

CSC2626

Imitation Learning for Robotics

Florian Shkurti

Week 4: Combining Imitation with Planning, Control, and Reinforcement Learning #1

Today's agenda

- Respecting demonstrations by a large margin (DQN with demonstrations)
- Demonstrating both actions and Q-values (AggreVaTe)
- Autonomous driving for high-speed racecars
- Conditional imitation learning
- End-to-end neural motion planner

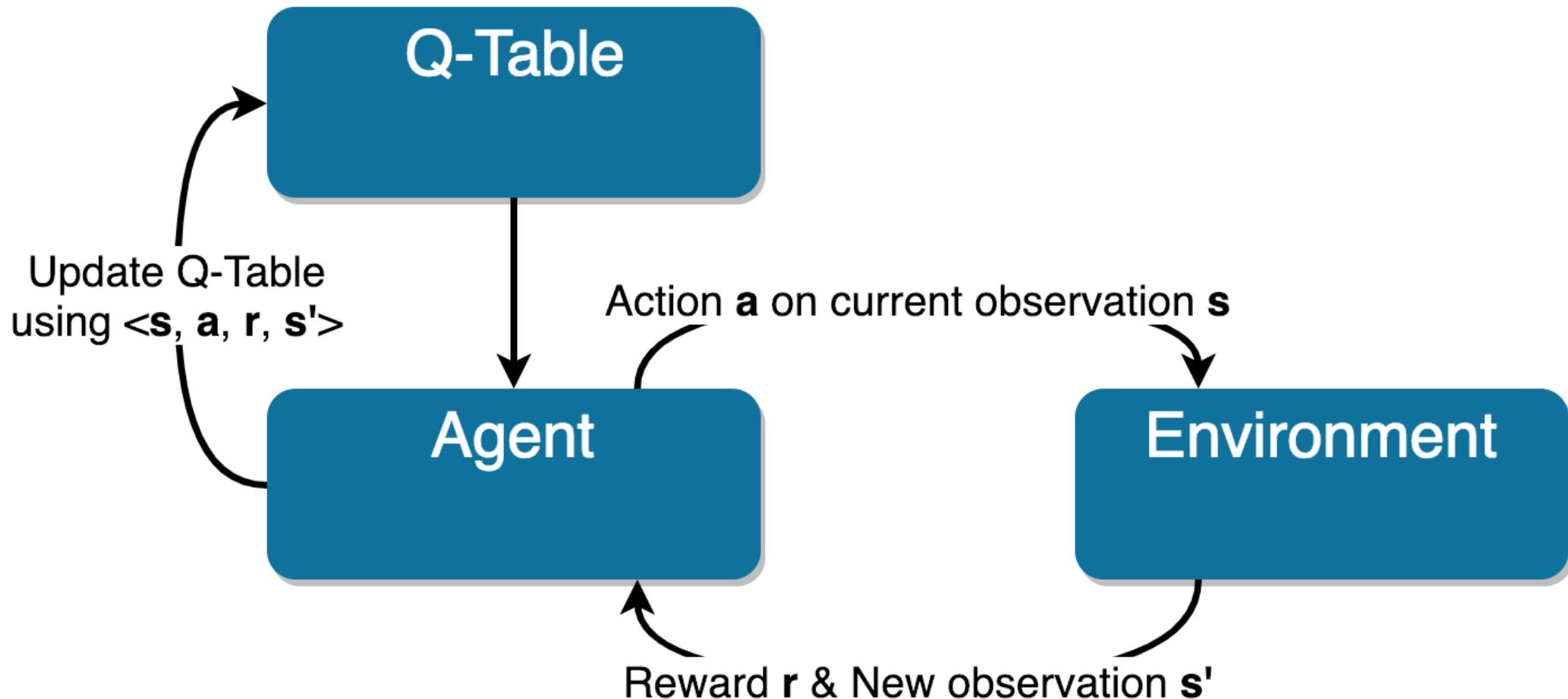
Acknowledgments

Today's slides are based on student presentations from 2019 by: Bryan Chan, Chandripal Budnarain, Renato Fereira, David Acuna, Brenna Li, Wei Cui

Deep Q-Learning from Demonstrations (**DQfD**)

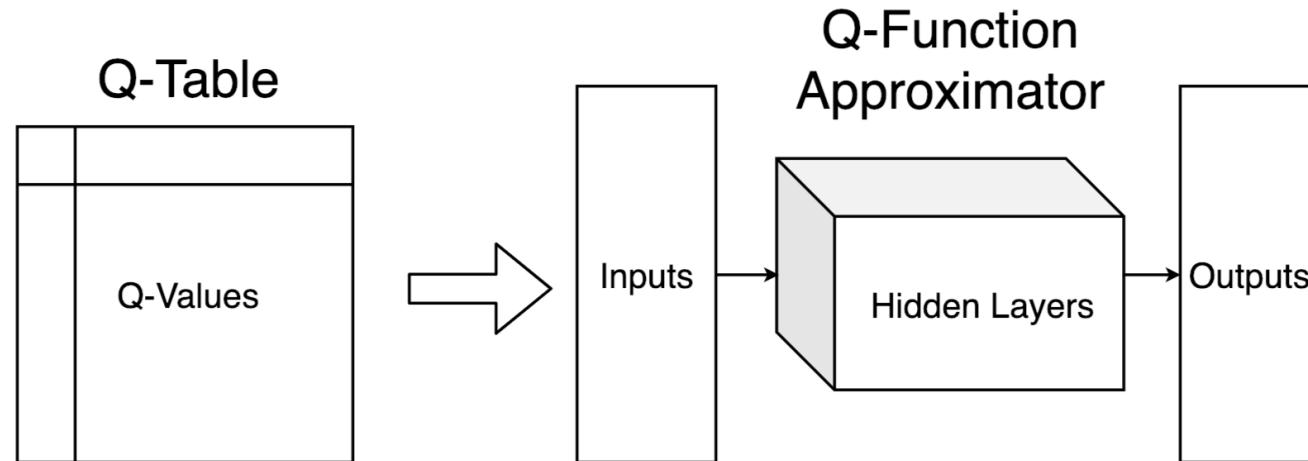
Bryan Chan & Chandripal Budnarain

Q-Learning Algorithm

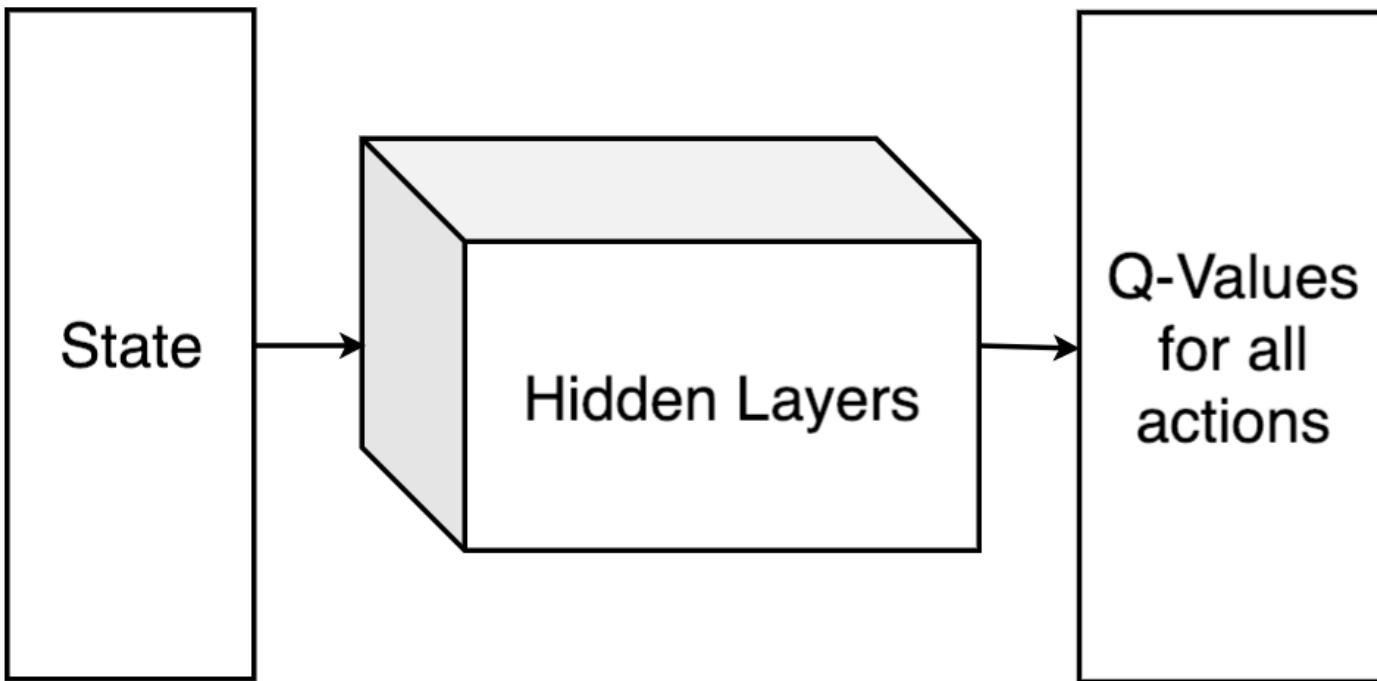


Deep Q-Network (DQN)

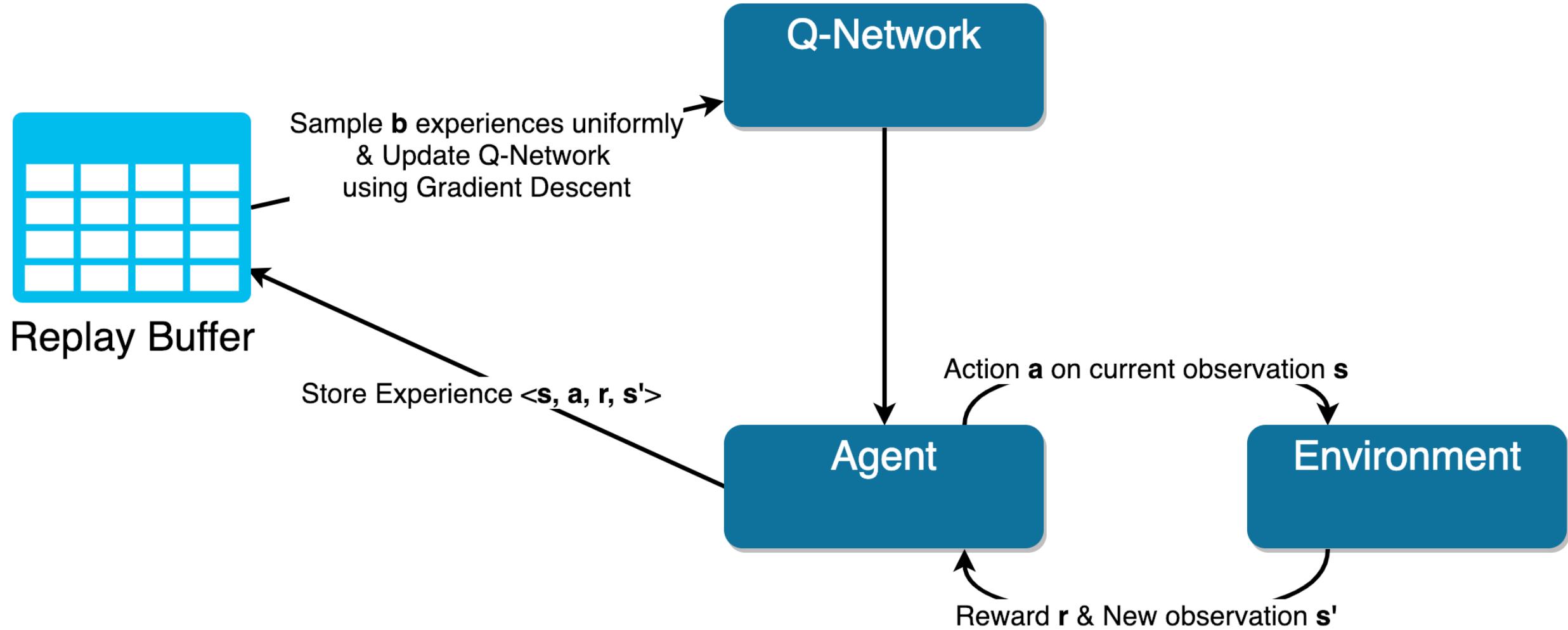
- State-action space might be too big for storing a Q-table!
- Idea: Replace Q-table with a neural network that approximates Q-values
- Deep Q-Network = Deep Learning + Q-Learning



Q-Function Approximator



DQN Algorithm



How to Combine Demonstration Data with DQN?

Loss Function

- Recall that the loss function for Q-Learning is:

$$J_{\text{DQN}}(\phi) = E_{\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s}), \mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})} \left[\left(Q_\phi(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + \gamma \max_{a'} Q_{\bar{\phi}}(\mathbf{s}', \mathbf{a}')) \right)^2 \right]$$

- 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:** Some values correspond to state-action pairs not observed during training and the Q-Network would propagate these values in the Bellman recursion

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:

$$J_E(\phi) = \max_a (Q_\phi(s, a) + l(a, a_E)) - Q_\phi(s, a_E)$$

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

Overall Loss Function

$$J(\phi) = J_{\text{DQN}}(\phi) + \lambda_1 J_n(\phi) + \lambda_2 J_E(\phi) + \lambda_3 J_{\text{reg}}(\phi)$$

where λ 's control the weighting between the losses,

$J_n(\phi)$ is the n-step TD-loss

$J_{\text{reg}}(\phi)$ is the L2 regularization loss

Tradeoff between following demonstration data and finding optimal Q-values

Prioritized Experience Replay

- In DQN, we sample experiences from the replay buffer uniformly
- What if we focus on addressing large mistakes first?
- We can prioritize what tuples we sample by looking at the TD-error corresponding to a tuple (s, a, r, s')

$$\delta = (r(s, a) + \gamma \max_{a'} Q_{\bar{\phi}}(s', a')) - Q_{\phi}(s, a)$$

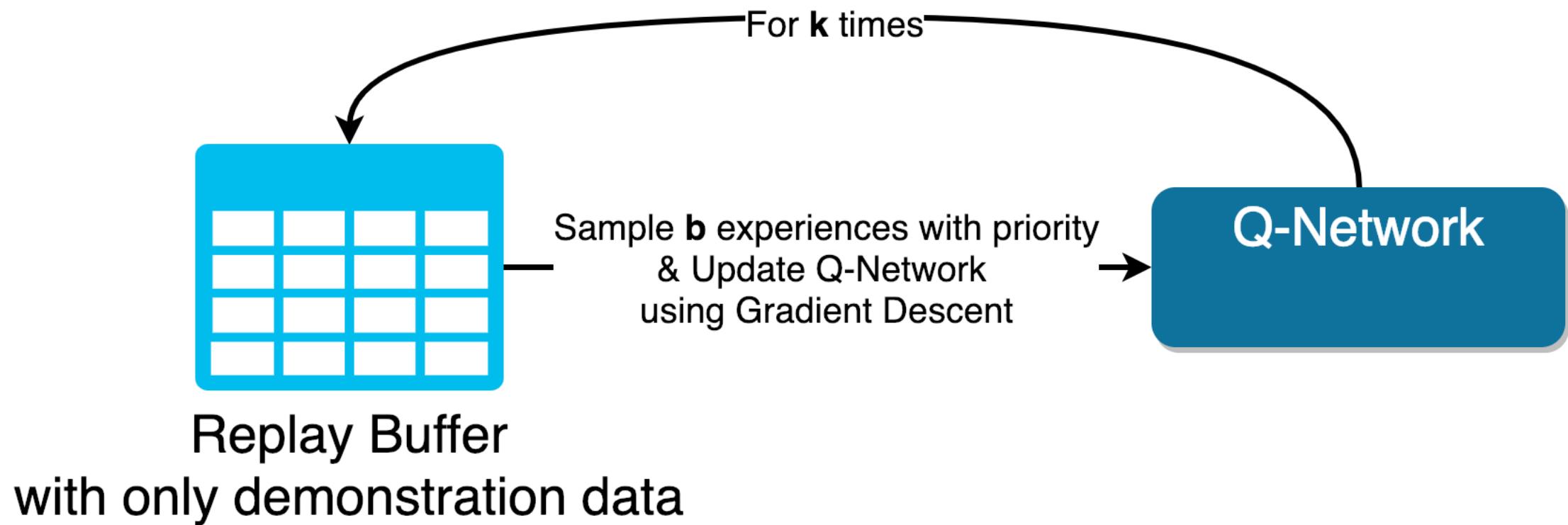
Prioritized Experience Replay

- Priority of experience i : $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ where $p_i = |\delta_i| + \epsilon$
- α (hyperparameter) decides how much prioritization is used.
If $\alpha = 0$, we are sampling uniformly
- **Issue:** Sampling with priority introduces bias and changes the behavior distribution

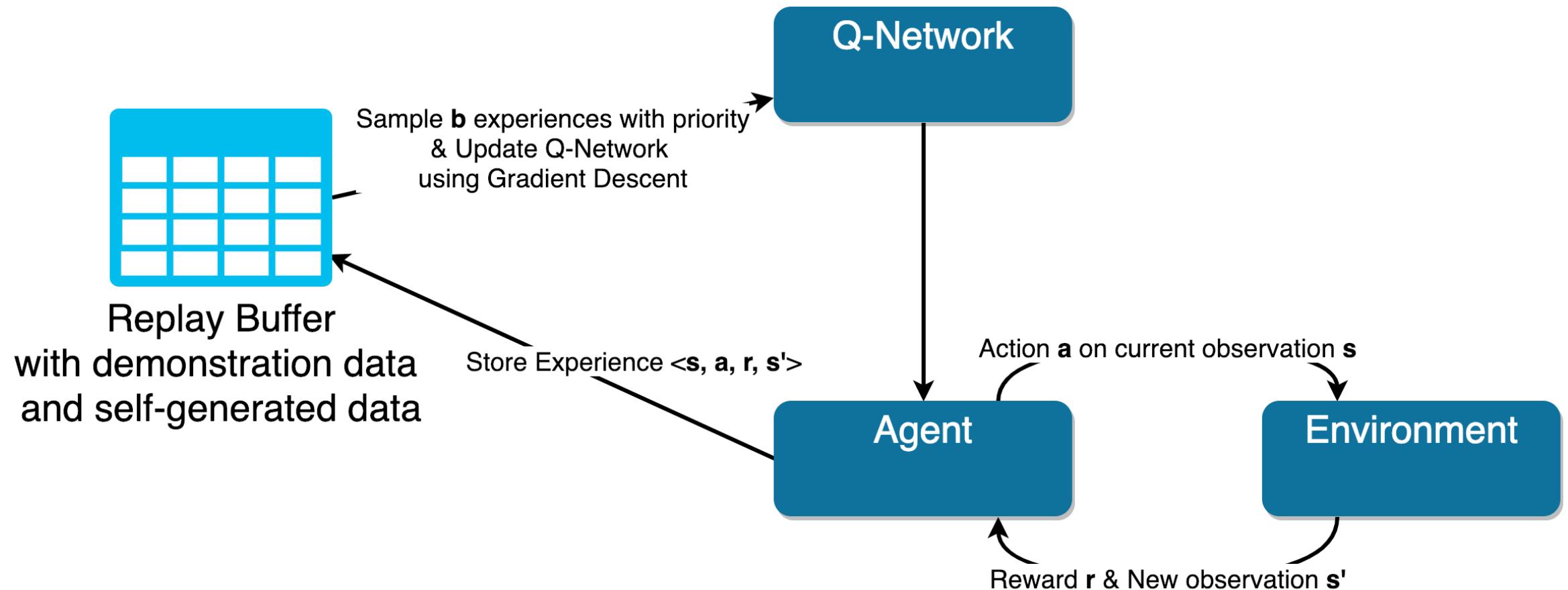
Prioritized Experience Replay

- **Solution:** Correct using weighted importance-sampling with weights
 $w_i = \left(\frac{1}{N} \frac{1}{P(i)}\right)^\beta$, where N is number of samples
- 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 change 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)

DQfD Pre-Training



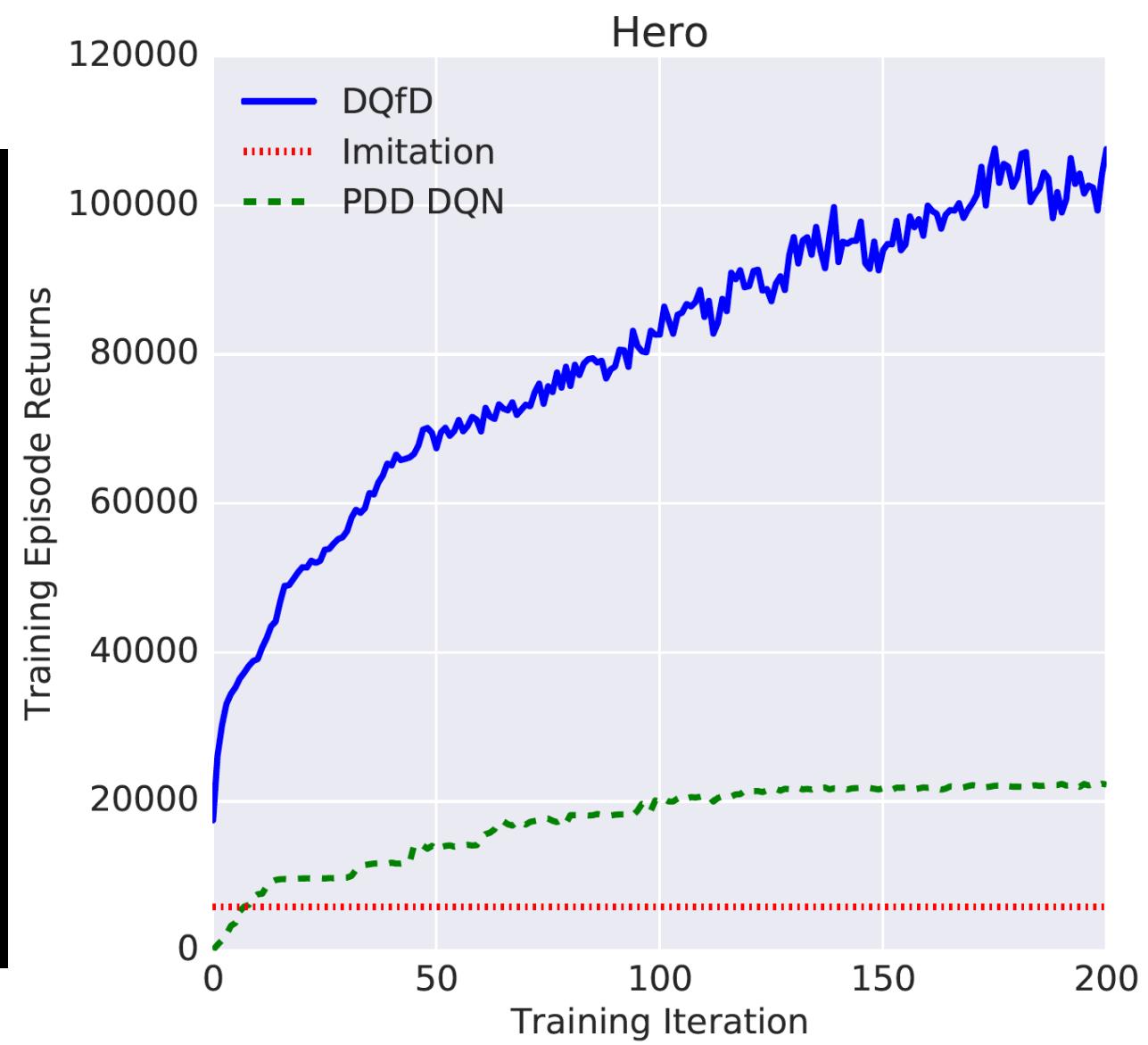
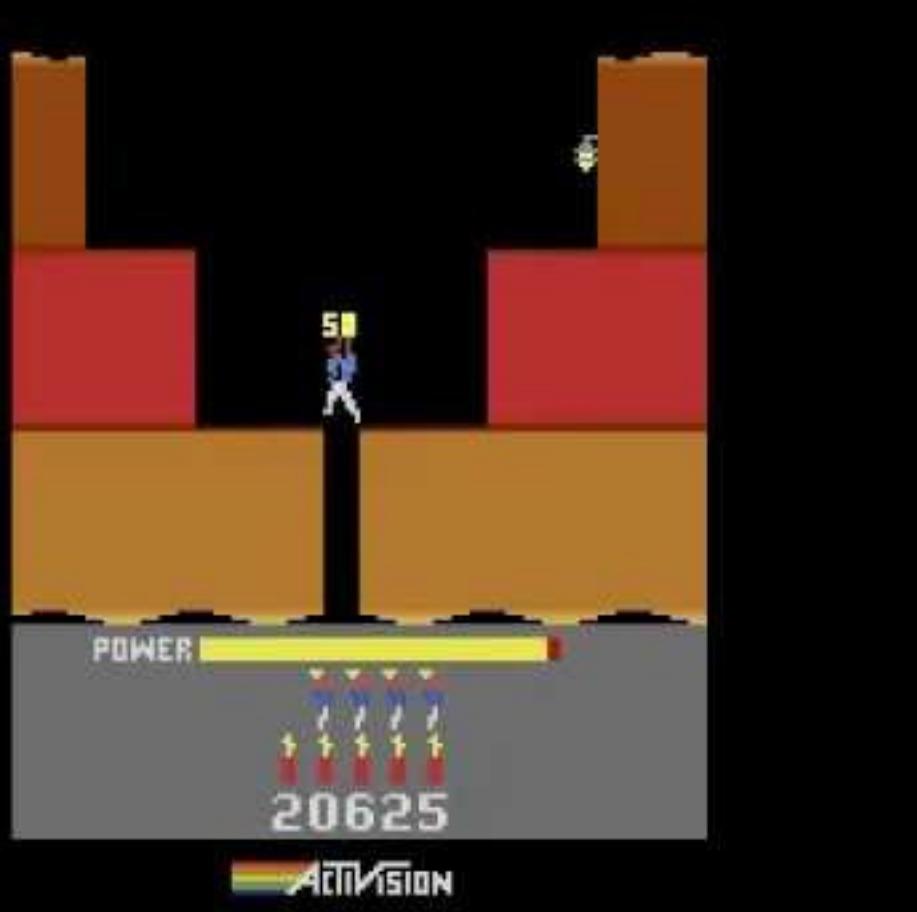
DQfD Post-Training



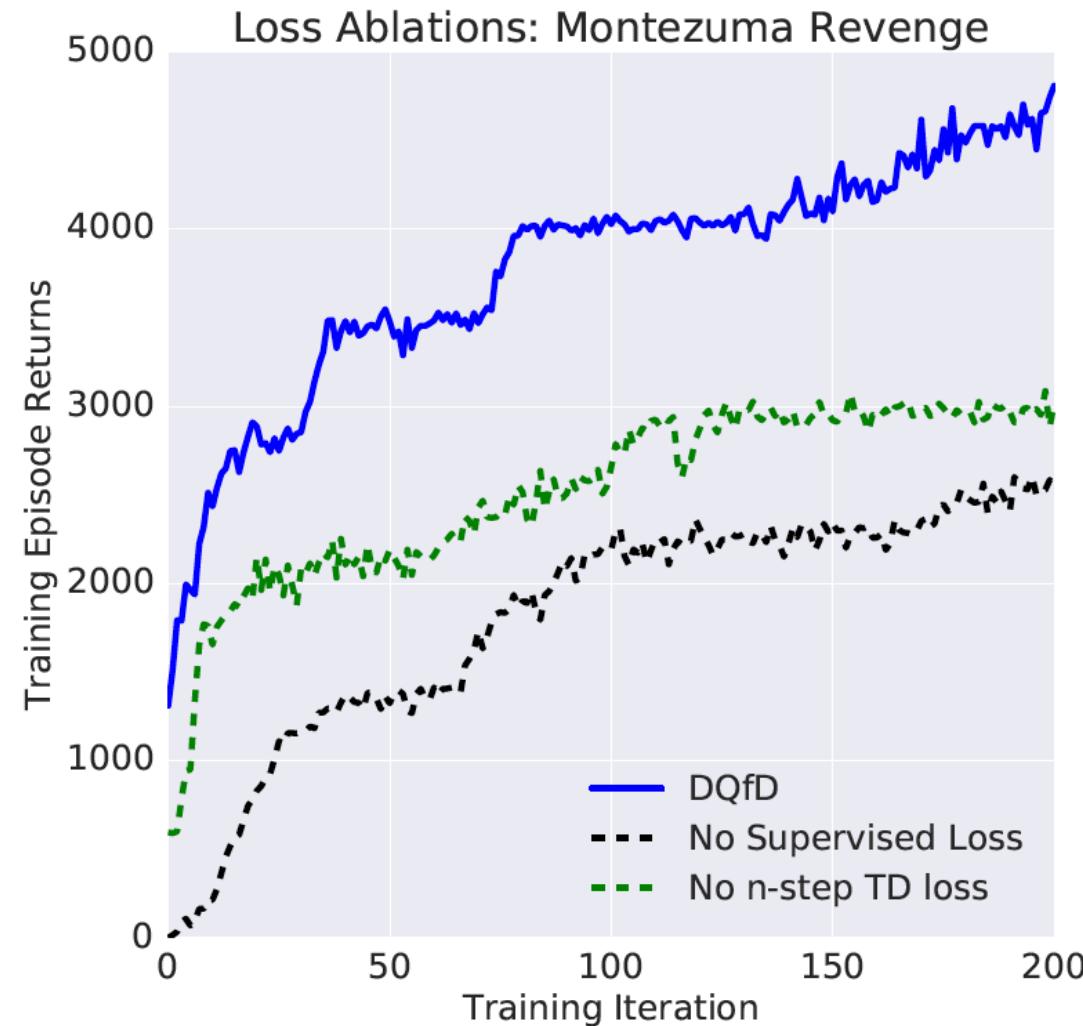
DQfD Replay Buffer Tweak

- 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 experience tuples are out-of-date: **remove oldest self-generated data**

Experiment



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
- Makes use of demonstration data during post-training using prioritized experience replay

Today's agenda

- Respecting demonstrations by a large margin (DQN with demonstrations)
- Demonstrating both actions and Q-values (AggreVaTe)
- Autonomous driving for high-speed racecars
- Conditional imitation learning
- End-to-end neural motion planner

Acknowledgments

Today's slides are based on student presentations from 2019 by: Bryan Chan, Chandripal Budnarain, Renato Fereira, David Acuna, Brenna Li, Wei Cui

AggreVaTe:
Reinforcement and Imitation
Learning via Interactive No-Regret
Learning

CSC 2621
Renato Ferreira Pinto Junior

Stéphane Ross & J. Andrew Bagnell (2014)

Pick one:



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

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*

Algorithm

Algorithm 1 AGGREGATE: 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 $t \in \{1, 2, \dots, T\}$.
 - 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 $\hat{\pi}_{i+1}$)

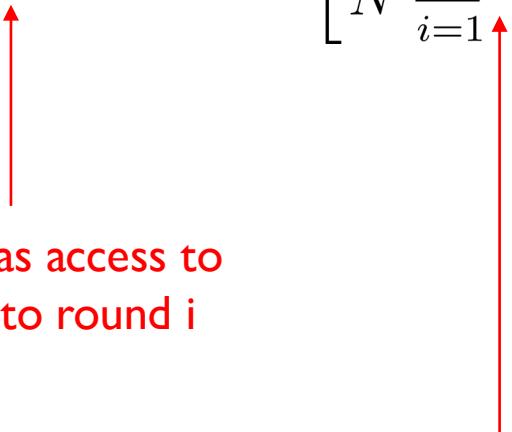
end for

Return best $\hat{\pi}_i$ on validation.

No-Regret Online Learners

Intuition: No matter what the distribution of input data, your online policy/classifier will do asymptotically as well as the best-in-hindsight policy/classifier.

$$r_N = \frac{1}{N} \sum_{i=1}^N l_i(\pi_i) - \min_{\pi \in \Pi} \left[\frac{1}{N} \sum_{i=1}^N l_i(\pi) \right]$$



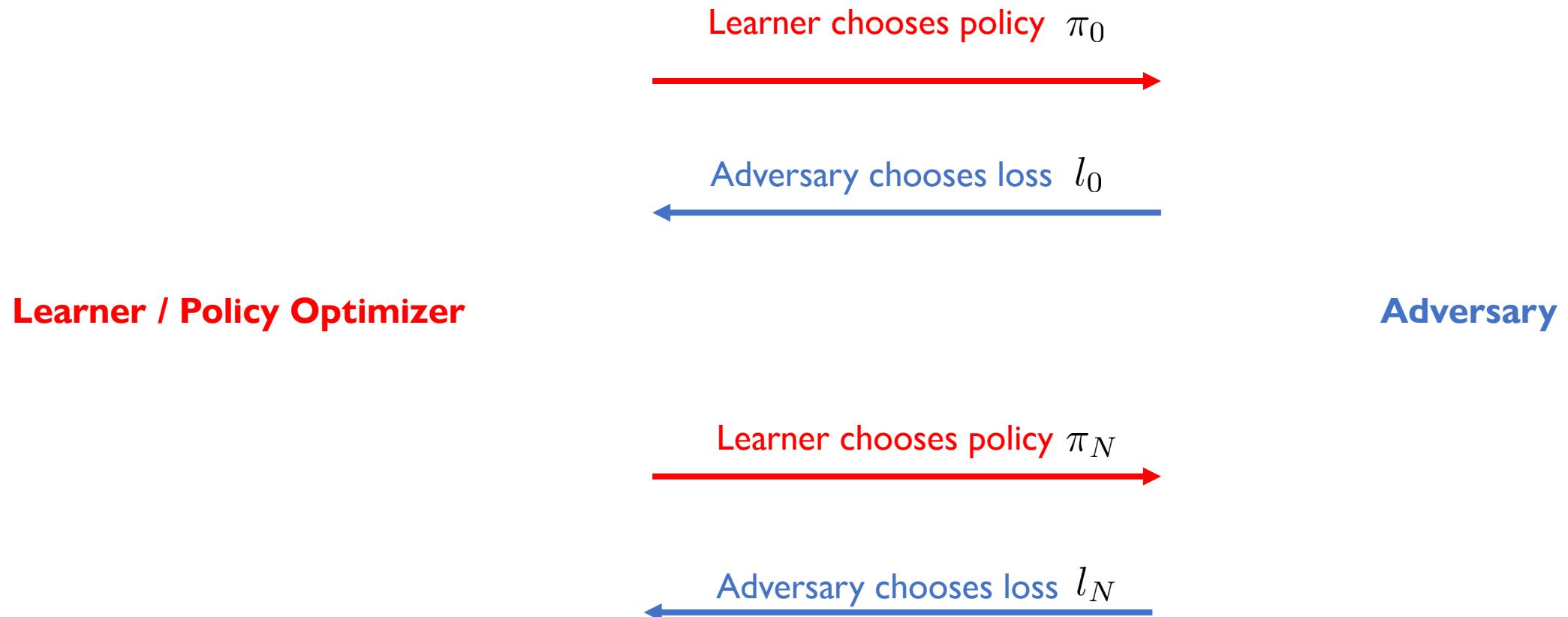
Policy has access to data up to round i

Policy has access to data up to round N

Loss function at round i

No-regret: $\lim_{N \rightarrow \infty} r_N = 0$

Online Learning



Regret $r_N = \frac{1}{N} \sum_{i=1}^N l_i(\pi_i) - \min_{\pi \in \Pi} \left[\frac{1}{N} \sum_{i=1}^N l_i(\pi) \right]$

A no-regret online learning algorithm

Assume the learner has seen losses during rounds 0...t-1. What policy can the learner pick next?

Follow-The-Leader $\pi_t = \operatorname{argmin}_{\pi \in \Pi} \left[\sum_{i=1}^{t-1} l_i(\pi) \right]$ (DAgger is equivalent to FTL)

Theorem (FTL): if Π is convex and the loss function l_i is strongly convex for all i, then we have:

$$r_N = \frac{1}{N} \sum_{i=1}^N l_i(\pi_i) - \min_{\pi \in \Pi} \left[\frac{1}{N} \sum_{i=1}^N l_i(\pi) \right] = O\left(\frac{\log N}{N}\right)$$

Analysis

- Classification regret: best in policy class compared to expert

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


 $a = \pi_i(s)$

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

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

- 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:

Theorem 2.1. After N iterations of AGGREGATE:

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

Thus if a no-regret online algorithm is used to pick the sequence of policies $\hat{\pi}_{1:N}$, then as the number of iterations $N \rightarrow \infty$:

$$\lim_{N \rightarrow \infty} J(\bar{\pi}) \leq J(\pi^*) + T\epsilon_{\text{class}}$$

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!)

Today's agenda

- Respecting demonstrations by a large margin (DQN with demonstrations)
- Demonstrating both actions and Q-values (AggreVaTe)
- Autonomous driving for high-speed racecars
- Conditional imitation learning
- End-to-end neural motion planner

Acknowledgments

Today's slides are based on student presentations from 2019 by: Bryan Chan, Chandripal Budnarain, Renato Fereira, David Acuna, Brenna Li, Wei Cui

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



Auto-Rally car

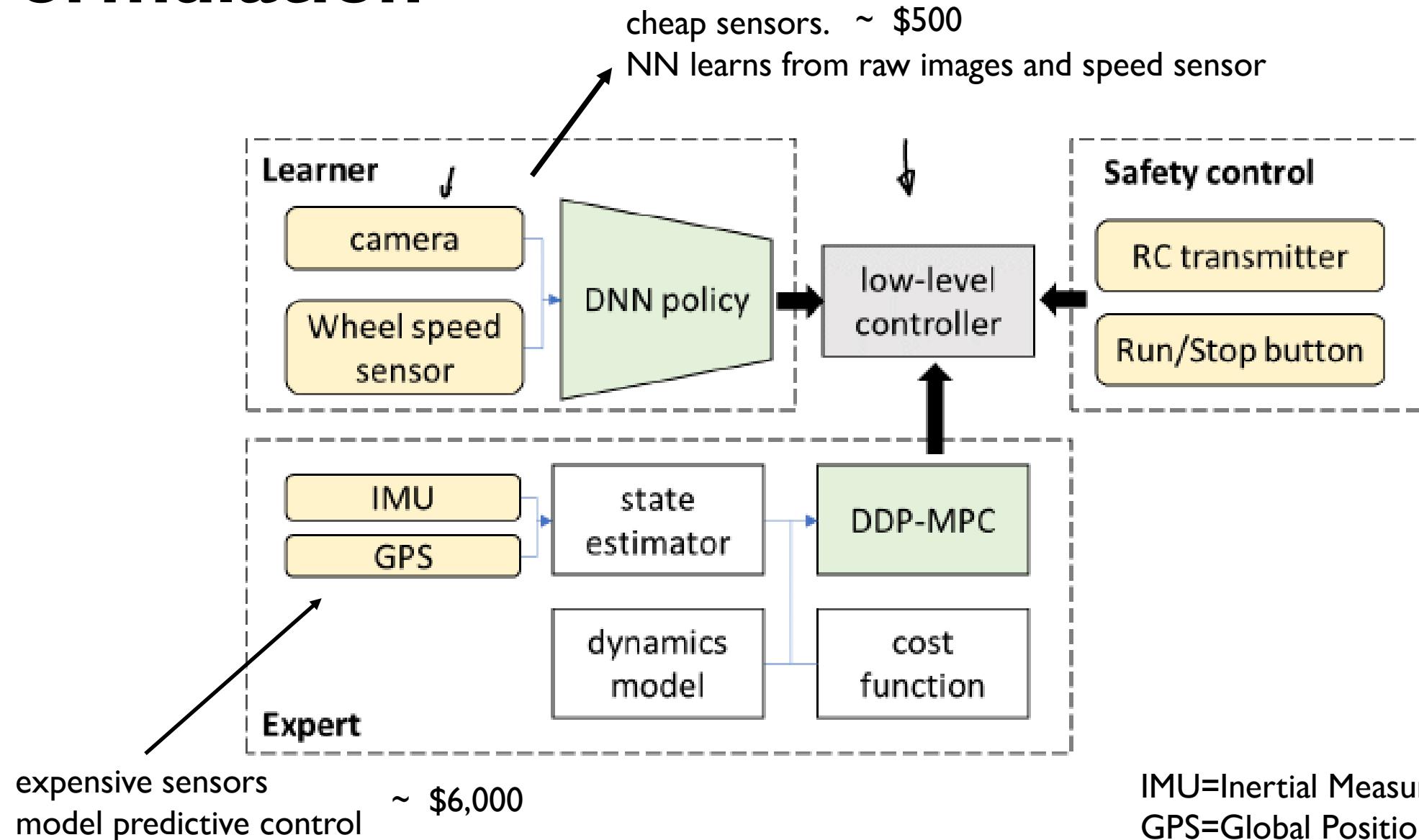


training/test track

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



Problem Formulation



Formulation

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

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

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



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

Advantage of taking action a at state s over what policy dictates

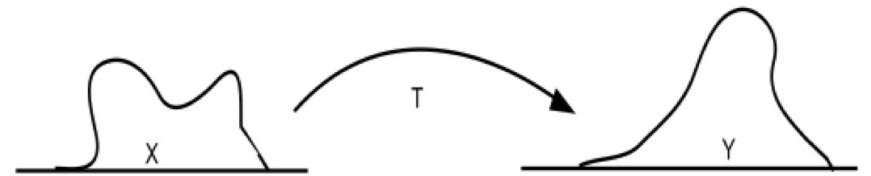
Formulation

$$\begin{aligned} J(\pi) - J(\pi^*) & \xrightarrow{\text{expert}} \\ &= \mathbb{E}_{s,t \sim d_\pi} [\mathbb{E}_{a \sim \pi_s} [Q_{\pi^*}^t(s, a)] - \mathbb{E}_{a^* \sim \pi_s^*} [Q_{\pi^*}^t(s, a^*)]] \end{aligned}$$

Advantage of taking action a at state s over what expert's policy dictates

Wasserstein Distance

$$\begin{aligned} D_W(p, q) &:= \sup_{f: \text{Lip}(f(\cdot)) \leq 1} \mathbb{E}_{x \sim p}[f(x)] - \mathbb{E}_{x \sim q}[f(x)] \\ &= \inf_{\gamma \in \Gamma(p, q)} \int_{\mathcal{M} \times \mathcal{M}} d(x, y) d\gamma(x, y), \end{aligned}$$



Formulation

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

$$= \mathbb{E}_{s,t \sim d_\pi} [\mathbb{E}_{a \sim \pi_s} [Q_{\pi^*}^t(s, a)] - \mathbb{E}_{a^* \sim \pi_s^*} [Q_{\pi^*}^t(s, a^*)]]$$

$$\leq C_{\pi^*} \mathbb{E}_{s,t \sim d_\pi} [D_W(\pi, \pi^*)]$$

$$\leq C_{\pi^*} \mathbb{E}_{s,t \sim d_\pi} \mathbb{E}_{a \sim \pi_s} \mathbb{E}_{a^* \sim \pi_s^*} [\|a - a^*\|],$$



learner policy expert's policy

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



Online Imitation Learning Problem (DAgger)

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

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

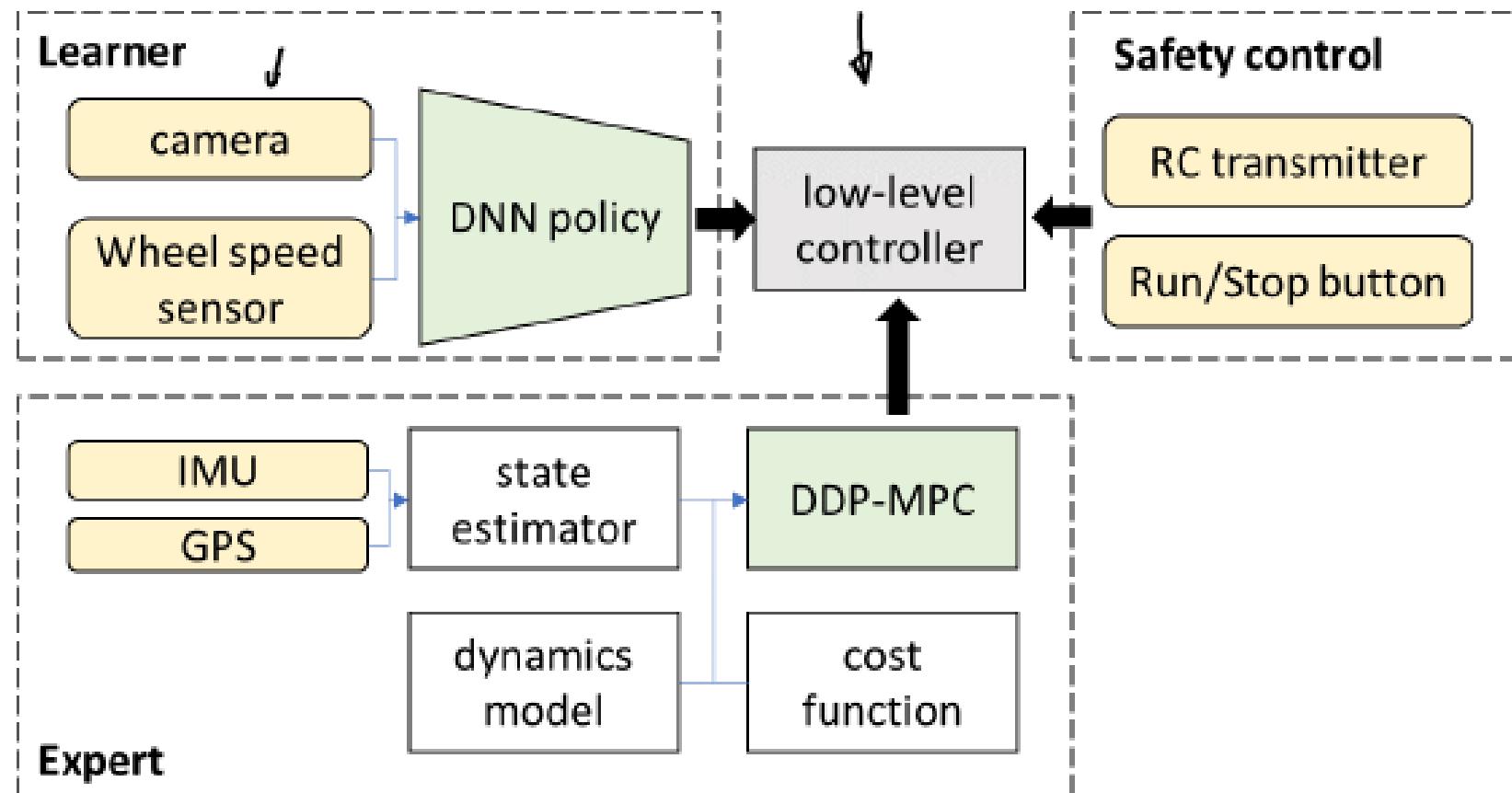
expert policy expert policy

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

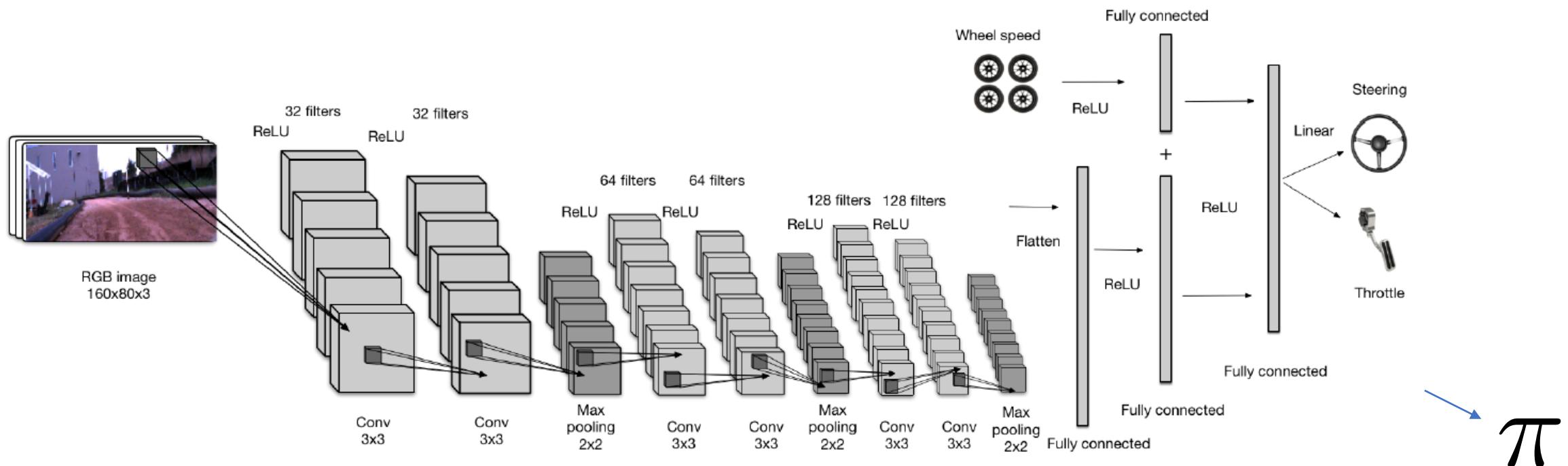


This corresponds to supervised learning (expectation is defined by a fixed policy)

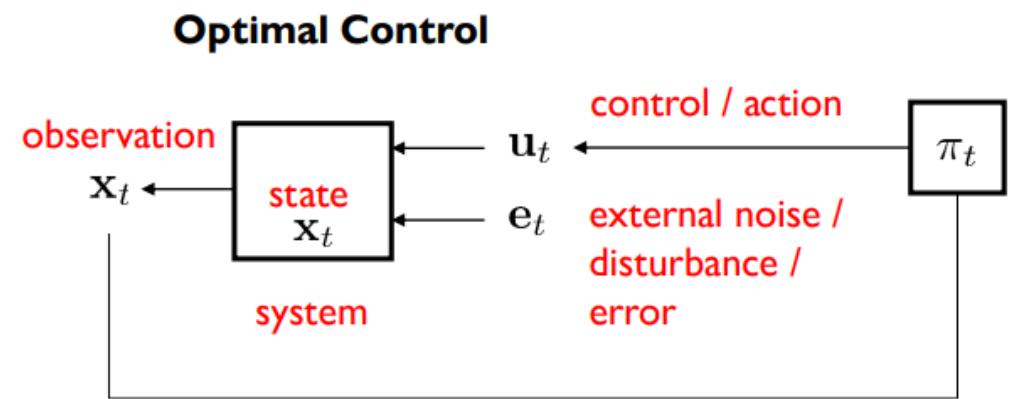
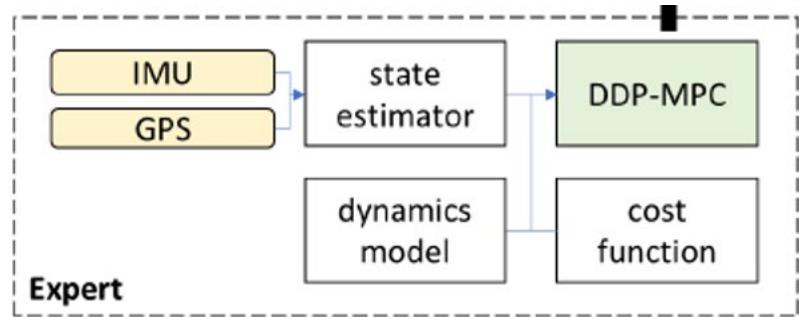
System Diagram



DNN Control Policy



Expert – recall control



$$\underset{\pi_0, \dots, \pi_{T-1}}{\text{minimize}} \quad \mathbb{E}_{\mathbf{e}_t} \left[\sum_{t=0}^T c(\mathbf{x}_t, \mathbf{u}_t) \right]$$

subject to $\mathbf{x}_{t+1} = f_t(\mathbf{x}_t, \mathbf{u}_t, \mathbf{e}_t)$ known dynamics

$$\mathbf{u}_t = \pi_t(\mathbf{x}_{0:t}, \mathbf{u}_{0:t-1})$$

Sparse Spectrum Gaussian Process

control law / policy

Expert – MPC

Differential Dynamic Program (DDP) ~ Recall iLQR

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

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

$$\mathbf{b}_t \quad \mathbf{A}_t \quad \delta\mathbf{x}_t \quad \mathbf{B}_t \quad \delta\mathbf{u}_t$$

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

$$\mathbf{h}_t \quad H_t$$

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 $\bar{\mathbf{x}}_0, \dots, \bar{\mathbf{x}}_N$ and $\bar{\mathbf{u}}_0, \dots, \bar{\mathbf{u}}_N$

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

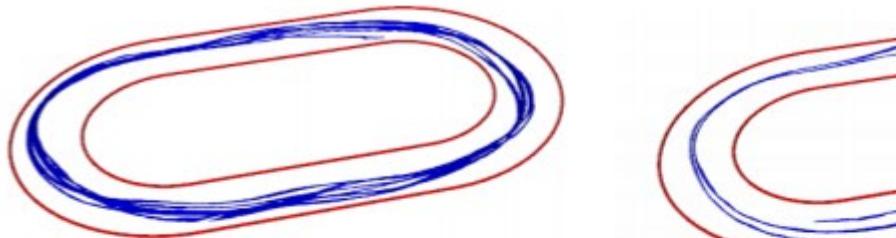


High Speed driving
at 7.5 m/s or 135 km / h

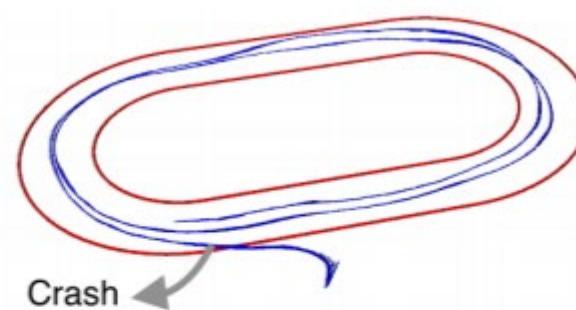
Cost for expert:

$$c(s_t, a_t) = \alpha_1 c_{\text{pos}}(s_t) + \alpha_2 c_{\text{spd}}(s_t) + \alpha_3 c_{\text{slip}}(s_t) + \alpha_4 c_{\text{act}}(a_t)$$

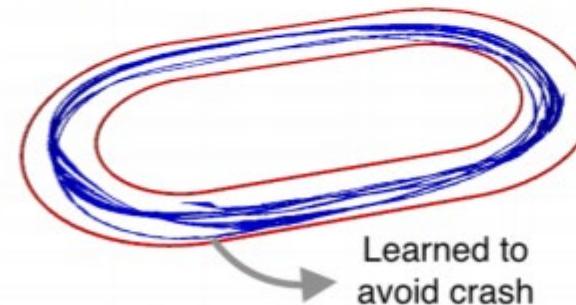
Experiment – learning trajectories



(a) MPC expert.



(b) Batch IL.



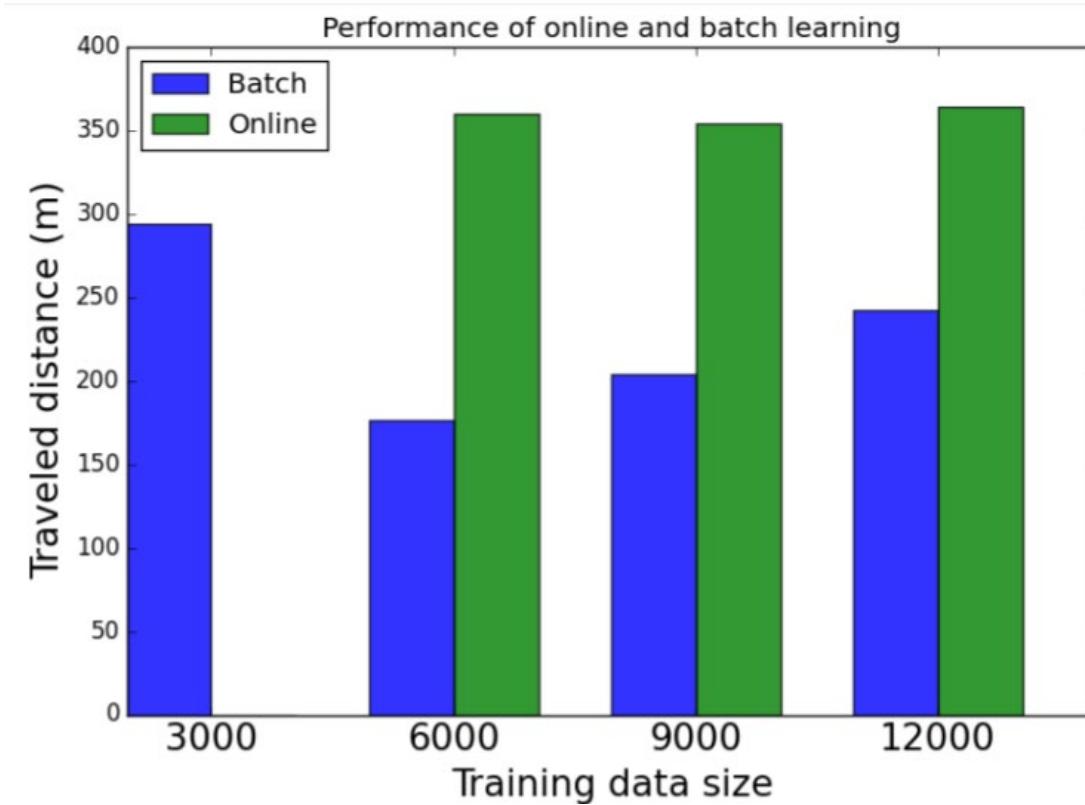
(c) Online IL.

Learned to
avoid crash

Comparing – Loss (to expert)

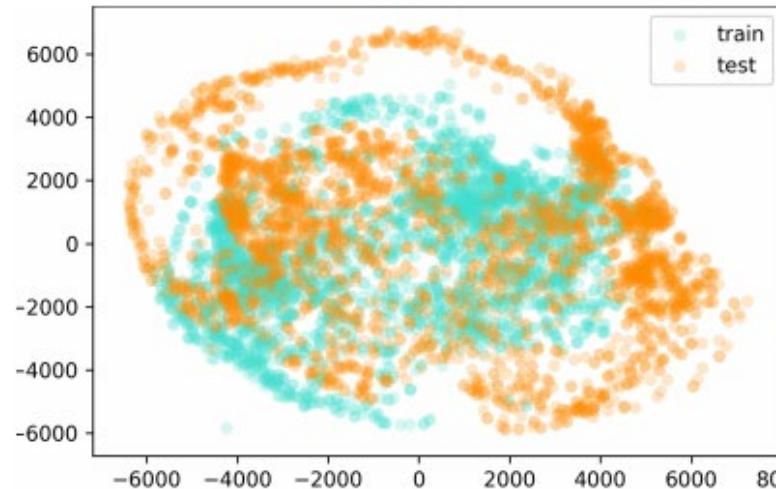
Policy	Avg. speed	Top speed	Training data	Completion ratio	Total loss	Steering/Throttle loss
Expert	6.05 m/s	8.14 m/s	N/A	100 %	0	0
Batch	4.97 m/s	5.51 m/s	3000	100 %	0.108	0.092/0.124
Batch	6.02 m/s	8.18 m/s	6000	51 %	0.108	0.162/0.055
Batch	5.79 m/s	7.78 m/s	9000	53 %	0.123	0.193/0.071
Batch	5.95 m/s	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

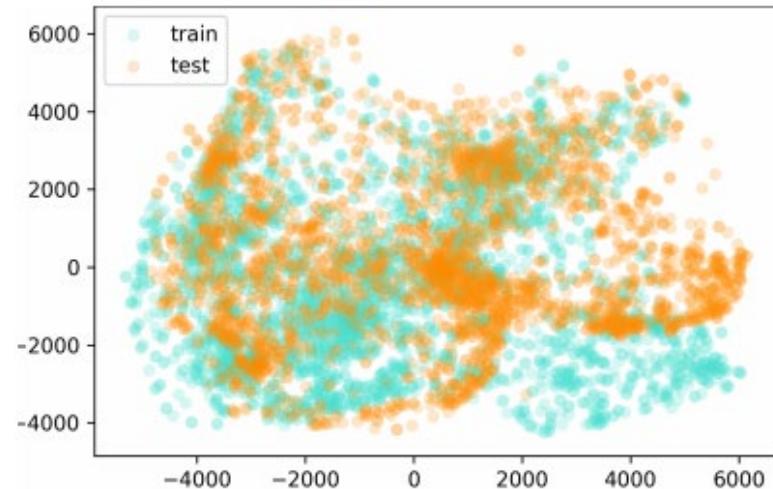


Comparing – generalizability

t-Distributed Stochastic Neighbor Embedding (t-SNE)

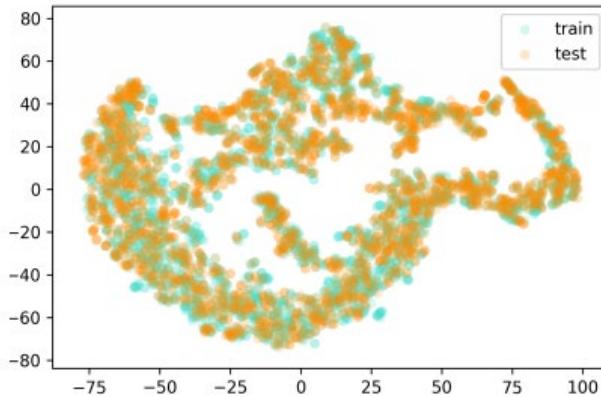


(a) Batch raw image

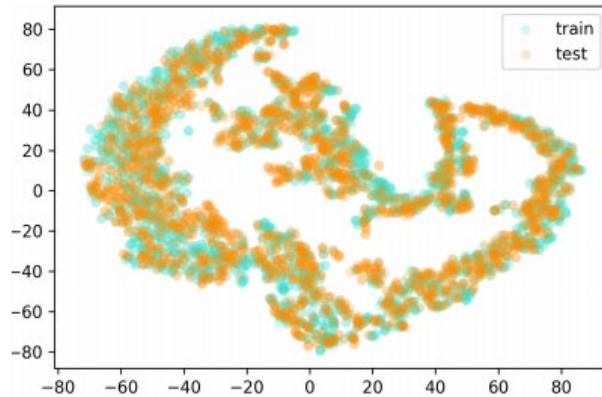


(b) Online raw image

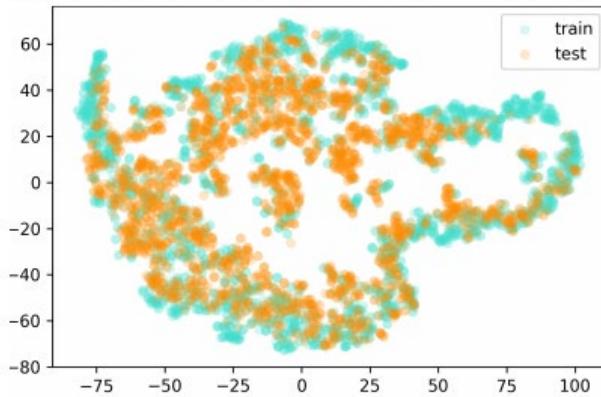
Comparing – generalizability



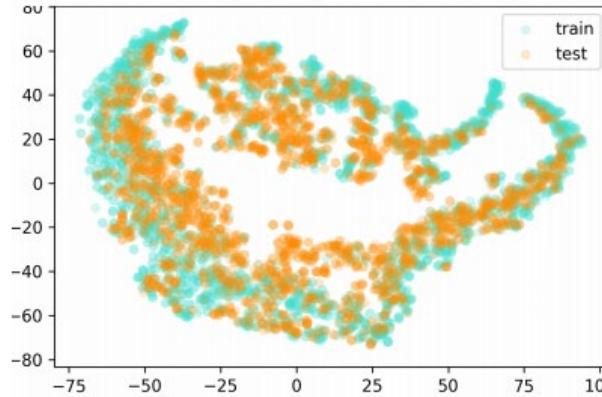
(a) Batch data wrt online model



(b) Online data wrt online model



(c) Batch data wrt batch model

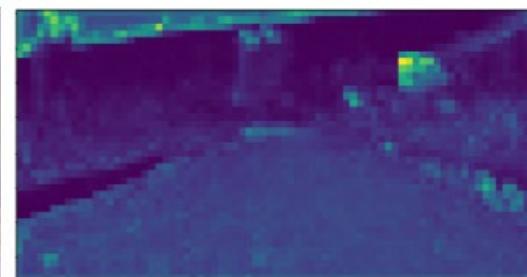


(d) Online data wrt batch model

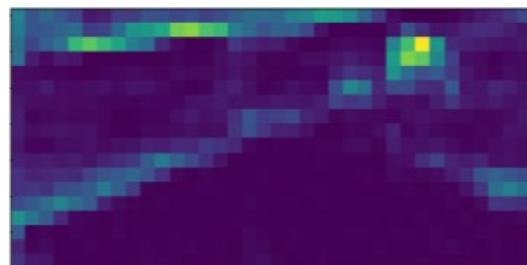
DNN – high and low capture



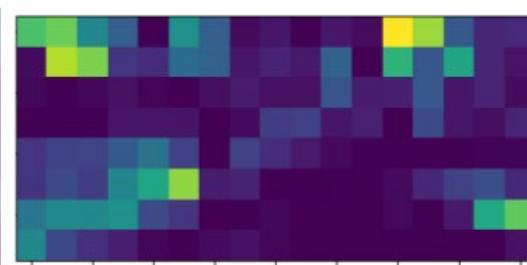
(a) raw image



(b) max-pooling1

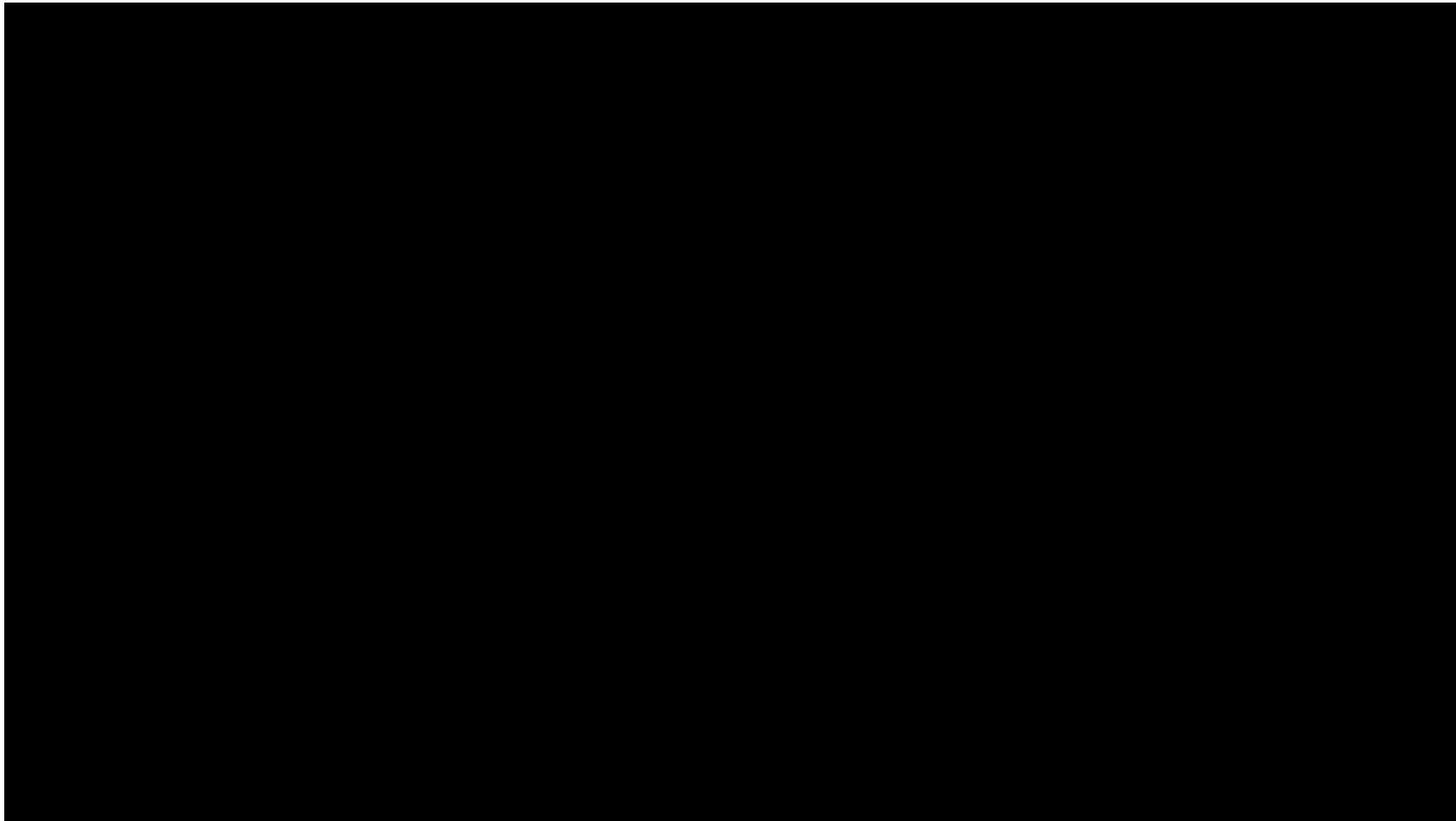


(c) max-pooling2



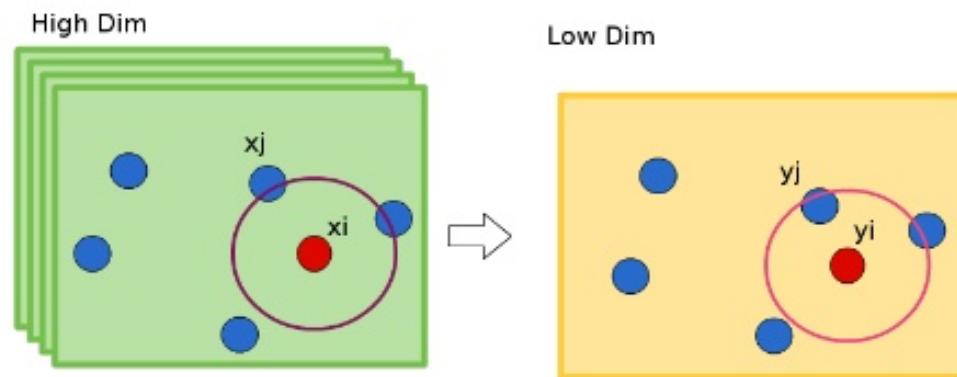
(d) max-pooling3

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



Introduction

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



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

Today's agenda

- Respecting demonstrations by a large margin (DQN with demonstrations)
- Demonstrating both actions and Q-values (AggreVaTe)
- Autonomous driving for high-speed racecars
- **Conditional imitation learning**
- **End-to-end neural motion planner**

Acknowledgments

Today's slides are based on student presentations from 2019 by: Bryan Chan, Chandripal Budnarain, Renato Fereira, David Acuna, Brenna Li, Wei Cui

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



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 :**

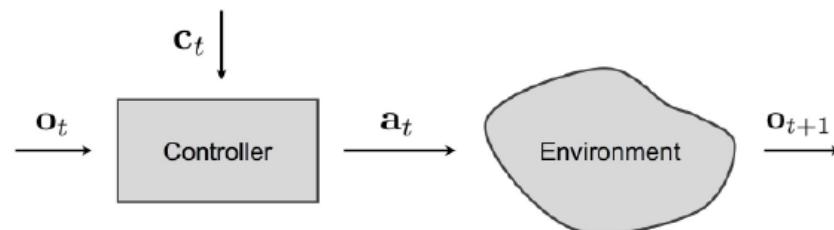
$$\mathbf{a} = [s, \alpha] \quad (1)$$

- ▶ s : steering angle
- ▶ α : acceleration

Conditional Imitation Learning

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

$$\min_{\theta} \sum_i \mathcal{L}(F(\mathbf{o}_i, \mathbf{c}_i; \theta), \mathbf{a}_i) \quad (2)$$

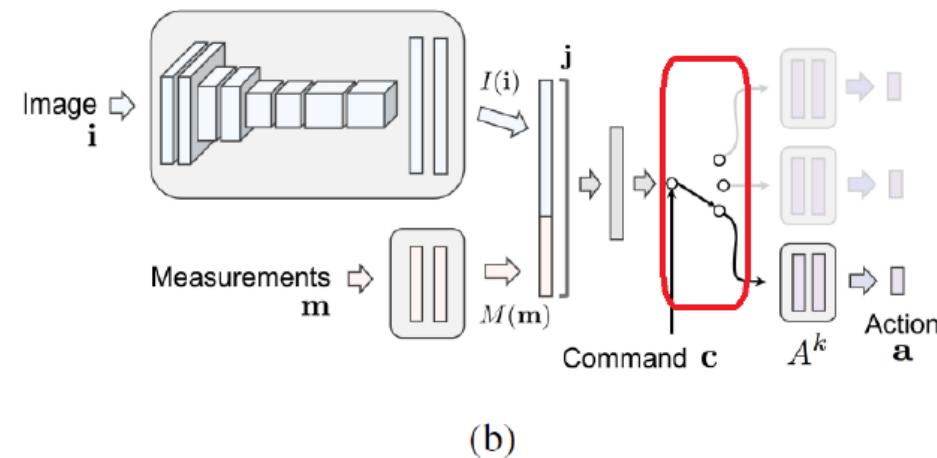
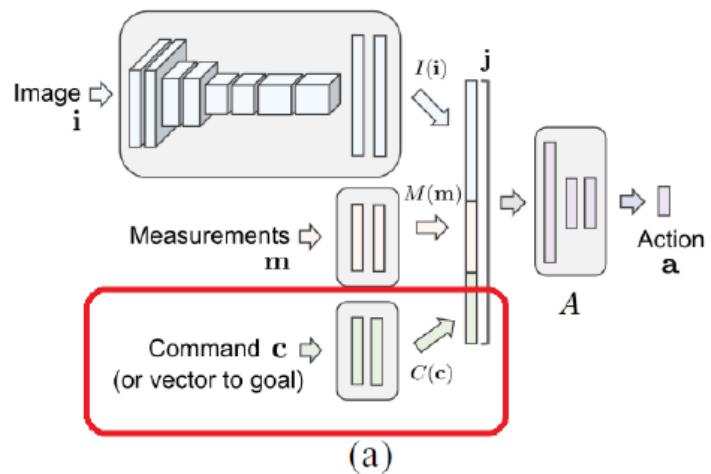


- The high level commands explored for this paper :

$$\mathbf{c} \in \{\text{continue, left, straight, right}\} \quad (3)$$

Network Architecture

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



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.



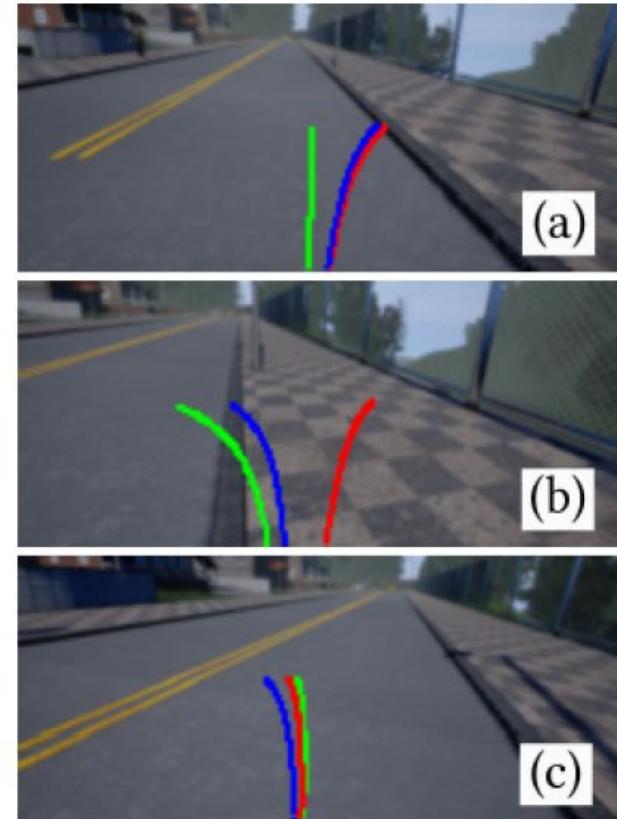
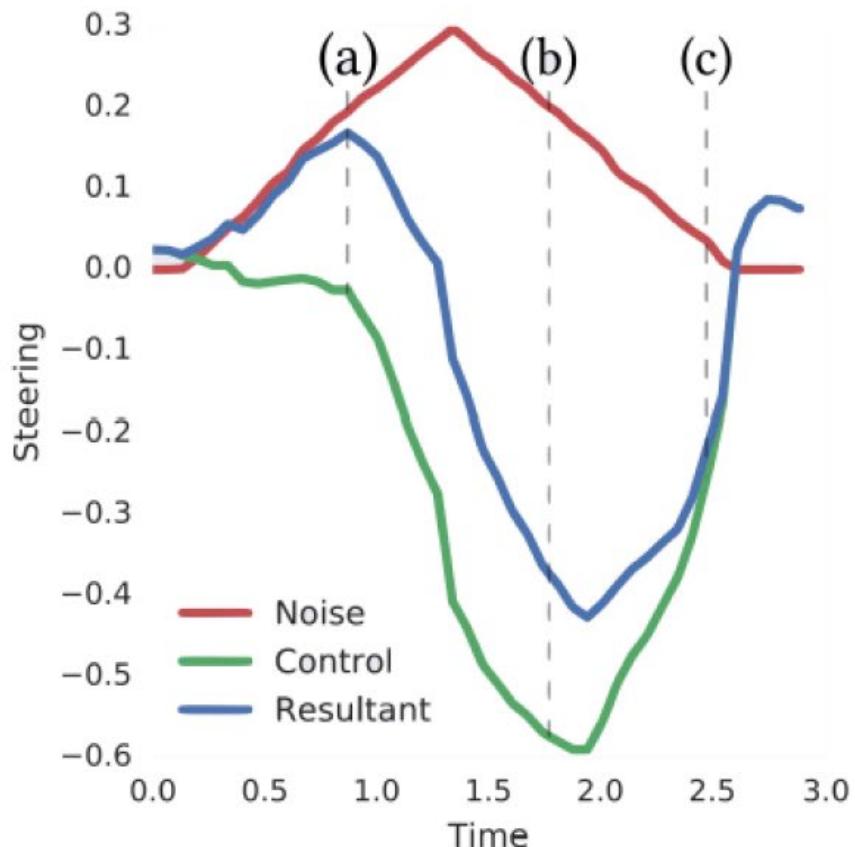
Town 1 (training)



Town 2 (testing)

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 :

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

- 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 : $a = \mathcal{F}(o)$
- **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		Km per infraction	
	Town 1	Town 2	Town 1	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

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.

Today's agenda

- Respecting demonstrations by a large margin (DQN with demonstrations)
- Demonstrating both actions and Q-values (AggreVaTe)
- Autonomous driving for high-speed racecars
- Conditional imitation learning
- End-to-end neural motion planner

Acknowledgments

Today's slides are based on student presentations from 2019 by: Bryan Chan, Chandripal Budnarain, Renato Fereira, David Acuna, Brenna Li, Wei Cui

Main idea

**Jointly optimize
perception and planning**

End-to-end Interpretable Neural Motion Planner

Wenyuan Zeng^{1,2*} Wenjie Luo^{1,2*}

Simon Suo^{1,2} Abbas Sadat¹ Bin Yang^{1,2} Sergio Casas^{1,2} Raquel Urtasun^{1,2}

¹Uber Advanced Technologies Group ²University of Toronto

{wenyuan, wenjie, suo, abbas, byang10, sergio.casas, urtasun}@uber.com

Abstract

In this paper, we propose a neural motion planner for learning to drive autonomously in complex urban scenarios that include traffic-light handling, yielding, and interactions with multiple road-users. Towards this goal, we design a holistic model that takes as input raw LIDAR data and a HD map and produces interpretable intermediate representations in the form of 3D detections and their future trajectories, as well as a cost volume defining the goodness of each position that the self-driving car can take within the planning horizon. We then sample a set of diverse physically possible trajectories and choose the one with the minimum learned cost. Importantly, our cost volume is able to naturally capture multi-modality. We demonstrate the effectiveness of our approach in real-world driving data captured in several cities in North America. Our experiments show that the learned cost volume can generate safer planning than all the baselines.

due to the nature of self-driving control being a sequential decision problem, and requires massive amounts of data to generalize. Furthermore, interpretability is difficult to obtain for analyzing the mistakes of the network. It is also hard to incorporate sophisticated prior knowledge about the scene, e.g. that vehicles should not collide.

In contrast, most self-driving car companies, utilize a *traditional engineering stack*, where the problem is divided into subtasks: perception, prediction, motion planning and control. Perception is in charge of estimating all actors' positions and motions, given the current and past evidences. This involves solving tasks such as 3D object detection and tracking. Prediction¹, on the other hand, tackles the problem of estimating the future positions of all actors as well as their intentions (e.g., changing lanes, parking). Finally, motion planning takes the output from previous stacks and generates a safe trajectory for the SDV to execute via a control system. This framework has interpretable intermediate representations by construction, and prior knowledge can be easily exploited, for example in the form of high definition

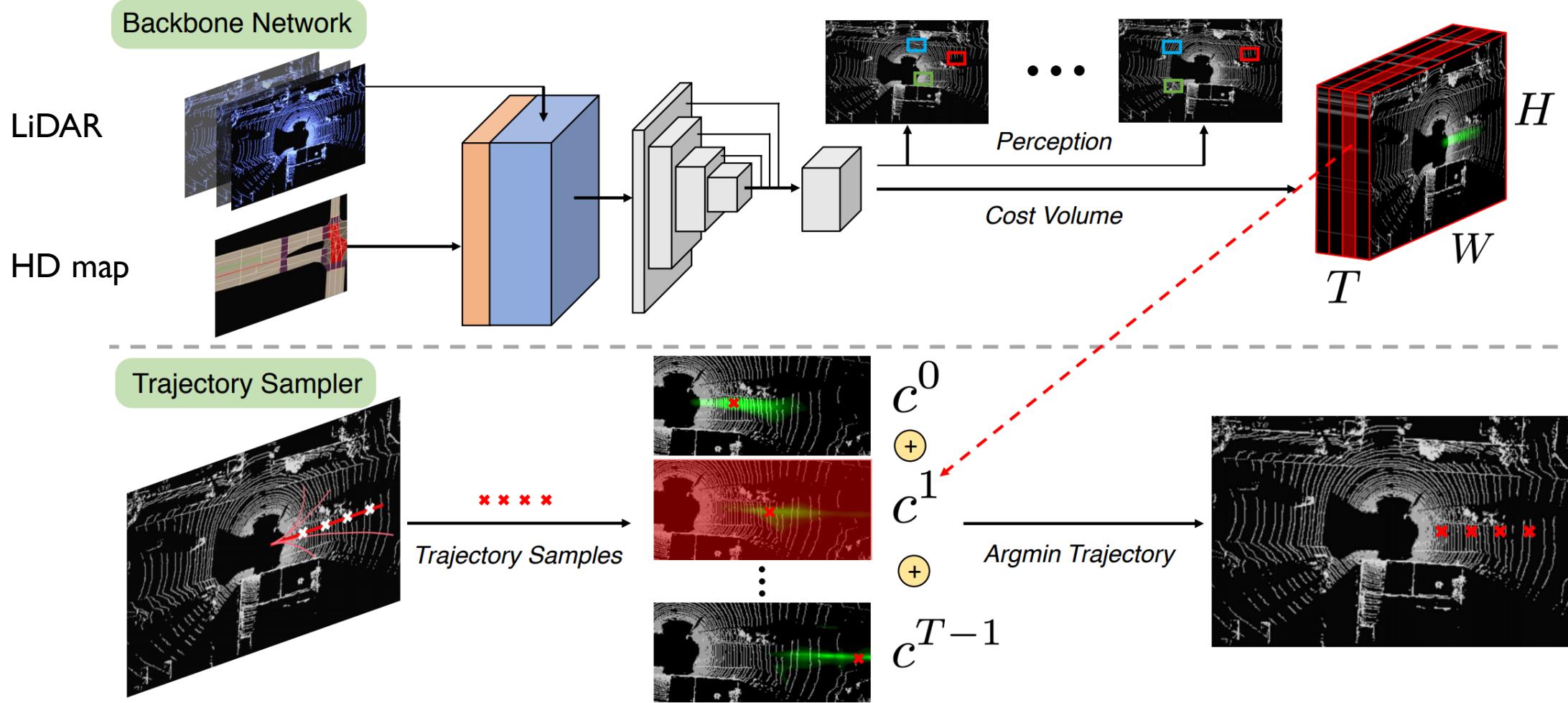


Figure 1. Our end-to-end interpretable neural motion planner. Backbone network takes LiDAR data and maps as inputs, and outputs bounding boxes of other actors for future timesteps (perception), as well as a cost volume for planning with T filters. Next, for each trajectory proposal from the sampler, its cost is indexed from different filters of the cost volume and summed together. The trajectory with the minimal cost will be our final planning.

Perception and planning loss functions

Perception Loss: Our perception loss includes classification loss, for distinguishing a vehicle from the background, and regression loss, for generating precise object bounding boxes. For each predefined anchor box, the network outputs a classification score as well as several regression targets. This classification score $p_{i,j}^k$ indicates the probability of existence of a vehicle at this anchor. We employ a cross-entropy loss for the classification defined as

$$\mathcal{L}_{cla} = \sum_{i,j,k} (q_{i,j}^k \log p_{i,j}^k + (1 - q_{i,j}^k) \log(1 - p_{i,j}^k)) , \quad (6)$$

where $q_{i,j}^k$ is the class label for this anchor (i.e., $q_{i,j}^k = 1$ for vehicle and 0 for background). The regression outputs include information of position, shape and heading angle at each time frame t , namely

Planning Loss: Learning a reasonable cost volume is challenging as we do not have ground-truth. To overcome this difficulty, we minimize the max-margin loss where we use the ground-truth trajectory as a positive example, and randomly sampled trajectories as negative examples. The intuition behind is to encourage the ground-truth trajectory to have the minimal cost, and others to have higher costs. More specifically, assume we have a ground-truth trajectory $\{(x^t, y^t)\}$ for the next T time steps, where (x^t, y^t) is the position of our vehicle at the t time step. Define the cost volume value at this point (x^t, y^t) as \hat{c}^t . Then, we sample N negative trajectories, the i^{th} among which is $\{(x_i^t, y_i^t)\}$ and the cost volume value at these points are c_i^t . The sampling procedure for negative trajectories is similar as we described in Section. 3.2, except there is 0.8 probability that the negative sample doesn't obey SDV's initial states, e.g. we randomly sample a velocity to replace SDV's initial velocity. This will provide easier negative examples for the model to start with. The overall max-margin loss is defined as

$$\mathcal{L}_{planning} = \sum_{\{(x^t, y^t)\}} \left(\max_{1 \leq i \leq N} \left(\sum_{t=1}^T [\hat{c}^t - c_i^t + d_i^t + \gamma_i^t]_+ \right) \right)$$

Results

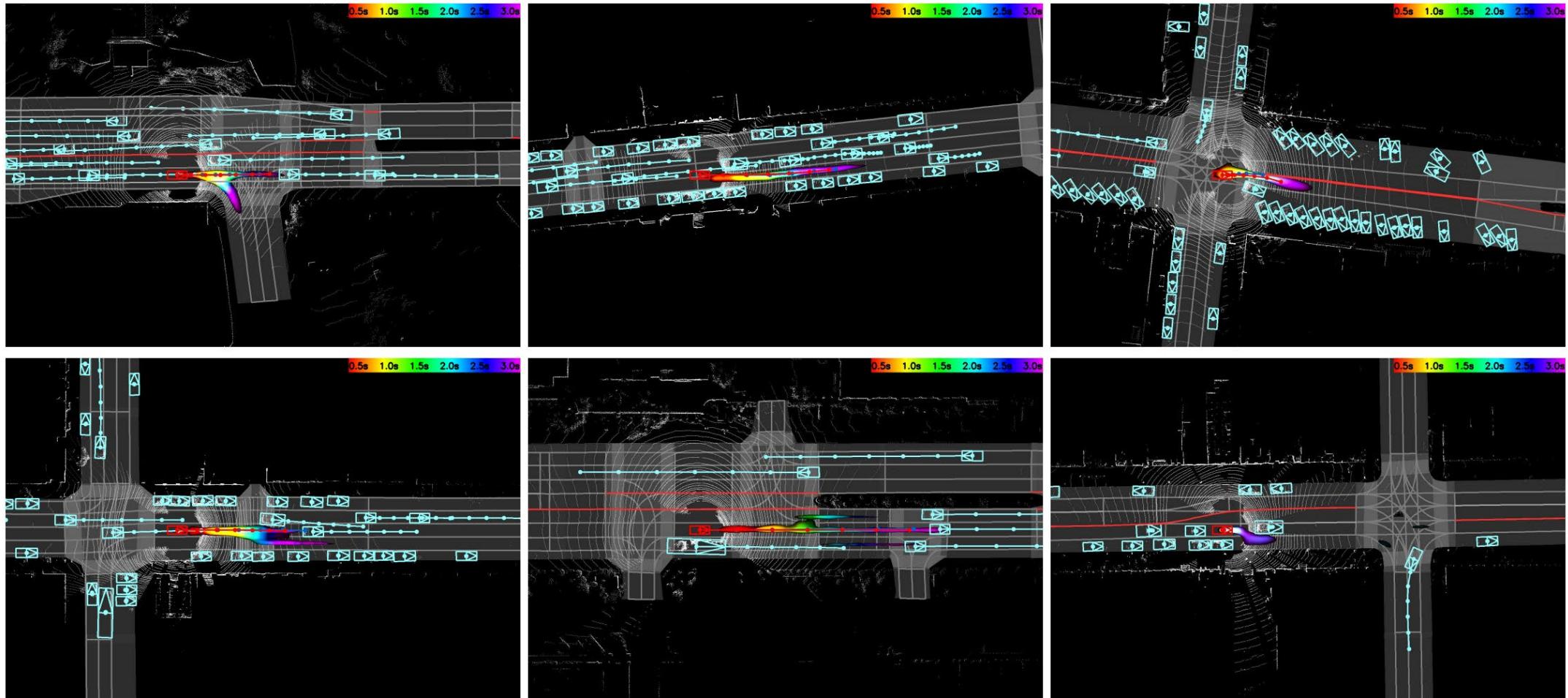


Figure 3. **Cost Volume across Time** We show planned trajectory in red and ground-truth in blue. We overlay lower cost region for different timesteps in the same figure, using different colors (indicated by legend). Detection and corresponding prediction results are in cyan. (best view in color)

Results

Method	L2 along trajectory (m)				L2 across trajectory (m)				L1 (m)			L2 (m)				
	0s	1s	2s	3s	0s	1s	2s	3s	0s	1s	2s	3s	0s	1s	2s	3s
FaF[18]	0.29	0.49	0.87	1.52	0.16	0.23	0.39	0.58	0.45	0.72	1.31	2.14	0.37	0.60	1.11	1.82
IntentNet[5]	0.23	0.42	0.79	1.27	0.16	0.21	0.32	0.48	0.39	0.61	1.09	1.79	0.32	0.51	0.93	1.52
Ours	0.21	0.37	0.69	1.15	0.12	0.16	0.25	0.37	0.34	0.54	0.94	1.52	0.28	0.45	0.80	1.31

Table 2. Motion Forecasting Metric

Method	Detection mAP @ IoU (pts ≥ 1)				
	0.5	0.6	0.7	0.8	0.9
MobileNet[13]	86.1	78.3	60.4	27.5	1.1
FaF[18]	89.8	82.5	68.1	35.8	2.5
IntentNet[5]	94.4	89.4	75.4	43.5	3.9
Pixor[32]	93.4	89.4	78.8	52.2	7.6
Ours	94.2	90.8	81.1	53.7	7.1

Table 4. Detection mAP Result

LiDAR 3D object detection