\pi: S \rightarrow A$ such that it maximizes the expected discounted total reward

Q-Function

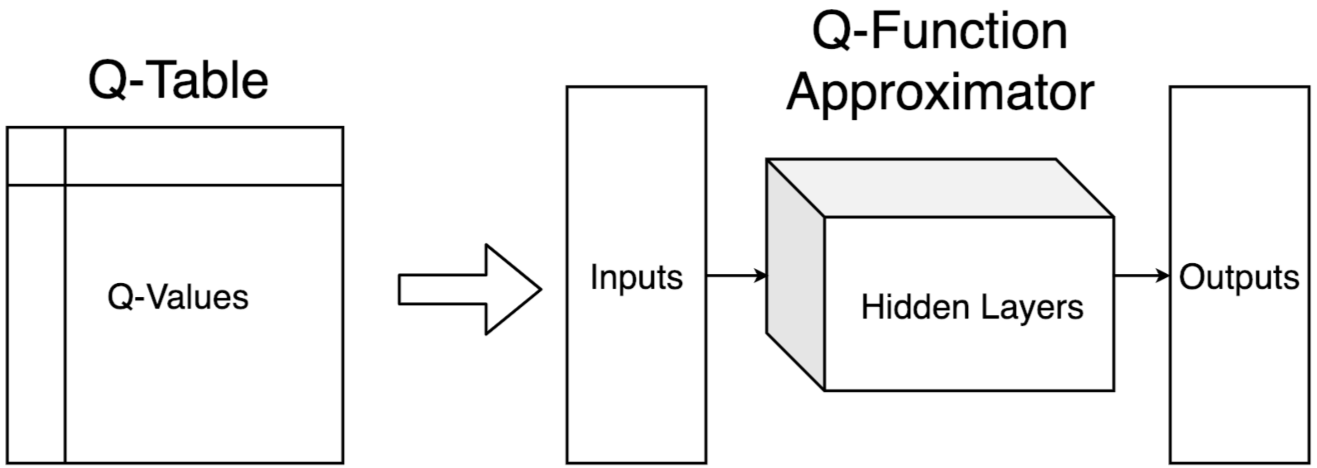
- The action-value Q-function $Q^{\pi}(s_t, a_t)$ is the expected return starting from state s_t, taking action a_t, and then following policy π
- $$\begin{aligned} Q^{\pi}(s_t, a_t) &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid s_t, a_t] \\ &= E_{s'}[R_{t+1} + \gamma Q^{\pi}(s', a') \mid s_t, a_t] \end{aligned}$$
- The optimal policy $\pi^*(s)$ can be obtained from optimal Q-function $\mathrm{argmax}_a Q^*(s, a)$

Q-Learning Algorithm

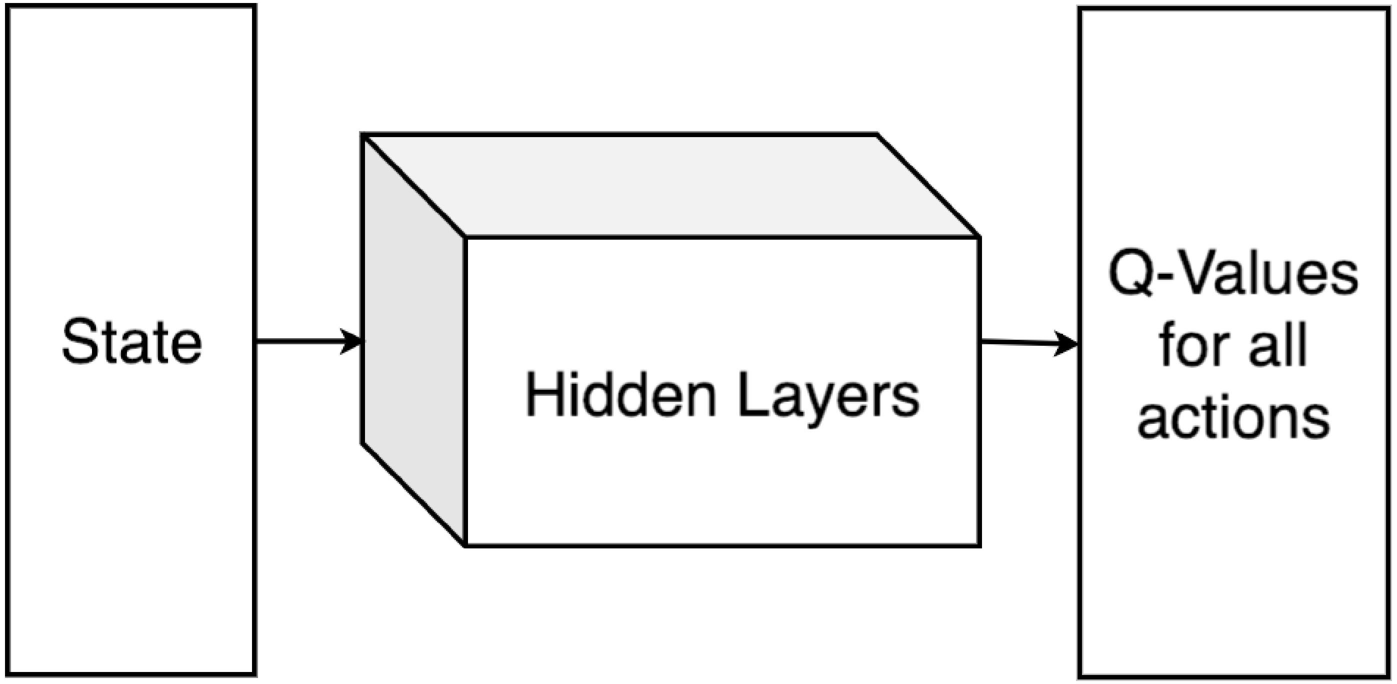


Deep Q-Network (DQN)

- State-action space might be too big for storing a Q-table!
- Idea: Replace Q-table with a neural network that approximates Q-values
- Deep Q-Network = Deep Learning + Q-Learning

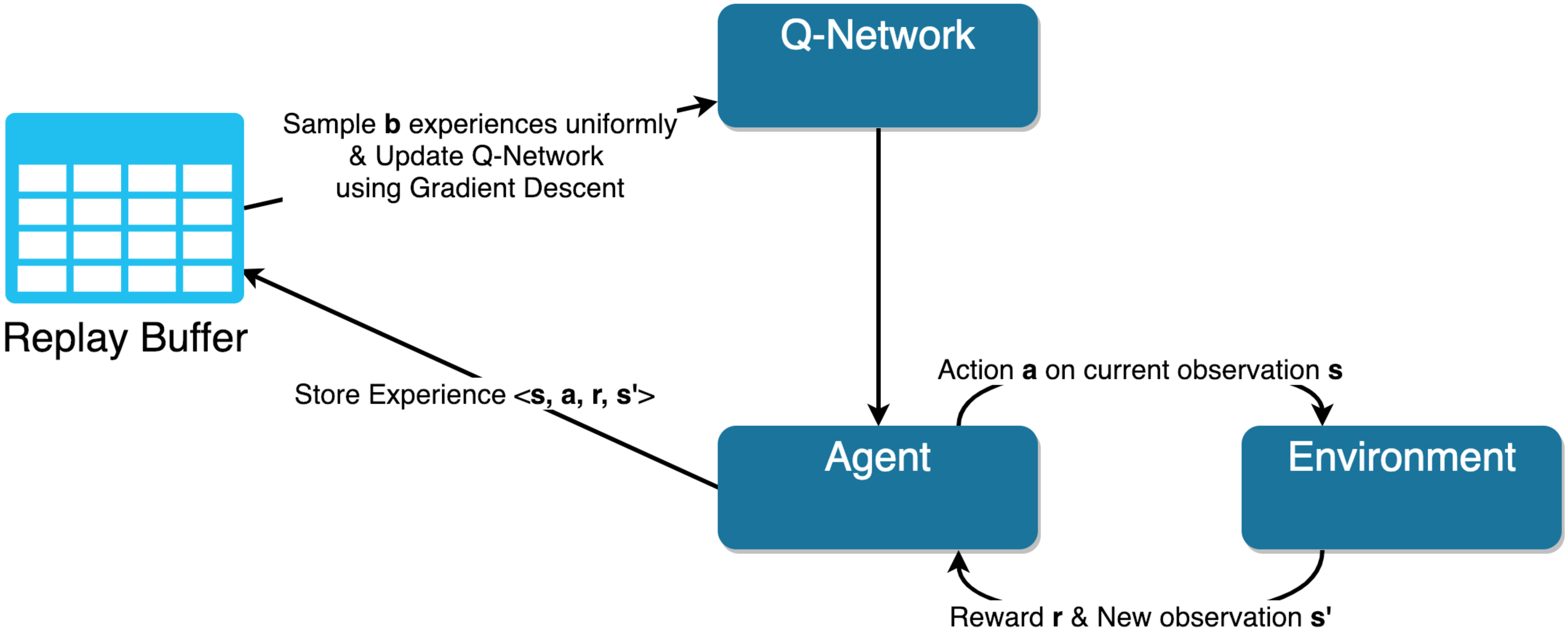


Q-Function Approximator



- $Loss = [(R(s,a) + \gamma \max_{a \in A} Q(s', a; \theta)) \ - Q(s, a; \theta)]^2$

DQN Algorithm



How to Combine Demonstration Data with DQN?

Loss Function

- Recall that the loss function for Q-Learning is:
$$J_{DQN}(Q) = [(R(s,a) + \gamma \max_a Q(s', a; \theta)) - Q(s,a;\theta)]^2$$
- Given demonstration data, we want the agent to learn from it
- **Issue:** Demonstration data only covers a small subset of the state space and does not consider a lot of actions
- **Issue:** Many (ungrounded) values are not realistic and the Q-Network would propagate these values

Supervised Large Margin Classification Loss

- Push the values of other actions to be at least a margin lower than the demonstrator's action
- The loss function:

$$J_E(Q) = \max_{a \in A}[Q(s,a) + l(a,a_E)] - Q(s,a_E),$$

where $l(a,a_E)$ is a margin function that is 0 when $a = a_E$ and some positive value otherwise, and a_E is the demonstrator's action

- In this paper, $l(a,a_E) = 0$ if $a = a_E$, and 0.8 otherwise

New Loss Function

- $J(Q) = J_{DQN}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)$,

where λ's control the weighting between the losses, $J_n(Q)$ is the n-step TD-loss, and $J_{L2}(Q)$ is the L2 regularization loss

- There is a trade off between following demonstration data and finding optimal Q-values

Prioritized Experience Replay

- In DQN, we sample experiences from the replay buffer uniformly
- **Issue:** We tend to learn better when there is a big difference between what we imagine and the actual outcome
- For example, we focus on mistakes and learn from them!
- We can prioritized what we sample instead – By looking at the latest TD-error: $\delta = \underbrace{R(s,a) + \gamma \max_{a \in A} Q(s', a; \theta)}_{\text{“actual” outcome}} - \underbrace{Q(s, a; \theta)}_{\text{“estimated” outcome}}$

Prioritized Experience Replay

- Specifically, priority of experience i, $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$,

where $p_i = |\delta_i| + \epsilon$ is the absolute of last TD-error with some positive constant

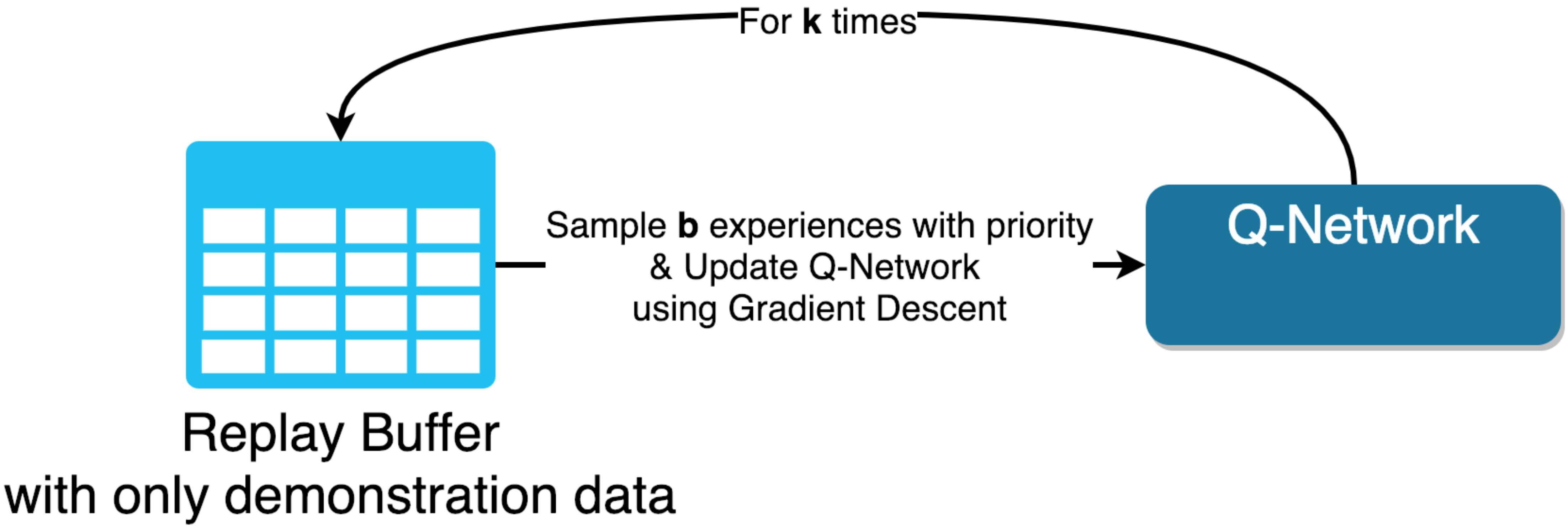
- **What is α?**
- α (hyperparameter) decides how much prioritization is used. If $\alpha = 0$, we are sampling uniformly
- **Issue:** Sampling with priority introduces bias and changes the distribution

Prioritized Experience Replay

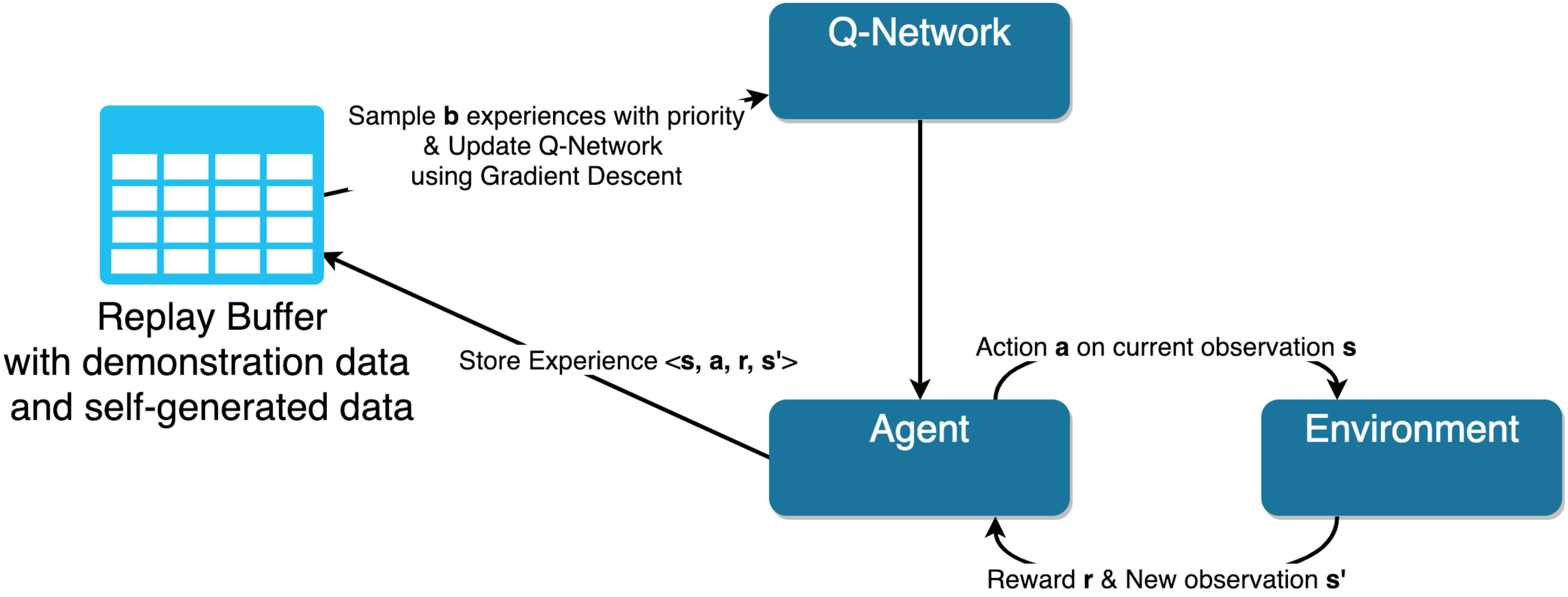
- **Solution:** Correct using weighted importance-sampling with weights $w_i = (\frac{1}{N}\frac{1}{P(i)})^{\beta}$, where N is number of samples
- **What is $\boldsymbol{\beta}$?**
- β (hyperparameter) decides how much we should compensate for the non-uniform probabilities $P(i)$. If $\beta = 1$, we fully compensate
- In general, α and β grows together as time goes on. The idea is that we first sample close to uniformly, then slowly sample with priority
- In this paper, $\alpha = 0.4$ and $\beta = 0.6$ (Fixed)

Deep Q-Learning from Demonstration (DQfD)

DQfD Pre-Training



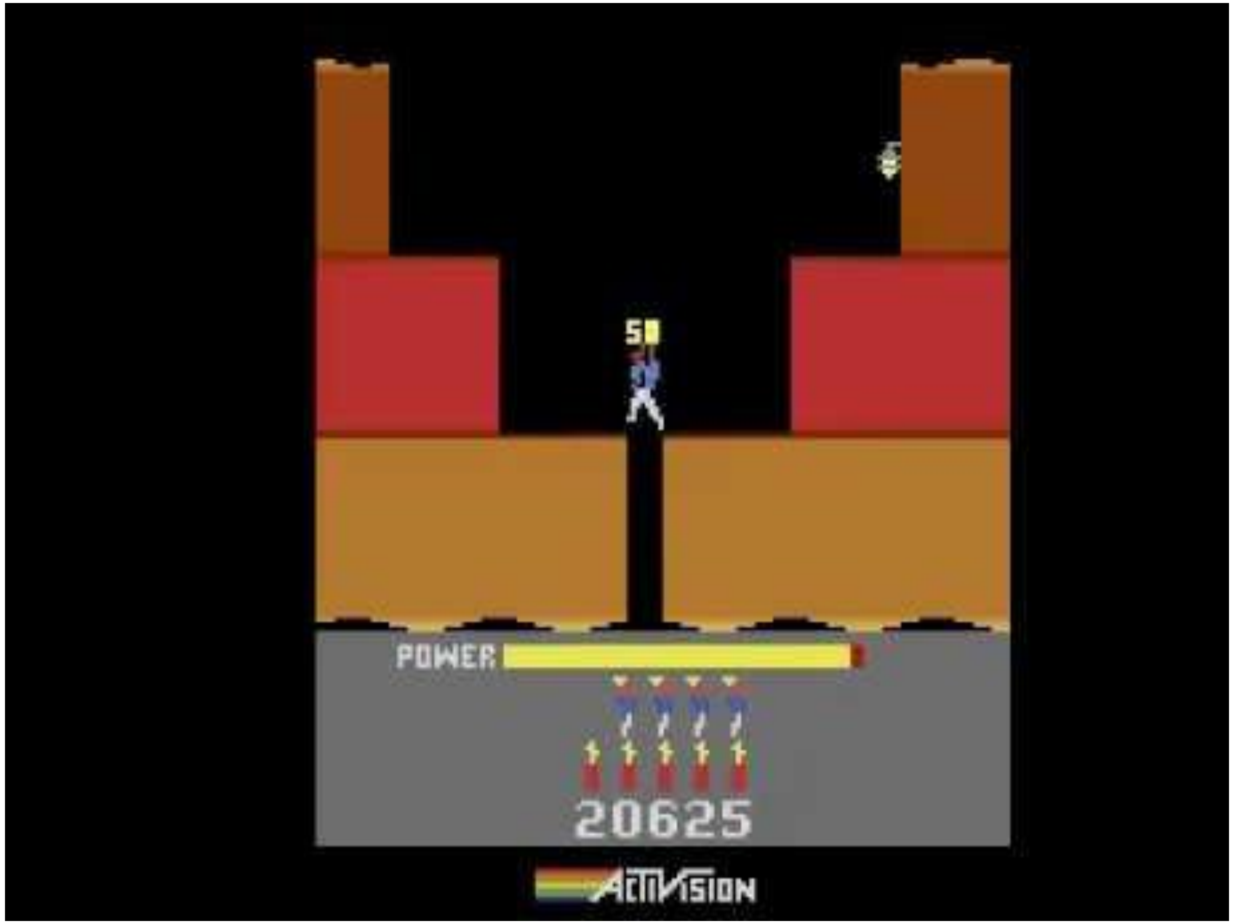
DQfD Post-Training



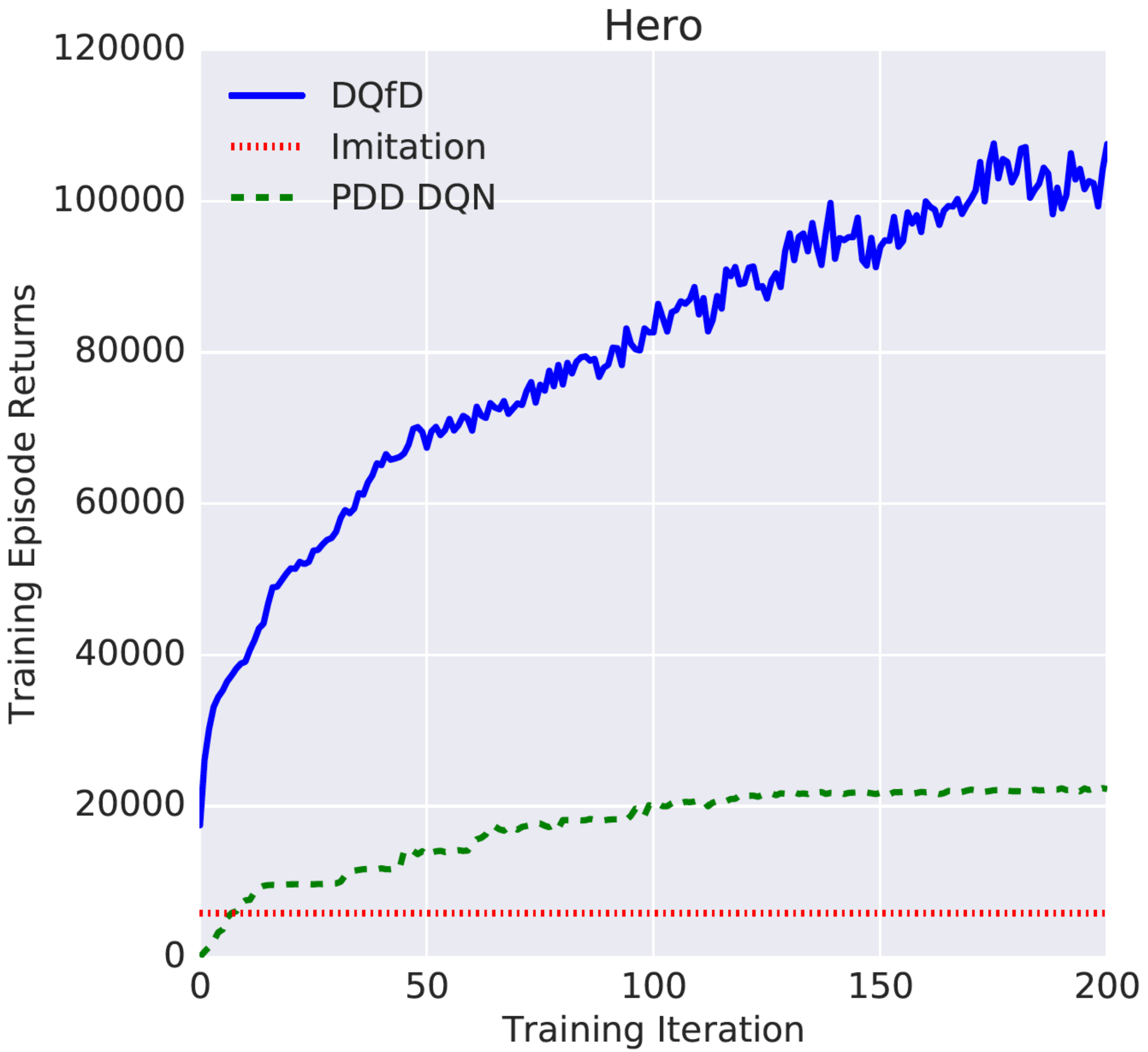
DQfD Replay Buffer Tweak

- We give more priority on demonstration data (by having a higher ϵ)
- In this paper, $\epsilon_a = 0.001$ (self-generated) and $\epsilon_d = 1.0$ (demonstration)
- **Problem:** What if the replay buffer is full?
- 1) We want to make sure the agent does not go too far from demonstrator unless some other action is optimal
  - **Keep demonstration data**
- 2) Old sampled experiences are out-of-date
  - **Remove oldest self-generated data**

Experiment

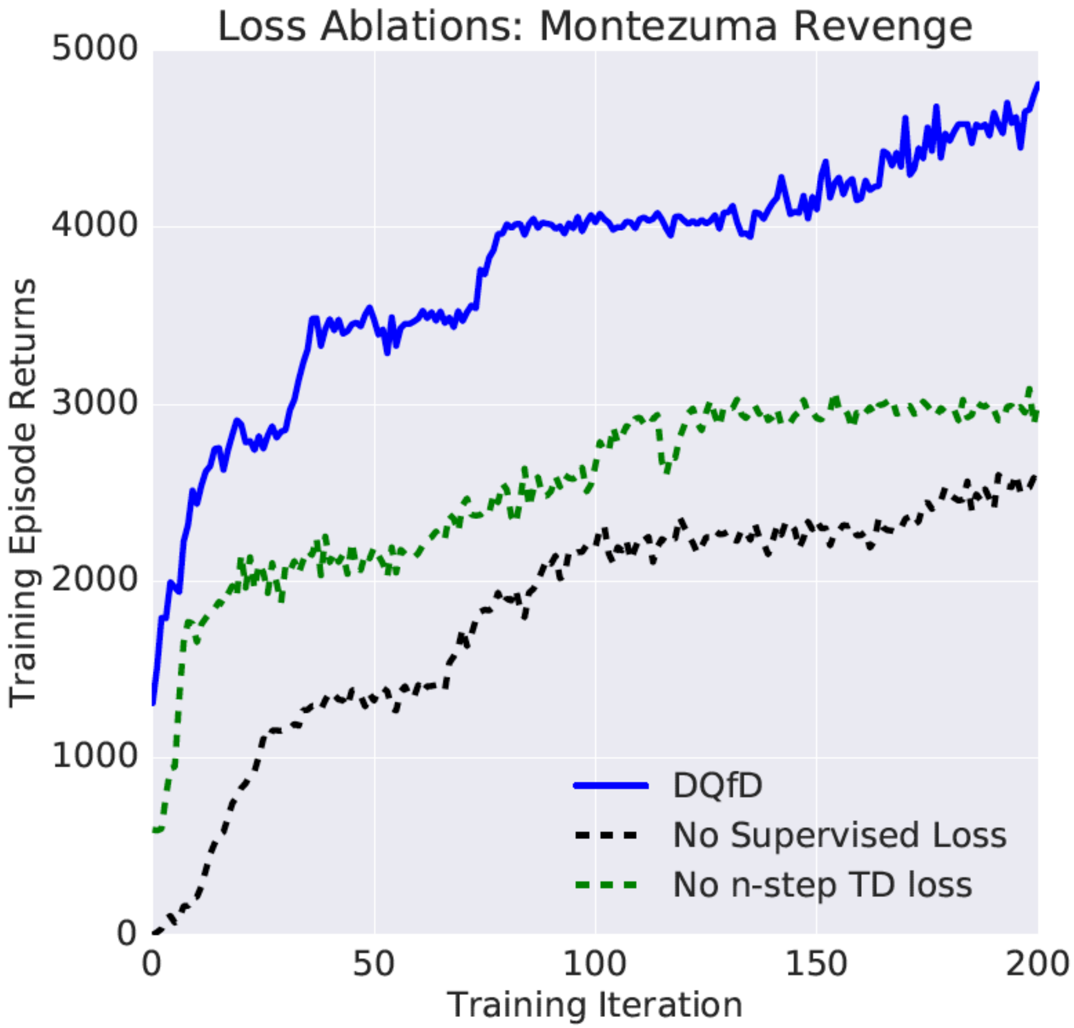


POWER
20625
ACTIVISION



Hero
DQfD
Imitation
PDD DQN
Training Episode Returns
Training Iteration
0
20000
40000
60000
80000
100000
120000
0
50
100
150
200

Removing Supervised Loss



Summary

- Improved initial performance in real system using demonstration data
- Accelerated learning by combining supervised large margin classification loss and traditional DQN loss
- Smartly utilizes demonstration data during post-training using prioritized experience replay

Limitations

- Does not explore continuous state-action space scenarios
- Similar to previous paper, algorithm does not explore hidden state humans might consider

AggreVaTe:

Reinforcement and Imitation Learning via Interactive No-Regret Learning

CSC 2621
Renato Ferreira Pinto Junior

Stéphane Ross & J. Andrew Bagnell (2014)

Pick one:

Pick one:

Main idea

- DAgger aims to minimize disagreement with expert

Main idea

- DAgger aims to minimize disagreement with expert
    - What if it’s easier to imitate the expert in an *unsafe* situation than in *safe* ones?

Main idea

- DAgger aims to minimize disagreement with expert
  - What if it's easier to imitate the expert in an *unsafe* situation than in *safe* ones?
  - No consideration of *cost-to-go* of learned policy

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm 3.1: DAGGER Algorithm.

Main idea

- DAgger aims to minimize disagreement with expert
  - What if it's easier to imitate the expert in an *unsafe* situation than in *safe* ones?
- Instead, AggreVaTe (*Aggregate Values to Imitate*):
  - Minimizes expert's *cost-to-go*
  - Provides *regret* (rather than *error*) guarantees

Regret

- In *hindsight*, how much better could I have performed?

$$\textbf{regret}(h_1, \ldots, h_T) = \frac{1}{N}\sum_{t=1}^{T} Cost(h_t, t) - \min_{h \in H} \frac{1}{N}\sum_{t=1}^{T} Cost(h, t)$$

Regret

- In *hindsight*, how much better could I have performed?

$$\textbf{regret}(h_1, \ldots, h_T) = \frac{1}{N}\sum_{t=1}^{T} Cost(h_t, t) - \min_{h \in H} \frac{1}{N}\sum_{t=1}^{T} Cost(h, t)$$

Limited information at each time t

Regret

- In *hindsight*, how much better could I have performed?

$$\textbf{regret}(h_1, \ldots, h_T) = \frac{1}{N}\sum_{t=1}^{T} Cost(h_t, t) - \min_{h \in H} \frac{1}{N}\sum_{t=1}^{T} Cost(h, t)$$

Limited information at each time t

- AggreVaTe minimizes **regret** with respect to **expert's cost**
- DAgger minimizes **loss** with respect to **expert's actions**

Algorithm

Algorithm 1 AGGREVATE: Imitation Learning with Cost-To-Go

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$ #Optionally mix in expert's own behavior.
  Collect m data points as follows:
  for $j = 1$ **to** m **do**
    Sample uniformly $t \in \{1, 2, \ldots, T\}$.
    Start new trajectory in some initial state drawn from initial state distribution
    Execute current policy π_i up to time $t - 1$.
    Execute some exploration action a_t in current state s_t at time t
    Execute expert from time $t + 1$ to T, and observe estimate of cost-to-go $\hat{Q}$ starting at time t
  end for
  Get dataset $\mathcal{D}_i = \{(s, t, a, \hat{Q})\}$ of states, times, actions, with expert's cost-to-go.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
    (Alternately: use any online learner on the data-sets $\mathcal{D}_i$ in sequence to get $\hat{\pi}_{i+1}$)
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm

Initialization {

Algorithm 1 AGGREVATE: Imitation Learning with Cost-To-Go

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$ #Optionally mix in expert's own behavior.
  Collect m data points as follows:
  for $j = 1$ **to** m **do**
    Sample uniformly $t \in \{1, 2, \ldots, T\}$.
    Start new trajectory in some initial state drawn from initial state distribution
    Execute current policy π_i up to time $t - 1$.
    Execute some exploration action a_t in current state s_t at time t
    Execute expert from time $t + 1$ to T, and observe estimate of cost-to-go $\hat{Q}$ starting at time t
  end for
  Get dataset $\mathcal{D}_i = \{(s, t, a, \hat{Q})\}$ of states, times, actions, with expert's cost-to-go.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
    (Alternately: use any online learner on the data-sets $\mathcal{D}_i$ in sequence to get $\hat{\pi}_{i+1}$)
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm

Algorithm 1 AGGREVATE: Imitation Learning with Cost-To-Go

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$ #Optionally mix in expert's own behavior.
  Collect m data points as follows:
  for $j = 1$ **to** m **do**
Own policy up to t
    Sample uniformly $t \in \{1, 2, \ldots, T\}$.
    Start new trajectory in some initial state drawn from initial state distribution
    Execute current policy π_i up to time $t - 1$. → **similar to DAgger**
Exploration action → Execute some exploration action a_t in current state s_t at time t
Expert concludes → Execute expert from time $t + 1$ to T, and observe estimate of cost-to-go $\hat{Q}$ starting at time t
  end for → **expert cost-to-go estimate**
  Get dataset $\mathcal{D}_i = \{(s, t, a, \hat{Q})\}$ of states, times, actions, with expert's cost-to-go.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
    (*Alternately: use any online learner on the data-sets* $\mathcal{D}_i$ *in sequence to get* $\hat{\pi}_{i+1}$)
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm

Algorithm 1 AGGREVATE: Imitation Learning with Cost-To-Go

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$ #Optionally mix in expert's own behavior.
  Collect m data points as follows:
  for $j = 1$ **to** m **do**
    Sample uniformly $t \in \{1, 2, \ldots, T\}$.
    Start new trajectory in some initial state drawn from initial state distribution
    Execute current policy π_i up to time $t - 1$. → similar to DAgger
    Execute some exploration action a_t in current state s_t at time t
    Execute expert from time $t + 1$ to T, and observe estimate of cost-to-go $\hat{Q}$ starting at time t
  end for → expert cost-to-go estimate
New data point: Get dataset $\mathcal{D}_i = \{(s, t, a, \hat{Q})\}$ of states, times, actions, with expert's cost-to-go.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
Train on dataset → Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$ → minimize total cost
   (Alternately: use any online learner on the data-sets $\mathcal{D}_i$ in sequence to get $\hat{\pi}_{i+1}$)
end for
Return best $\hat{\pi}_i$ on validation.

Analysis

- Classification regret: best in policy class compared to expert

$$\epsilon_{\text{class}} = \min_{\pi\in\Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t\sim U(1:T), s\sim d^t_{\pi_i}} [Q^*_{T-t+1}(s,a) - \min_a Q^*_{T-t+1}(s,a)]$$

Analysis

- Classification regret: best in policy class compared to expert

$$\epsilon_{\text{class}} = \min_{\pi\in\Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t\sim U(1:T), s\sim d_{\pi_i}^t} [Q^*_{T-t+1}(s,a) - \min_a Q^*_{T-t+1}(s,a)]$$

Recall:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \lambda \max_{a'} Q^*(s',a')|s,a]$$

Analysis

- Classification regret: best in policy class compared to expert

$$\epsilon_{\text{class}} = \min_{\pi\in\Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t\sim U(1:T), s\sim d_{\pi_i}^t} [Q^*_{T-t+1}(s,a) - \min_a Q^*_{T-t+1}(s,a)]$$

- Online learning regret: learned policies compared to best in policy class

$$\epsilon_{\text{regret}} = \frac{1}{N} [\sum_{i=1}^{N} \ell_i(\hat{\pi}_i) - \min_{\pi\in\Pi} \sum_{i=1}^{N} \ell_i(\pi)]$$

$$\ell_i(\pi) = \mathbb{E}_{t\sim U(1:T), s\sim d_{\pi_i}^t} [Q^*_{T-t+1}(s,\pi)]$$

Analysis

- Classification regret: best in policy class compared to expert

$$\epsilon_{\text{class}} = \min_{\pi\in\Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t\sim U(1:T), s\sim d^t_{\pi_i}} [Q^*_{T-t+1}(s,a) - \min_a Q^*_{T-t+1}(s,a)]$$

- Online learning regret: learned policies compared to best in policy class

$$\epsilon_{\text{regret}} = \frac{1}{N}[\sum_{i=1}^{N} \ell_i(\hat{\pi}_i) - \min_{\pi\in\Pi} \sum_{i=1}^{N} \ell_i(\pi)]$$

$$\ell_i(\pi) = \mathbb{E}_{t\sim U(1:T), s\sim d^t_{\pi_i}} [Q^*_{T-t+1}(s,\pi)]$$

- Guarantee:

$$J(\hat{\pi}) \leq J(\overline{\pi}) \leq J(\pi^*) + T[\epsilon_{class} + \epsilon_{regret}] + O\left(\frac{Q_{\max} T \log T}{\alpha N}\right)$$

Analysis

- Classification regret: best in policy class compared to expert

$$\epsilon_{\text{class}} = \min_{\pi\in\Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t\sim U(1:T), s\sim d_{\pi_i}^t} [Q^*_{T-t+1}(s,a) - \min_a Q^*_{T-t+1}(s,a)]$$

- Online learning regret: learned policies compared to best in policy class

$$\epsilon_{\text{regret}} = \frac{1}{N}[\sum_{i=1}^{N} \ell_i(\hat{\pi}_i) - \min_{\pi\in\Pi} \sum_{i=1}^{N} \ell_i(\pi)]$$

$$\ell_i(\pi) = \mathbb{E}_{t\sim U(1:T), s\sim d_{\pi_i}^t} [Q^*_{T-t+1}(s,\pi)]$$

- Guarantee:

$$J(\hat{\pi}) \leq J(\overline{\pi}) \leq \boxed{J(\pi^*)} + \boxed{T[\epsilon_{class} + \epsilon_{regret}]} + O\left(\frac{Q_{\max} T \log T}{\boxed{\alpha N}}\right)$$

Analysis

- If no-regret online algorithm is used to pick policies:

$$\lim_{N \to \infty} J(\overline{\pi}) \leq J(\pi^*) + T\epsilon_{class}$$

Analysis

- If no-regret online algorithm is used to pick policies:

$$\lim_{N \to \infty} J(\overline{\pi}) \leq J(\pi^*) + T\epsilon_{class}$$

- Can use online gradient descent descent

Conclusion

- Optimizes for *cost-to-go* rather than naive imitation
  - Prefer actions in which it’s possible to act optimally
  - Imitate expert *toward favourable situations*

Conclusion

- Optimizes for *cost-to-go* rather than naive imitation
  - Prefer actions in which it's possible to act optimally
  - Imitate expert *toward favourable situations*
- Limitations:
  - Expensive data collection (one data point per trajectory!)
  - Requires policy class to contain good policy compared to expert
  - Empirical evidence?

Agile Autonomous Driving using End-to-End Deep Imitation Learning

Yunpeng Pan*, Ching-An Cheng*, Kamil Saigol*, Keuntaek Lee†, Xinyan Yan*,
Evangelos A. Theodorou*, and Byron Boots*
*Institute for Robotics and Intelligent Machines, †School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
{ypan37,cacheng,kamilsaigol,keuntaek.lee,xyan43}@gatech.edu
evangelos.theodorou@gatech.edu, bboots@cc.gatech.edu

Presented by David Acuna and Brenna Li

Problem Formulation

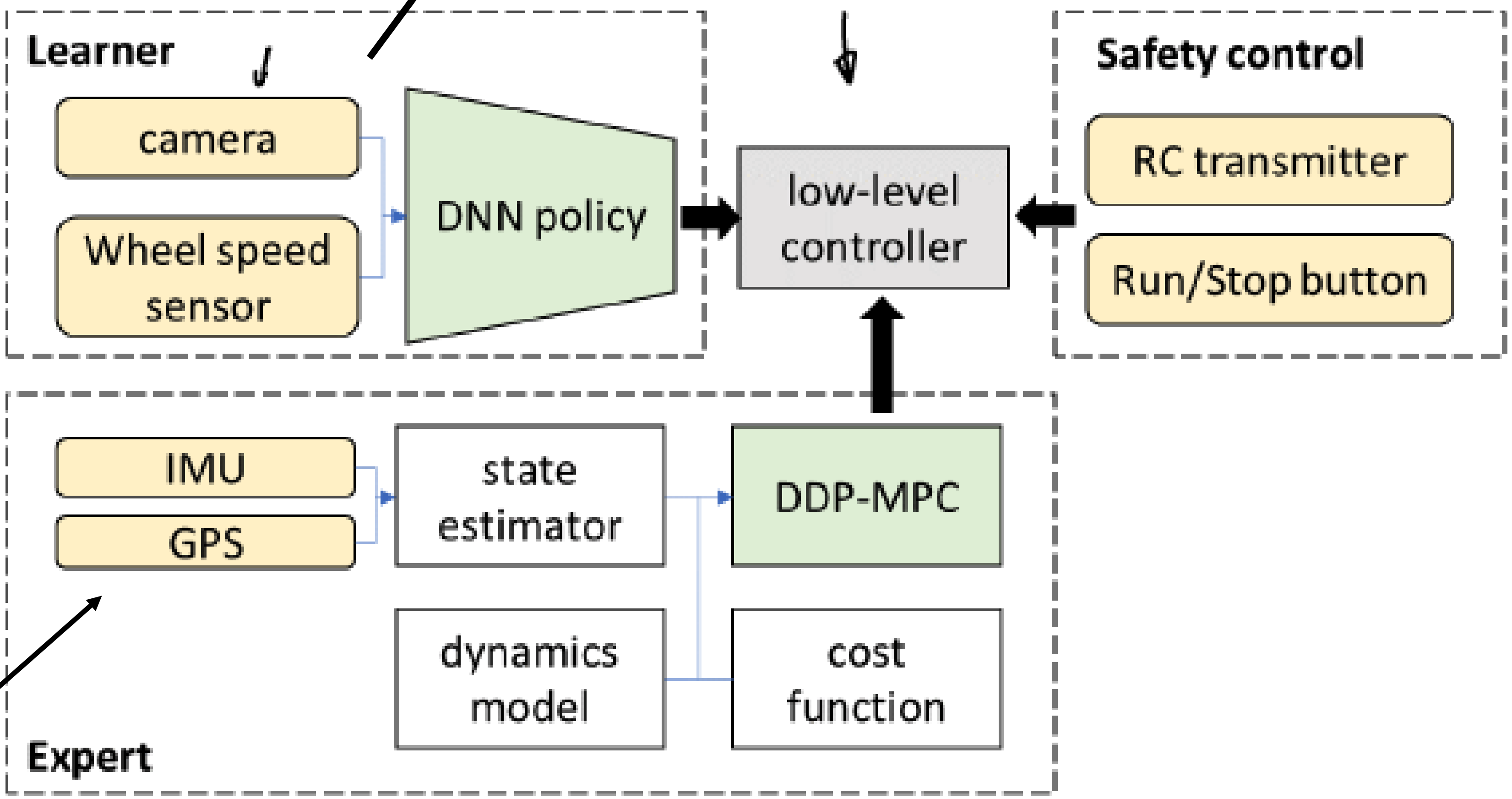
training/test track

off-the-road real-word scenario.
high-speed is a must

Auto-Rally car

Problem Formulation

cheap sensors. ~ $500
NN learns from raw images and speed sensor



expensive sensors
model predictive control ~ $6,000

IMU=Inertial Measurement Units
GPS=Global Positioning System

Formulation

$$\min_\pi J(\pi), \quad J(\pi) := \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T-1} c(s_t, a_t)\right],$$

- needs to account for high-speed
- involves a physical robot

state, action, observation

$$d_\pi(s,t) \;=\; \frac{1}{T} d_\pi^t(s)$$

$$J(\pi) = J(\pi') + \mathbb{E}_{s,t\sim d_\pi}\mathbb{E}_{a\sim\pi_s}[A_{\pi'}^t(s,a)]$$

$$A_{\pi'}^t(s,a) \;=\; Q_{\pi'}^t(s,a) - V_{\pi'}^t(s)$$

expected reward of this state

expected reward of taking this action

Formulation

Hard to solve

$$\min_\pi J(\pi), \quad J(\pi) := \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T-1} c(s_t, a_t)\right],$$

- needs to account for high-speed
- involves a physical robot

$$J(\pi) = J(\pi') + \mathbb{E}_{s,t\sim d_\pi}\mathbb{E}_{a\sim\pi_s}[A^t_{\pi'}(s,a)]$$

expert

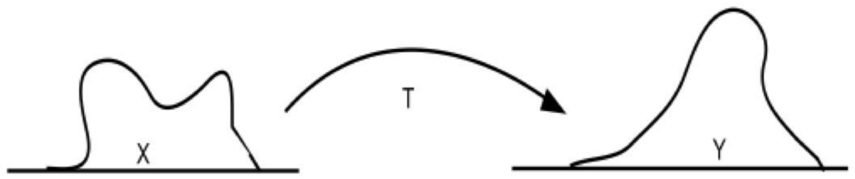
$$J(\pi) - J(\pi^*)$$

$$= \mathbb{E}_{s,t\sim d_\pi}\left[\mathbb{E}_{a\sim\pi_s}[Q^t_{\pi^*}(s,a)] - \mathbb{E}_{a^*\sim\pi^*_s}[Q^t_{\pi^*}(s,a^*)]\right]$$

Wasserstein Distance

$$D_W(p,q) := \sup_{f:\mathrm{Lip}(f(\cdot))\leq 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)]$$

$$= \inf_{\gamma\in\Gamma(p,q)} \int_{\mathcal{M}\times\mathcal{M}} d(x,y)\,d\gamma(x,y),$$



Formulation

$$J(\pi) = J(\pi') + \mathbb{E}_{s,t\sim d_\pi}\mathbb{E}_{a\sim\pi_s}[A^t_{\pi'}(s,a)]$$

$$J(\pi) - J(\pi^*)$$
$$= \mathbb{E}_{s,t\sim d_\pi}\left[\mathbb{E}_{a\sim\pi_s}[Q^t_{\pi^*}(s,a)] - \mathbb{E}_{a^*\sim\pi^*_s}[Q^t_{\pi^*}(s,a^*)]\right]$$
$$\leq C_{\pi^*}\mathbb{E}_{s,t\sim d_\pi}\left[D_W(\pi,\pi^*)\right]$$
$$\leq C_{\pi^*}\mathbb{E}_{s,t\sim d_\pi}\mathbb{E}_{a\sim\pi_s}\mathbb{E}_{a^*\sim\pi^*_s}[\|a-a^*\|],$$

learner policy experts policy

$$\min_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{t=1}^T \hat{c}(s_t,a_t)\right].$$

$$\hat{c}(s,a) = \mathbb{E}_{a^*\sim\pi^*_s}[\|a-a^*\|]$$

Online Imitation Learning Problem

Online Imitation Learning

$$\min_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{t=1}^{T} \hat{c}(s_t, a_t)\right]. \longrightarrow \hat{c}(s,a) = \mathbb{E}_{a^* \sim \pi_s^*}[||a - a^*||]$$

online IL problem

DAgger

Sequence of
Supervised Learning Problems

$$\pi_i = \arg\min_\pi \mathbb{E}_{\mathcal{D}}[\hat{c}(s_t, a_t)],$$

Batch Imitation Learning

Flipping the policies

$$
\begin{aligned}
&J(\pi) - J(\pi^*) \\
&= \mathbb{E}_{s^*, t \sim d_{\pi^*}} \left[\mathbb{E}_{a \sim \pi_{s^*}} [Q^t_\pi(s^*, a)] - \mathbb{E}_{a^* \sim \pi^*_{s^*}} [Q^t_\pi(s^*, a^*)] \right] \\
&\leq \mathbb{E}_{s^*, t \sim d_{\pi^*}} \mathbb{E}_{a^* \sim \pi^*_{s^*}} \left[C^t_\pi(s^*) \tilde{c}_\pi(s^*, a^*) \right]. \qquad (8)
\end{aligned}
$$

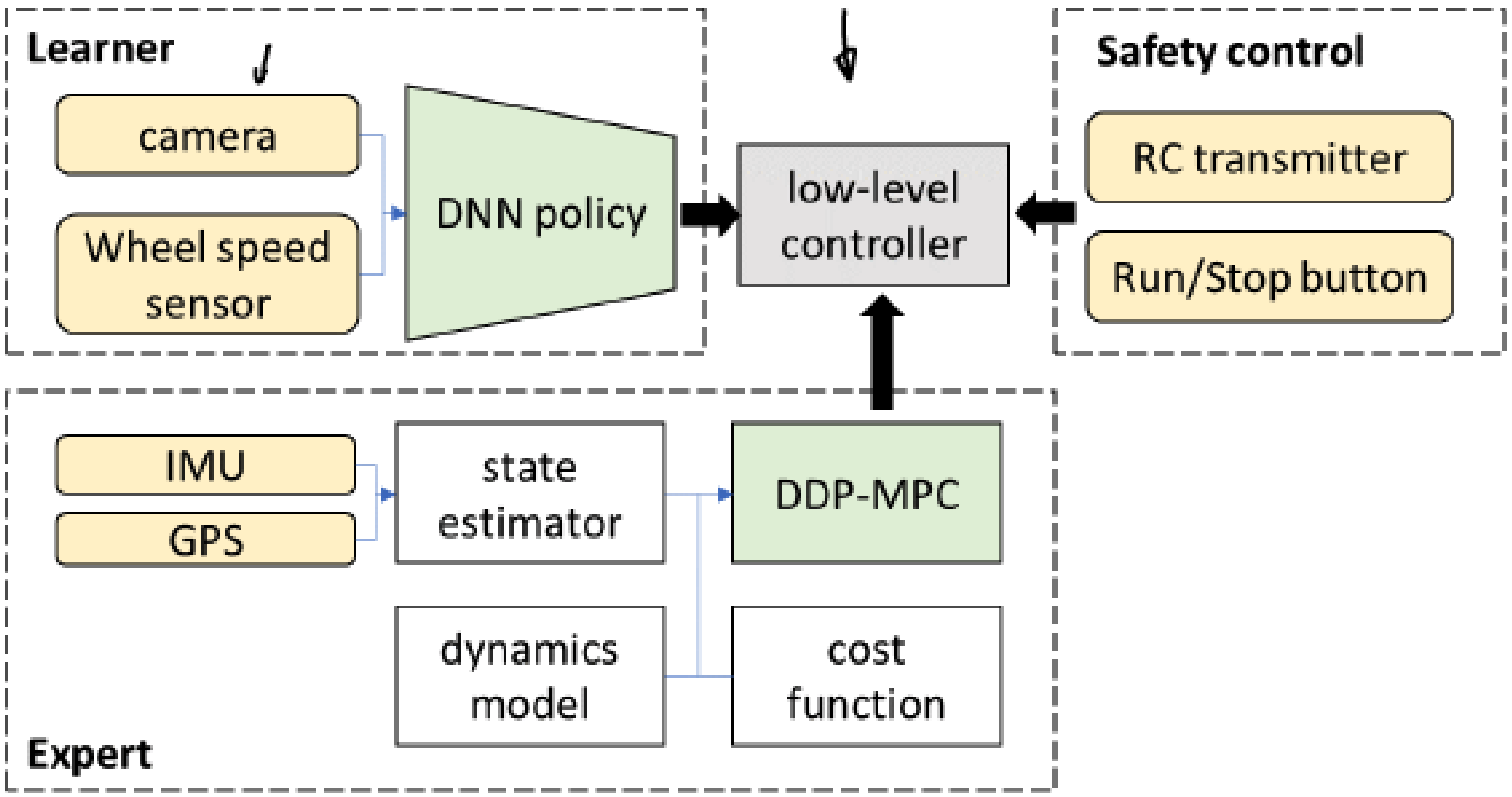
expert policy

expert policy

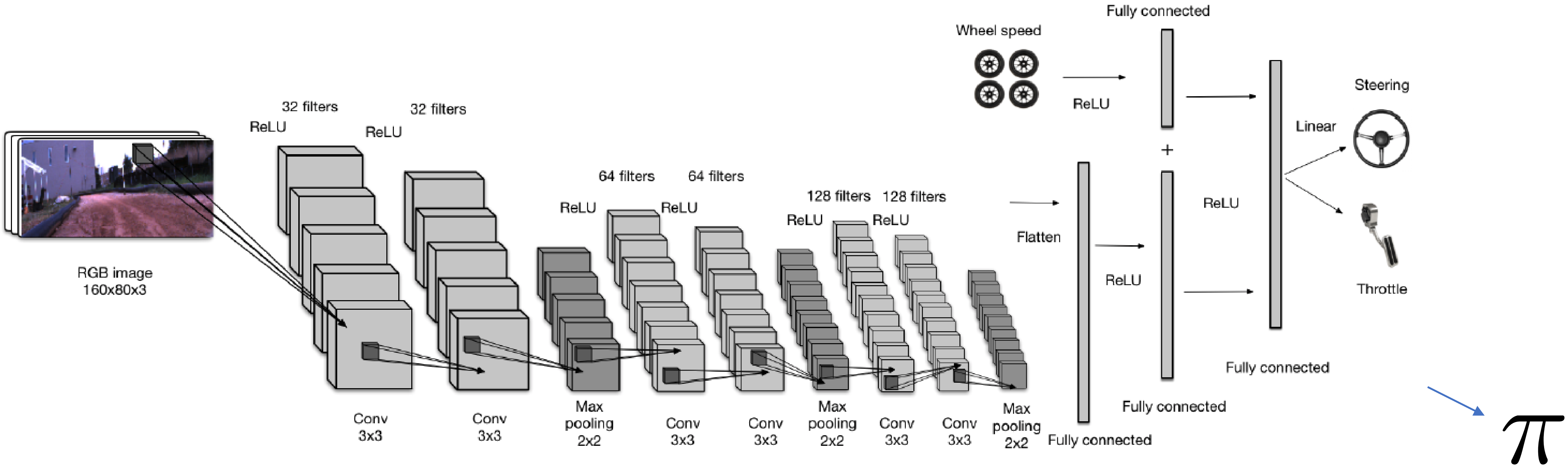
$$\min_\pi \mathbb{E}_{\rho_{\pi^*}} \left[\sum_{t=1}^T \tilde{c}_\pi(s^*_t, a^*_t) \right],$$

This resumes to supervised learning

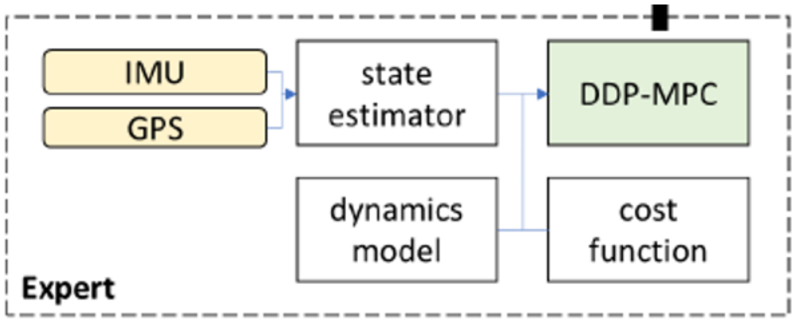
System Diagram



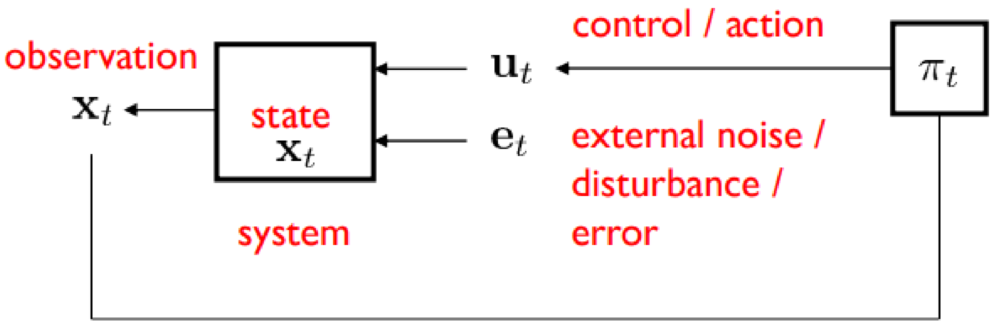
DNN Control Policy



Expert – recall control



Optimal Control



$$
\begin{aligned}
\underset{\pi_0,\ldots,\pi_{T-1}}{\text{minimize}} \quad & \mathbb{E}_{\mathbf{e}_t}\left[\sum_{t=0}^{T} c(\mathbf{x}_t, \mathbf{u}_t)\right] \\
\text{subject to} \quad & \mathbf{x}_{t+1} = f_t(\mathbf{x}_t, \mathbf{u}_t, \mathbf{e}_t) \\
& \mathbf{u}_t = \pi_t(\mathbf{x}_{0:t}, \mathbf{u}_{0:t-1})
\end{aligned}
$$

known dynamics

control law / policy

Sparse Spectrum Gaussian Process

Expert – MPC

Differential Dynamic Program (DDP) ~ Recall iLQR

Given an initial sequence of states $\bar{\mathbf{x}}_0, \ldots, \bar{\mathbf{x}}_N$ and actions $\bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{u}}_N$

Linearize dynamics $f(\mathbf{x}_t, \mathbf{u}_t) \approx \tilde{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \underbrace{f(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t)}_{\mathbf{b}_t} + \underbrace{\frac{\partial f}{\partial \mathbf{x}}(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t)}_{A_t}\underbrace{(\mathbf{x}_t - \bar{\mathbf{x}}_t)}_{\delta\mathbf{x}_t} + \underbrace{\frac{\partial f}{\partial \mathbf{u}}(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t)}_{B_t}\underbrace{(\mathbf{u}_t - \bar{\mathbf{u}}_t)}_{\delta\mathbf{u}_t}$

Taylor expand cost $c(\mathbf{x}_t, \mathbf{u}_t) \approx \tilde{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = c(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) + \underbrace{\nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t)}_{\mathbf{h}_t} \begin{bmatrix} \mathbf{x}_t - \bar{\mathbf{x}}_t \\ \mathbf{u}_t - \bar{\mathbf{u}}_t \end{bmatrix} + 1/2 \begin{bmatrix} \mathbf{x}_t - \bar{\mathbf{x}}_t \\ \mathbf{u}_t - \bar{\mathbf{u}}_t \end{bmatrix}^T \underbrace{\nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t)}_{H_t} \begin{bmatrix} \mathbf{x}_t - \bar{\mathbf{x}}_t \\ \mathbf{u}_t - \bar{\mathbf{u}}_t \end{bmatrix}$

Use LQR backward pass on the approximate dynamics $\tilde{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t)$ and cost $\tilde{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t)$

Do a forward pass to get $\delta\mathbf{u}_t$ and $\delta\mathbf{x}_t$ and update state and action sequence $\bar{\mathbf{x}}_0, \ldots, \bar{\mathbf{x}}_N$ and $\bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{u}}_N$

Related works:

TABLE I: Comparison of our method to prior work on IL for autonomous driving

Methods	Tasks	Observations	Action	Algorithm	Expert	Experiment
[1]	On-road low-speed	Single image	Steering	Batch	Human	Real &simulated
[23]	On-road low-speed	Single image & laser	Steering	Batch	Human	Real &simulated
[24]	On-road low-speed	Single image	Steering	Batch	Human	Simulated
[20]	Off-road low-speed	Left & right images	Steering	Batch	Human	Real
[33]	On-road unknown speed	Single image	Steering + break	Online	Pre-specified policy	Simulated
Our Method	Off-road high-speed	Single image + wheel speeds	Steering + throttle	Batch & online	Model predictive controller	Real & simulated

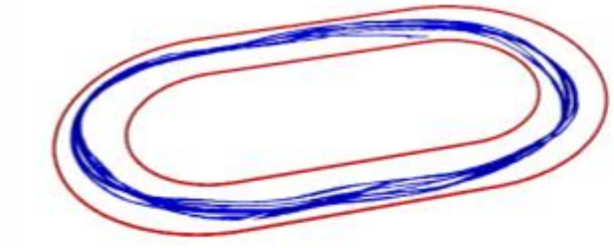
Experiment – Setup Experts

High Speed driving
at 7.5 m/s or 135 km / h

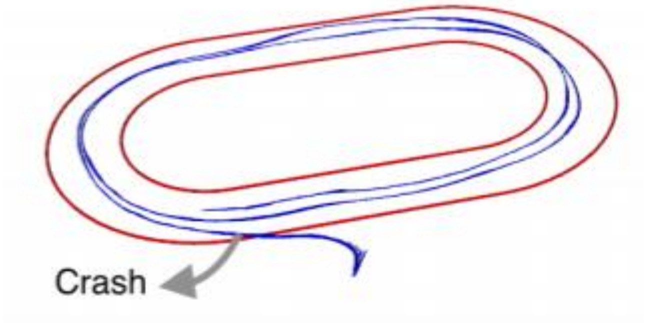
Cost for expert:

$$c(s_t, a_t) = \alpha_1 c_{\text{pos}}(s_t) + \alpha_2 c_{\text{spd}}(s_t) + \alpha_3 c_{\text{slip}}(s_t) + \alpha_3 c_{\text{act}}(a_t)$$

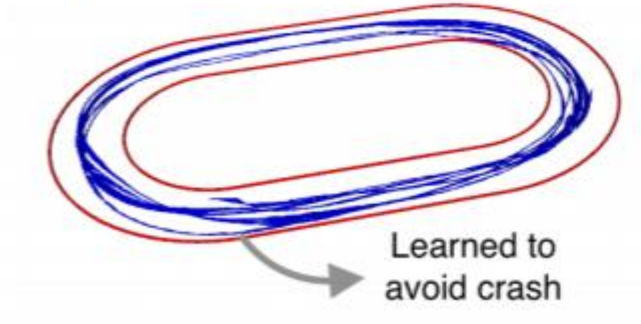
Experiment– learning trajectories



(a) MPC expert.



(b) Batch IL.

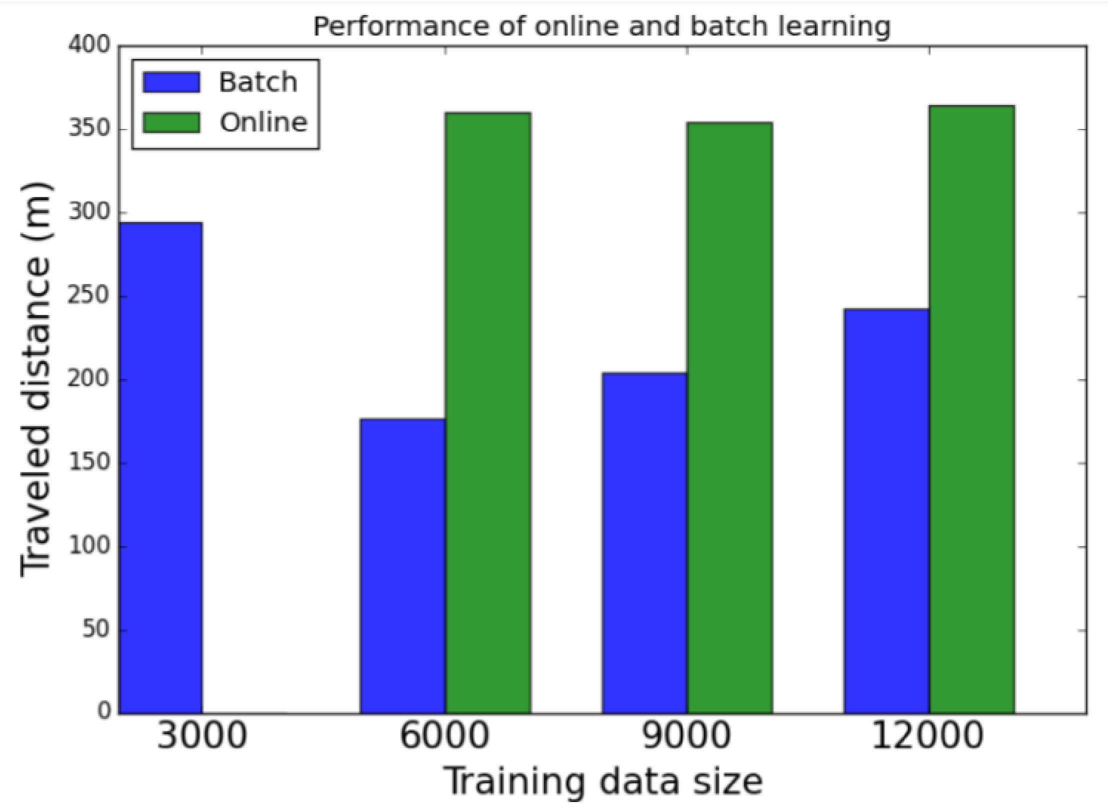


(c) Online IL.

Comparing – Loss (to expert)

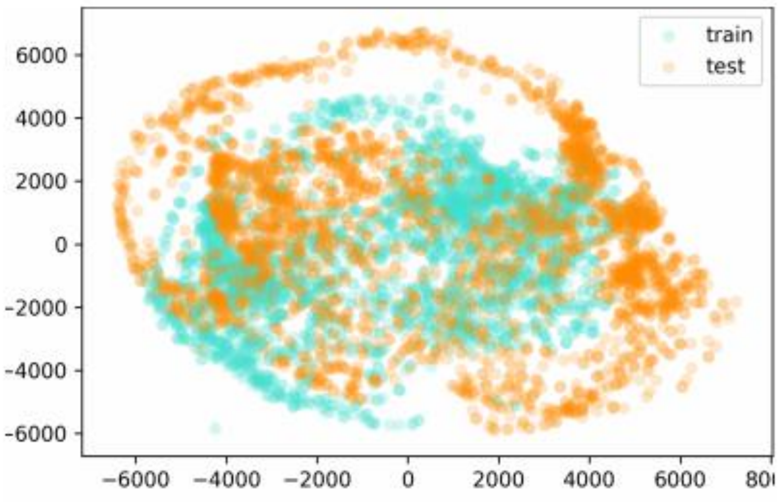
Policy	Avg. speed	Top speed	Training data	Completion ratio	Total loss	Steering/Throttle loss
Expert	6.05 m/s	8.14 m/s	N/A	100 %	0	0
Batch	4.97 m/s	5.51 m/s	3000	100 %	0.108	0.092/0.124
Batch	6.02 m/s	8.18 m/s	6000	51 %	0108	0.162/0.055
Batch	5.79 m/s	7.78 m/s	9000	53 %	0.123	0.193/0.071
Batch	5.95 m/s	8.01 m/s	12000	69 %	0.105	0.125/0.083
Online (1 iter)	6.02 m/s	7.88 m/s	6000	100 %	0.090	0.112/0.067
Online (2 iter)	5.89 m/s	8.02 m/s	9000	100 %	0.075	0.095/0.055
Online (3 iter)	6.07 m/s	8.06 m/s	12000	100 %	0.064	0.073/0.055

Comparing – distance travelled

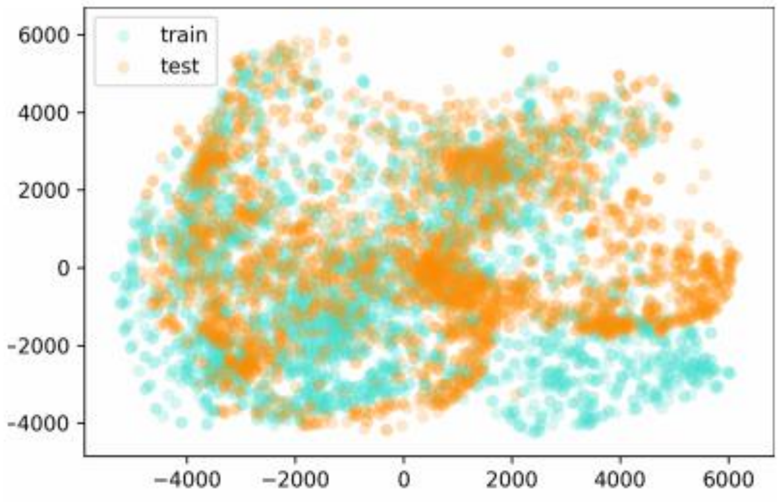


Comparing – generalizability

t-Distributed Stochastic Neighbor Embedding (t-SNE)

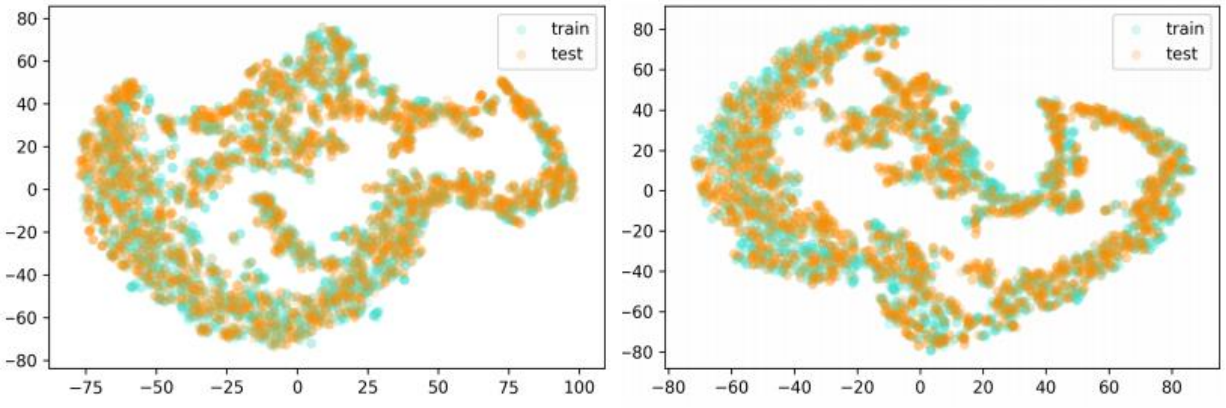


(a) Batch raw image

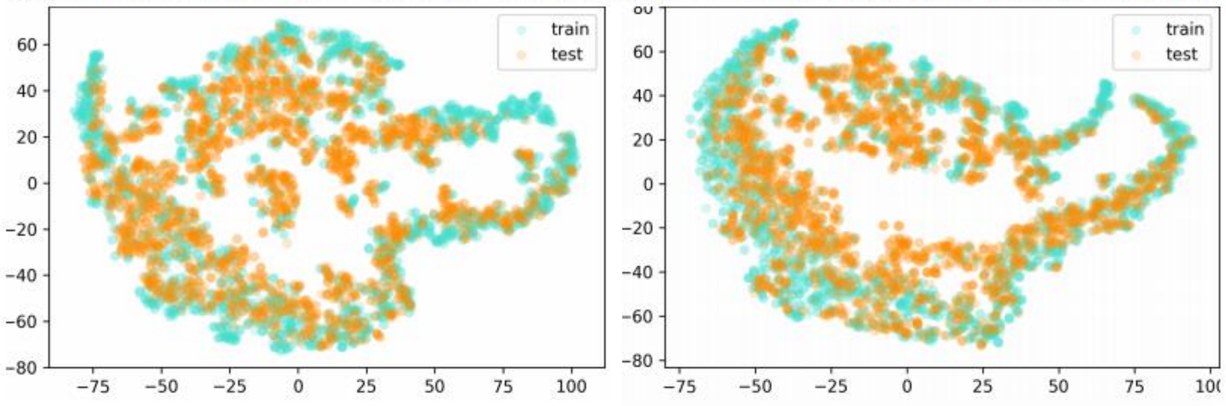


(b) Online raw image

Comparing – generalizability

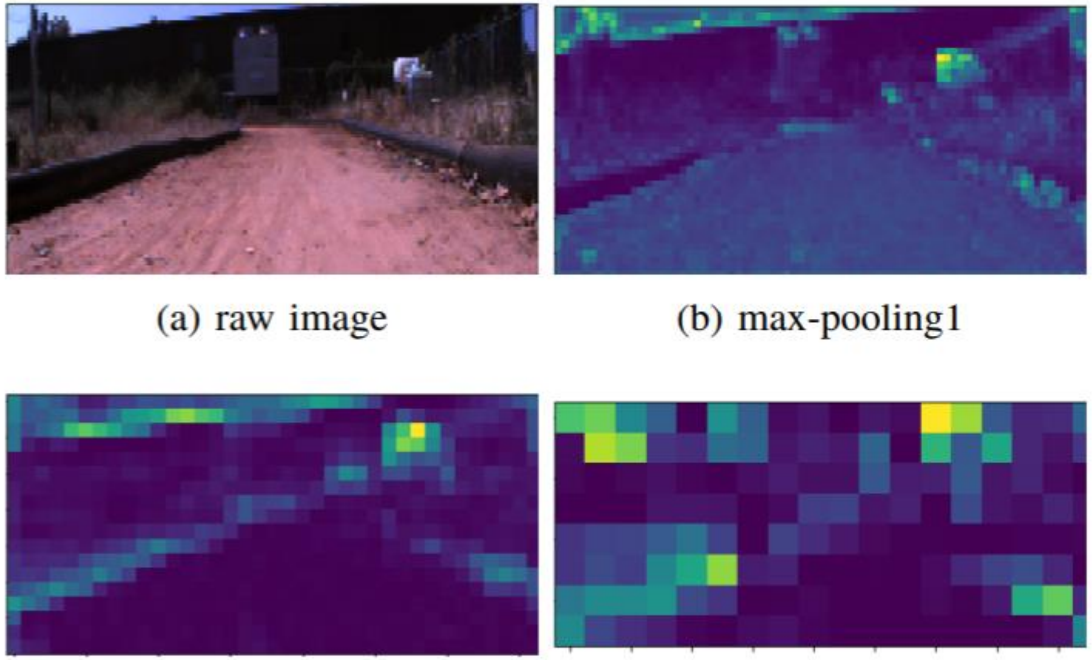


(a) Batch data wrt online model (b) Online data wrt online model



(c) Batch data wrt batch model (d) Online data wrt batch model

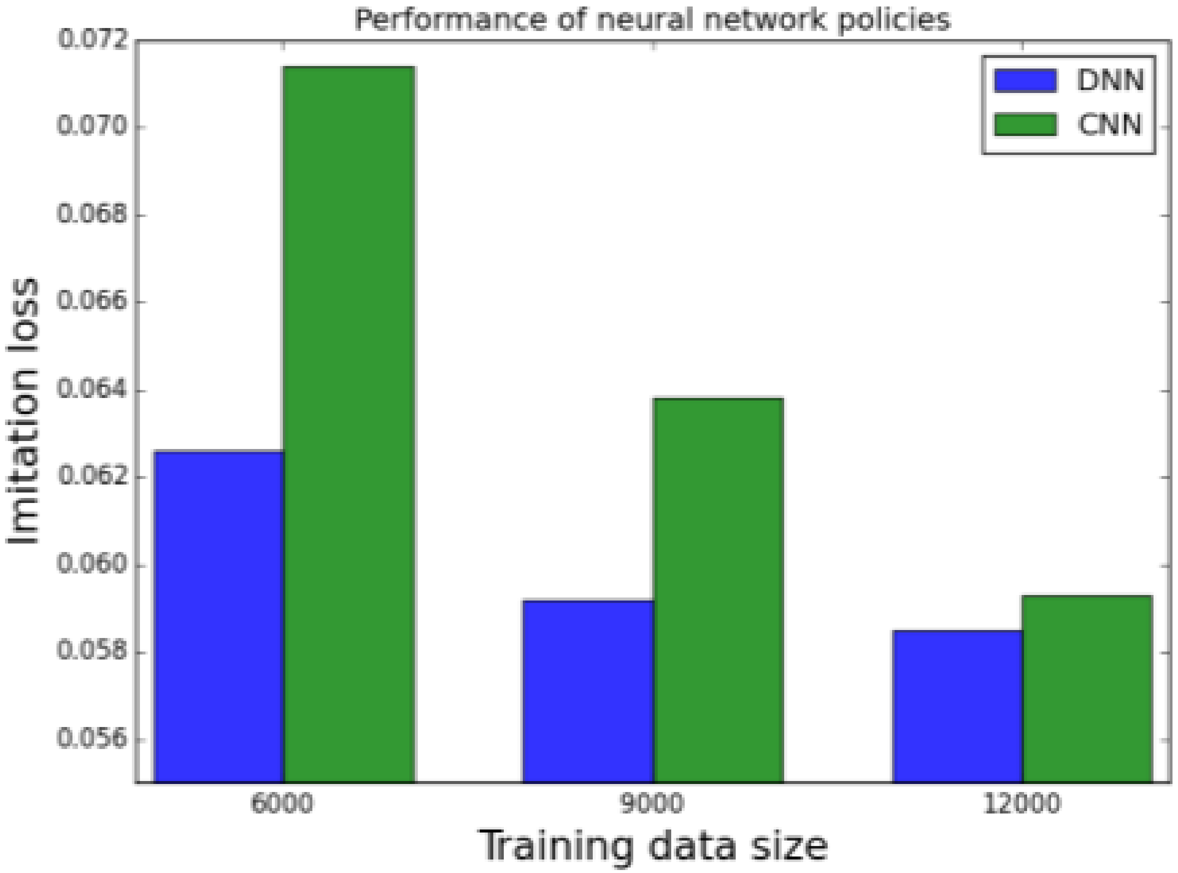
DNN – high and low capture

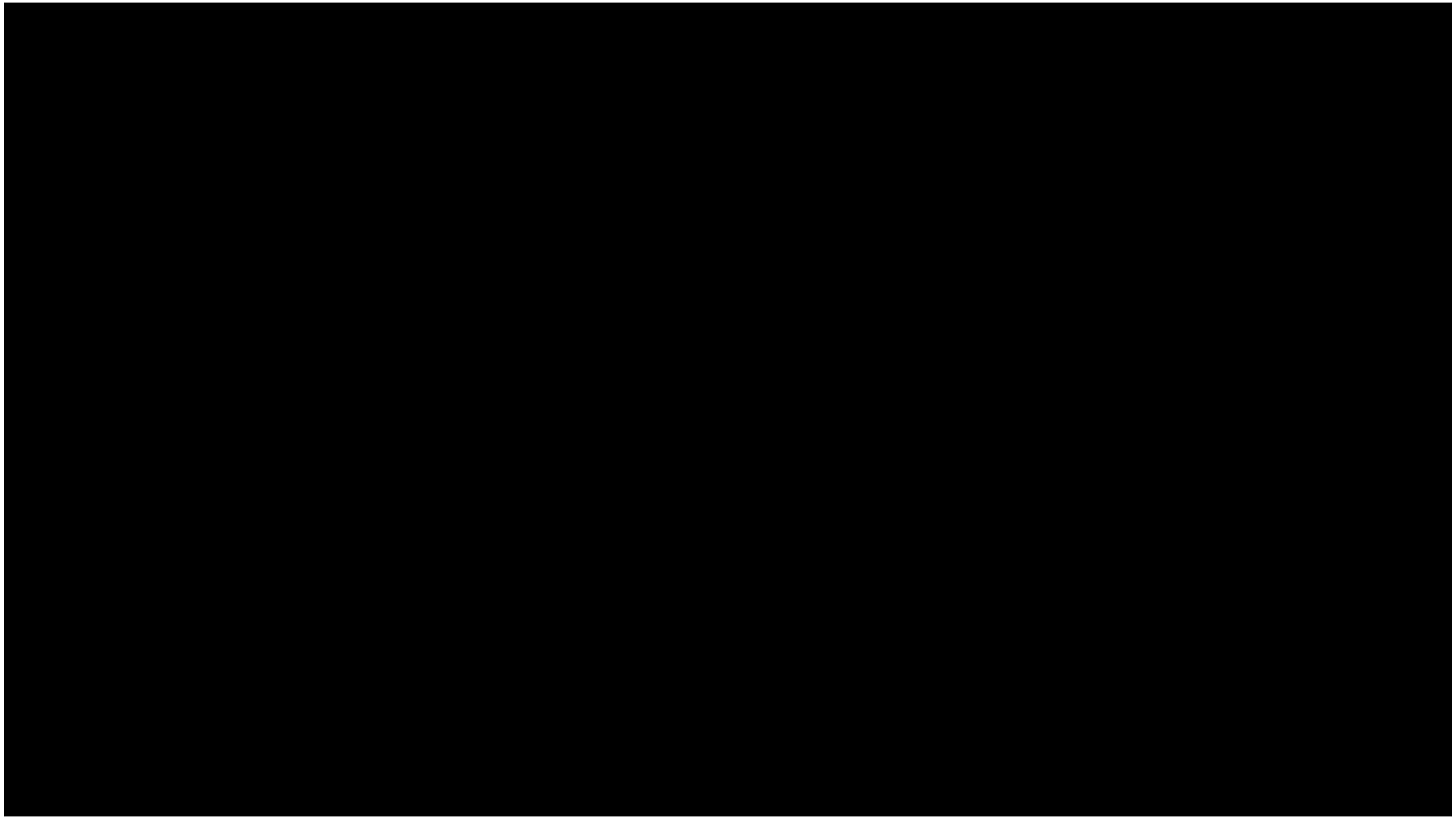


(a) raw image (b) max-pooling1 (c) max-pooling2 (d) max-pooling3

Fig. 9: The input RGB image and the averaged feature maps for each max-pooling layer.

DNN > CNN … or Limitation?



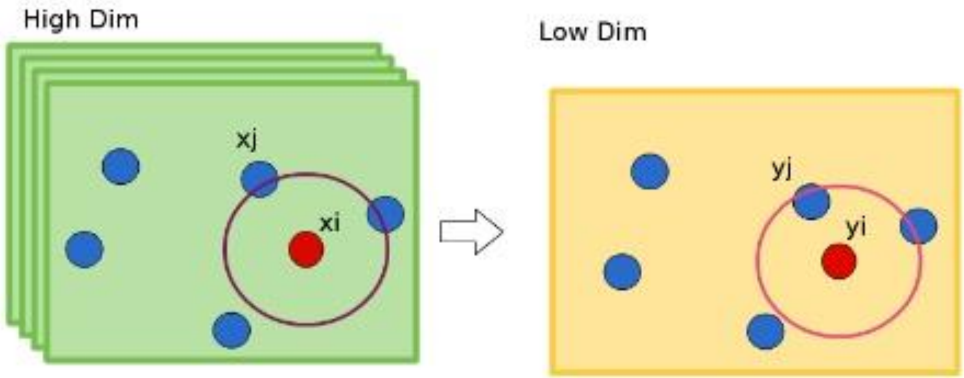


Thank you!

Any Questions?

Introduction

Measure pairwise similarities between high-dimensional and low-dimensonal objects



$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

End-to-end Driving via Conditional Imitation Learning

Wei Cui

Electrical & Computer Engineering
University of Toronto

Feb 1st 2019

Brief Overview of the Paper

- This paper focuses on the task of self-driving, while allowing users to interact with high-level navigation commands.
- As the conventional imitation learning is not sufficient, the agent solves the task through conditional imitation learning.

▸ Link


FPS
Speed
18 km/h
Gear
1
4x

Problem Formulation

- The main task : given specified sensory inputs, the agent achieves self-driving through computing controller outputs, while following navigational guidance.
- **Sensory Inputs** (Observation $\mathbf{o}$) :

&

Measurements (i.e. Speed)

- **Controller Outputs** :

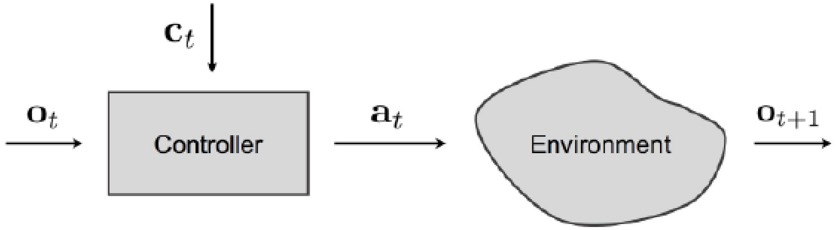
$$\mathbf{a} = [s, \alpha] \tag{1}$$

  - s : steering angle
  - α : acceleration

Conditional Imitation Learning

- Conditional Imitation Learning : for both training and testing, the agent receives additional input : **c** (navigation command).
- The formulation for Conditional Imitation Learning :

$$\min_{\theta} \sum_{i} \mathcal{L}(F(\mathbf{o}_i, \mathbf{c}_i; \theta), \mathbf{a}_i) \tag{2}$$

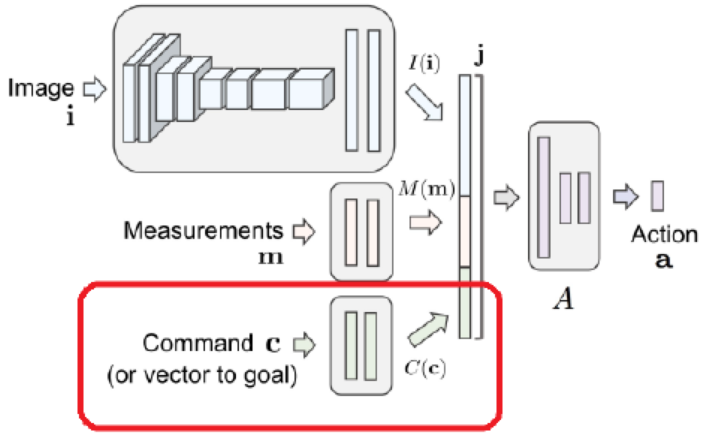


- The high level commands explored for this paper :

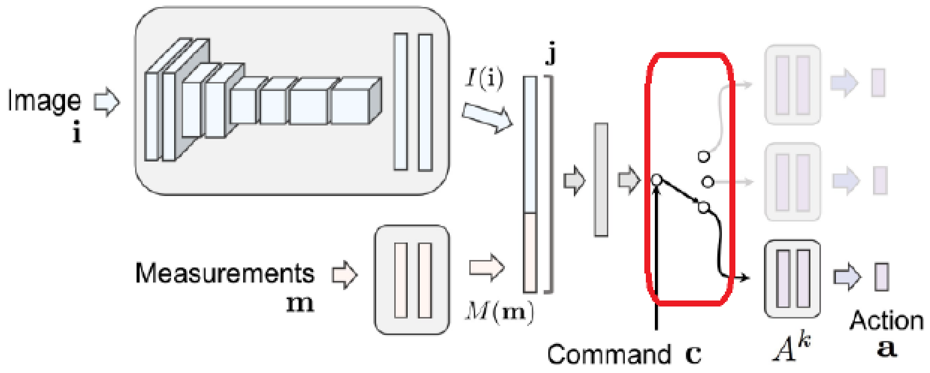
$$\mathbf{c} \in \{\text{continue, left, straight, right}\} \tag{3}$$

Network Architecture

- Two models are explored :
  - ***command input*** model
  - ***branched*** model



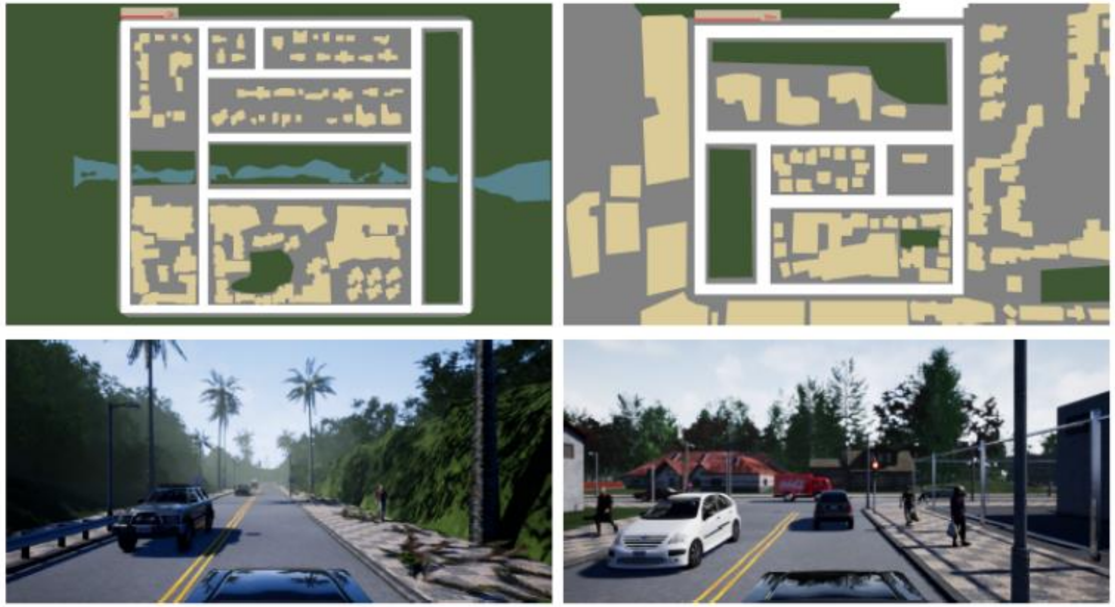
(a)



(b)

System Setup

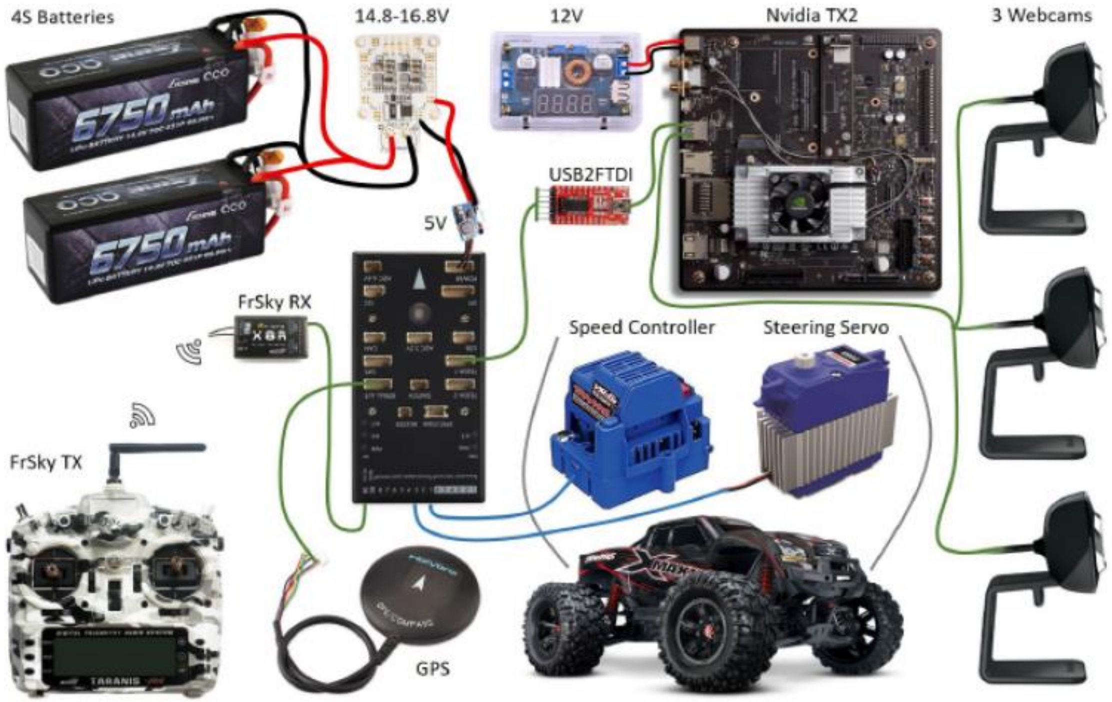
- Two systems : a simulated urban environment and a physical system.
- **Simulated Environment :** an urban driving simulator, CARLA.
- Town 1 for training; Town 2 for exclusive testing.



Town 1 (training) Town 2 (testing)

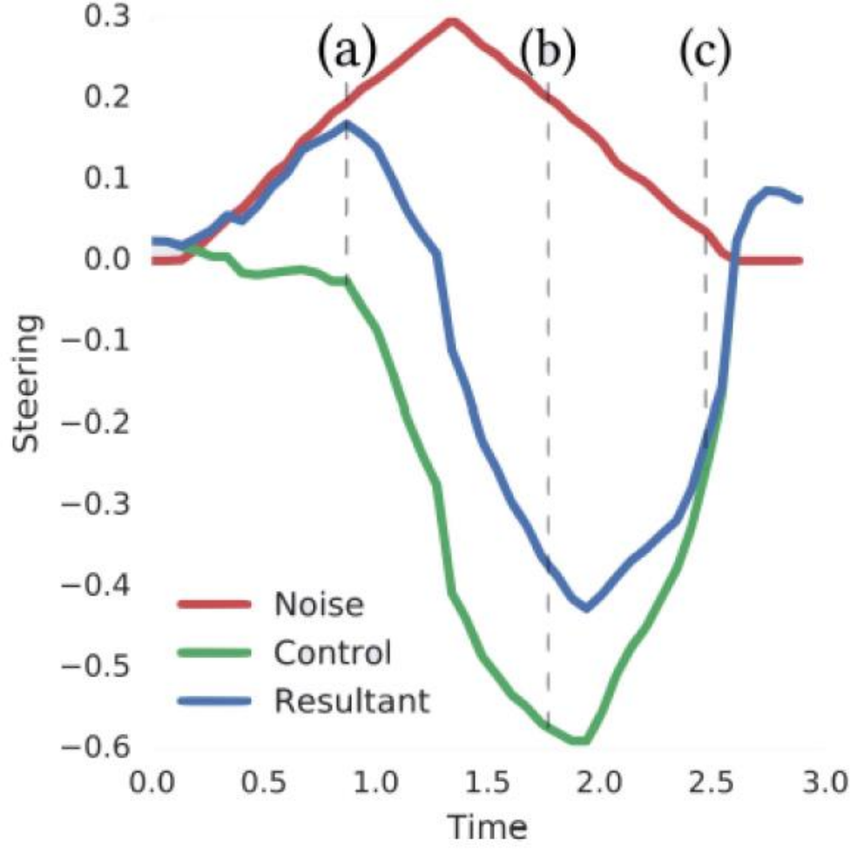
System Setup (Cont'd)

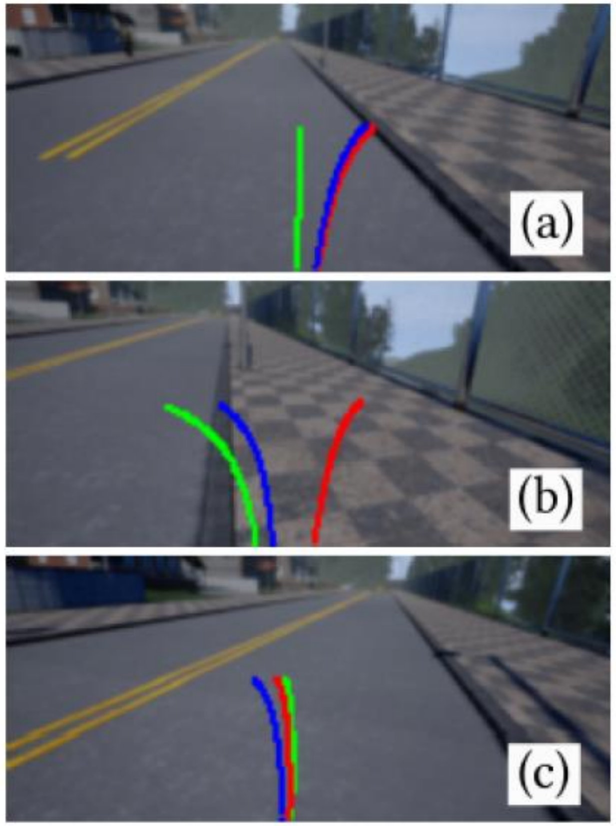
- **Physical System :** An off-the-shelf 1/5 scale truck is used (Traxxas Maxx), with an embedded computer (Nvidia TX2) which the agent model runs on.



Training Data Preparation

- Firstly, additional state-action pairs are collected through injecting noise into expert's control, and let the expert to respond. This method is an alternative to DAgger (not used in the paper).





Training Data Preparation (Cont'd)

- The authors further augment the data through applying random transformations to the images as inputs to the agent.
- The types of transformations include :
  - Change in contrast, brightness, and tone.
  - Adding Gaussian blur, Gaussian noise, salt-and-pepper noise (sparse white and black pixels).
  - Region dropout (masking out a random set of rectangles of roughly 1% of image area)

Training Details

- **Some normalization used :** 50% dropout after fully-connected hidden layers, and 20% dropout after convolutional layers.
- **Loss Function :** As mentioned before, each action contains a tuple of signals : $\mathbf{a} = [s, \alpha]$.
  With model's action $\mathbf{a}$ and expert's action $\mathbf{a}_e$, the per-sample loss function :

$$\mathcal{L}(\mathbf{a}, \mathbf{a}_e) = ||s - s_e||^2 + \lambda_a ||\alpha - \alpha_e||^2 \tag{4}$$

- Different than DAgger, the agent's parameters are optimized once after all the data is collected, without iterative loops.
- For the command-conditional models, minibatches were constructed to contain an equal number of samples with each command.

Testing Methods : Simulation Environment

- **Baseline Method :**
  - Standard Imitation Learning : $\mathbf{a} = \mathcal{F}(\mathbf{o})$
- **Variations on the current model :** Investigate on the importance of each component.
  - The ***command input*** model.
  - The ***branched*** model trained without noise-injected data.
  - The ***branched*** model trained without data augmentation.
  - The ***branched*** model implemented with a shallower network.

Testing Results : Simulation Environment

Model	Success rate Town 1	Success rate Town 2	Km per infraction Town 1	Km per infraction Town 2
Non-conditional	20%	26%	5.76	0.89
Ours branched	**88%**	**64%**	2.34	1.18
Ours cmd. input	78%	52%	3.97	1.30
Ours no noise	56%	22%	1.31	0.54
Ours no aug.	80%	0%	4.03	0.36
Ours shallow net	46%	14%	0.96	0.42

Testing Methods & Results : Physical System

- The authors picked only 3 competitive methods in simulation environment testing for this comparison :
  - The ***command input*** model.
  - The ***branched*** model trained without noise-injected data.
  - The ***branched*** model trained without data augmentation.
- The results still support the necessity for each of the model's component :

Model	Missed turns	Interventions	Time
Ours branched	**0%**	**0.67**	**2:19**
Ours cmd. input	11.1%	2.33	4:13
Ours no noise	24.4%	8.67	4:39
Ours no aug.	73%	39	10:41

Conclusions

- This paper recognizes one key problem in conventional imitation learning : expert's demonstrations are often decided by certain latent factors not included in the observations (such as intentions).
- It is important to introduce a channel for the communication of this extra information, which motivates conditional imitation learning
- The method has been shown with its efficacy in self-driving task, where users' high-level navigation needs are also considered into the requirement.

A few discussions of mine...

- Under misguiding **c**, the agent might perform dangerous actions (such as [**o**=driving on the straight highway, **c**=turn right !]). This is never tested for this work (at least based on the paper).
- Under these considerations, perhaps a rejection option against certain **c** should be built into the agent as a safety feature.
- It is not convincing to me why the authors decided to remove the benchmark during testing the physical system case.

Appendix A : Network Architecture Details

- For both architectures explored as shown above, the individual modules are identical.
- **The image module :**
  - Consists of 8 convolutional and 2 fully connected layers.
  - The convolution kernel size is 5 in the first layer and 3 in the following layers. The first, third, and fifth convolutional layers have a stride of 2.
  - The number of channels increases from 32 in the first convolutional layer to 256 in the last.
  - Fully-connected layers contain 512 units each.
- **Other modules :**
  - Implemented as standard multilayer perceptrons, with ReLU nonlinearities after all hidden layers.