A Brief Introduction to Deep Reinforcement Learning

Wentao Bao

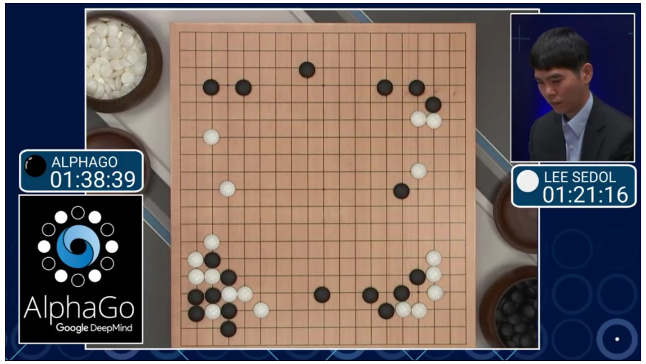
April 14, 2022

Outline

- Foundations of DRL
- DRL Algorithm: DQN and SAC
- DRL Application: Traffic Accident Anticipation

Foundations of DRL

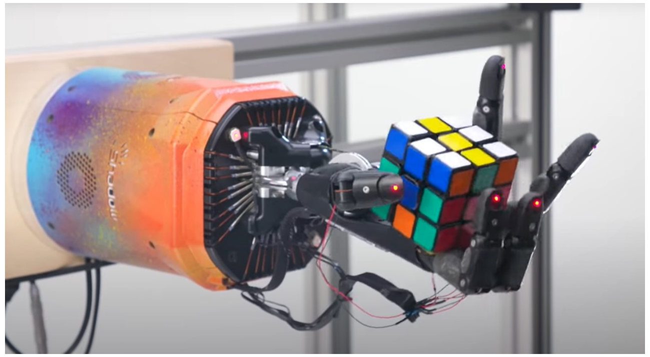
□ What Can DRL Do?



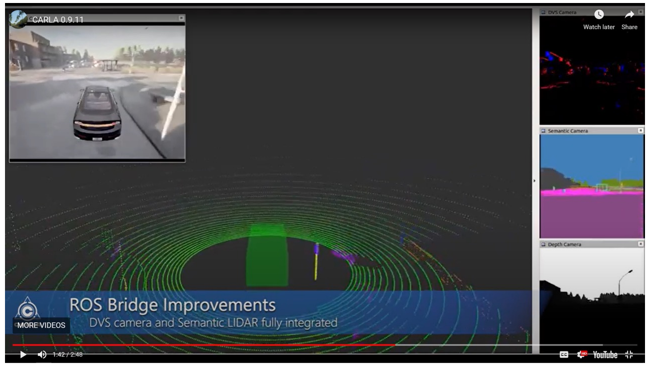
AlphaGo for Go game
(DeepMind, 2016)

TStarBots for StarCraft2
(Tencent AI, 2018)

ReBel for Texas Hold'em
(Facebook AI, 2020)



Solving the Rubik's Cube
(OpenAI, 2019)



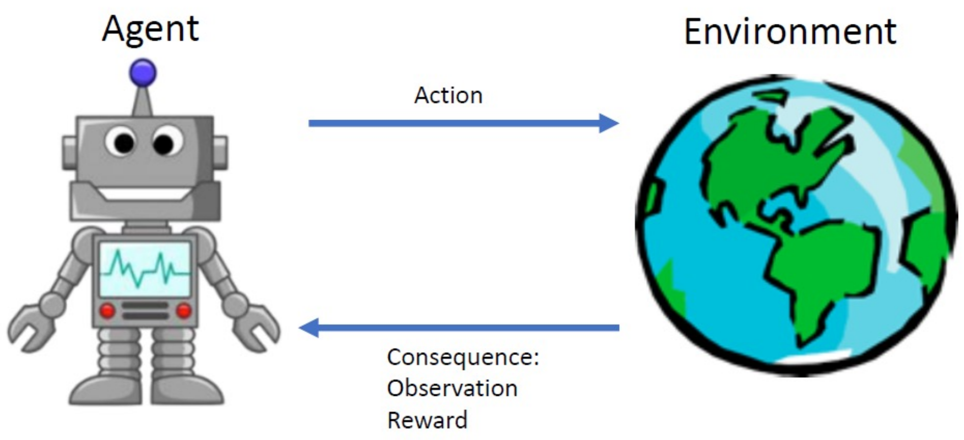
CARLA for self-driving
(Toyota TRI, on-going)

• • •

Foundations of DRL

☐ What is Reinforcement Learning (RL)?

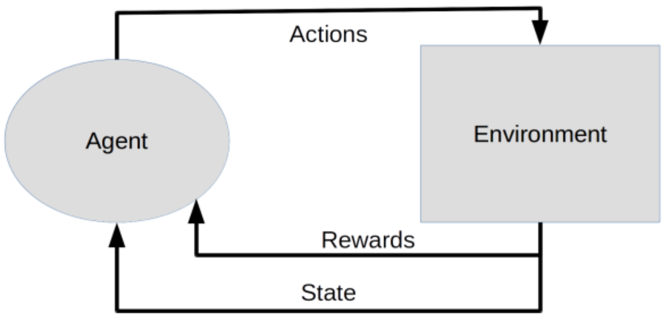
- The **agent learns to take actions** for a **long-term goal** by interacting with the **environment**.
- Basic Elements (System):
  - ✓ **Agent**: the model (e.g., an AI player in video games.)
  - ✓ **Environment**: the world to be explored (e.g., checkerboard)
  - ✓ **Goal**: e.g., win/fail



Foundations of DRL

What is Reinforcement Learning (RL)?

- The **agent learns to take actions** for a **long-term goal** by interacting with the **environment**.
- Main Elements (System):
  - ✓ **State**: any useful information about the agent and environment, e.g.,
    - Go game: maximumly 3^{361} checkerboard states
    - Self-driving: the car's location/velocity/acceleration, etc., and the traffic scene.
  - ✓ **Action**: how the agent will do in each step, e.g.,
    - Go game: take a location in a checkerboard
    - Self-driving: brake/accelerate/make a turn for a self-driving car.
  - ✓ **Reward**: the instant feedback (scalar value) after taking the action , e.g.,
    - Go game: long-term rewards [0, 0, …, 0, 1] (win the game).
    - Self-driving: arrive at the destination.



- DRL Output

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

Foundations of DRL

What is Reinforcement Learning (RL)?

- The **agent learns to take actions** for a **long-term goal** by interacting with the **environment**.
- Core Elements (Algorithm):

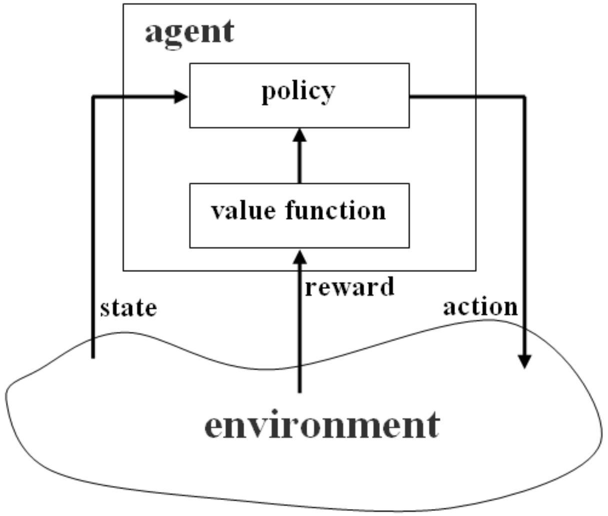
✓ **Policy**: a function π: $\mathbb{R}^D \to \mathbb{R}^d$ that tells how to take an action under a certain state,

- Formula: $\boldsymbol{a} = \pi(\boldsymbol{s}; \theta)$
- Instantiated by Deep Neural Networks in DRL.

✓ **Value**: a function V: $\mathbb{R}^D \to \mathbb{R}$ that evaluates the quality of an action (or action-state pair).,

- State Value Function: $v = V(\boldsymbol{s}; \varphi)$, or
- State-Action Value Function: $q = Q(\boldsymbol{s}, \boldsymbol{a}; \varphi)$
- Instantiated by Deep Neural Networks in DRL.

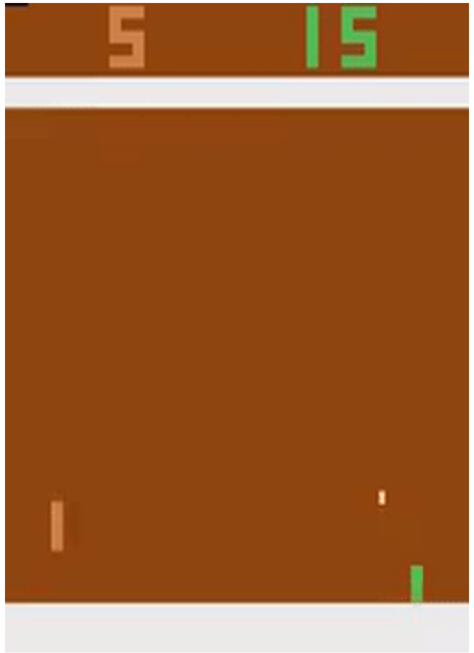
✓ Value function determines how a policy function is learned.



Foundations of DRL

Atari Pong: a video game example

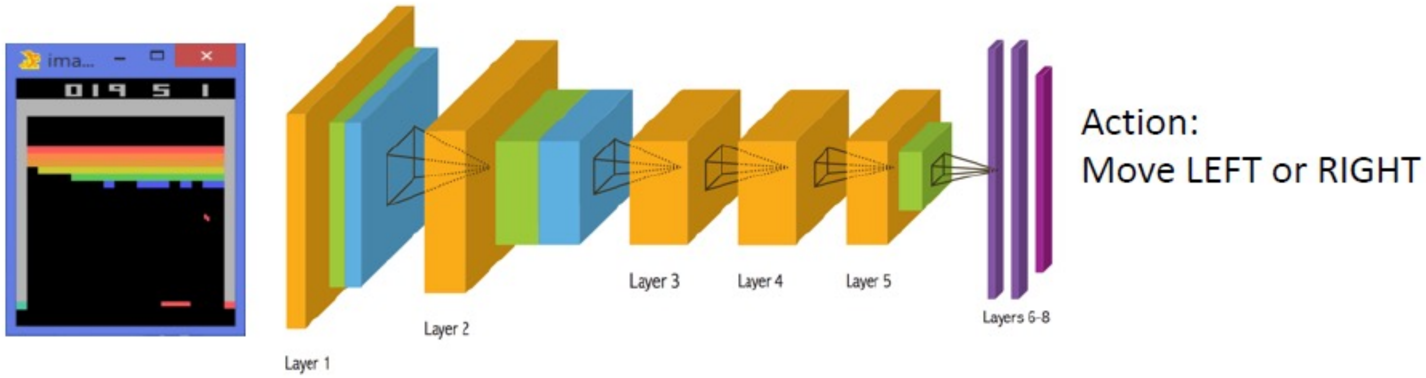
- Rule: hit the ball by moving the pad up or down.
- Basic Elements:
  - ✓ **Agent**: the AI player that controls the pad on the right.
  - ✓ **Environment:** the raw pixels at each time step.
  - ✓ **Goal**: get higher final score than the opponent.
- Main Elements:
  - ✓ **Action**: move UP / DOWN.
  - ✓ **State**: raw pixels, CNN features, location and speed of the pad, etc
  - ✓ **Reward**: scalar value, e.g., +1 (win), or -1 (lose)



Foundations of DRL

Core Elements

- **Policy Function**
  - Stochastic Policy: $\pi(a|s) = P(A_t = a|S_t = s)$.
  - Deterministic Policy: $\mu(a|s) = \underset{a}{\operatorname{argmax}}\, \pi(a|s)$



Foundations of DRL

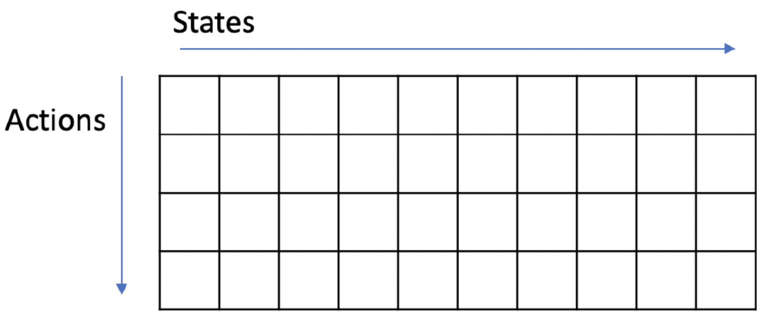
Core Elements

- **Value Function**
  - Can be decomposed into the immediate reward plus discounted value of successive future states
  - **Bellman Equation** of State Value:

$$v^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma v^{\pi}(s_{t+1}) | s_t = s]$$

  - **Bellman Equation** of State-Action value (Q-function):

$$q^{\pi}(s, a) = E_{\pi}[R_{t+1} + \gamma q^{\pi}(s_{t+1}, A_{t+1}) | s_t = s, A_t = a]$$



Q-table

Foundations of DRL

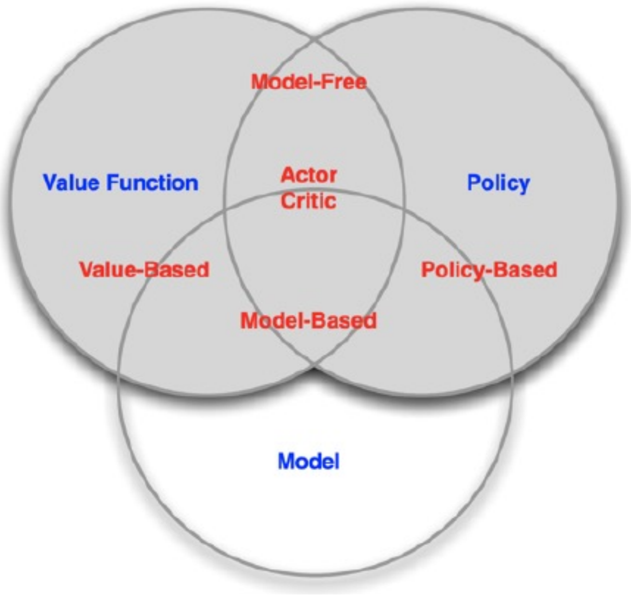
DRL Algorithms

- DRL Objective: Find a policy to maximize the total expected reward

$$\max_{\emptyset} \sum_t \mathbb{E}_{(s_t, a_t)\sim p_\pi}[r(s_t, a_t)]$$

- Based on what the DRL agent explicitly learns:
  - **Policy-based:** policy (explicitly learned); no value function.
  - **Value-based:** value (explicitly learned); policy (implicitly derived)
  - **Actor-Critic Method:** learn both policy and value explicitly.

- Based on if the DRL agent learns an environment model.
  - **Model-based:** the state transition is given/predicted.
  - **Model-free:** state transition is sampled from experience



Credit to: David Silver

Foundations of DRL

Code Example

- With existing libraries, implementation is simple!
  - OpenAI gym library

```python
import gym

env = gym.make("Taxi-v2")
observation = env.reset()
agent = load_agent()

for step in range(10000):
    action = agent(observation)
    observation, reward, done, info = env.step(action)

```

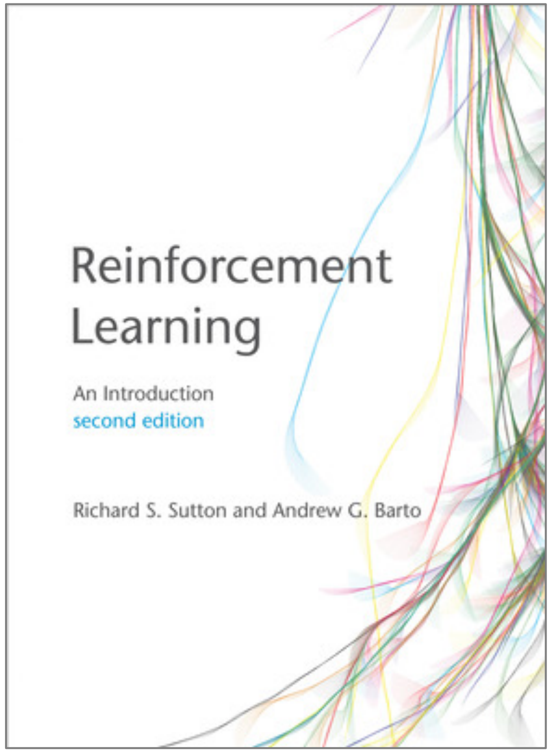
Online image

- A complete PyTorch example of DQN:
  - https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

Foundations of DRL

☐ Recommended Resource

- Books
  - Richard Sutton's book (2d edition)
  - Online version: http://incompleteideas.net/book/RLbook2020.pdf
- Online Courses
  - Berkeley CS-285: http://rail.eecs.berkeley.edu/deeprlcourse
  - Stanford CS-234: https://web.stanford.edu/class/cs234
- Open source RL libraries:
  - Ray/RLlib (TF/PT, 20k): https://github.com/ray-project/ray/tree/master/rllib
  - OpenAI Baselines (TF, 12.5k): https://github.com/openai/baselines
  - PyTorch DRL (PT, 4.2k): https://github.com/p-christ/Deep-Reinforcement-Learning-Algorithms-with-PyTorch
  - THU Tianshou (PT, 4.5k): https://github.com/thu-ml/tianshou
  - RLpyt (PT, 2k): https://github.com/astooke/rlpyt



DRL Algorithm: DQN

Published: 25 February 2015

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

Nature **518**, 529–533 (2015) | Cite this article

407k Accesses | **8728** Citations | **1559** Altmetric | Metrics

- The most impactful DRL algorithm till now (18.9K+ GS citations)
- The first work that combines RL and DNNs.

https://www.nature.com/articles/nature14236

DRL Algorithm: DQN

Deep Q-learning

- Given a state, the optimal policy $\pi^*(\boldsymbol{s})$ can be determined by maximizing the optimal Q-value.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s, a)$$

- To learn the $Q^*(s, a)$, <u>*value function approximation*</u> is introduced by using DNNs

$$\hat{Q}(s, a, w) \approx Q_\pi(s, a)$$

- Using the Bellman equation, such an approximation is achieved by minimizing the temporal difference (TD) error

$$Q^\pi(s, a) = r + \gamma Q^\pi(s', \pi(s')) \qquad \text{Bellman Equation}$$

$$\delta = Q(s, a) - (r + \gamma \max_a Q(s', a)) \qquad \text{TD Error}$$

- Eventually, Q-learning aims to minimize the average TD error by SGD optimizer

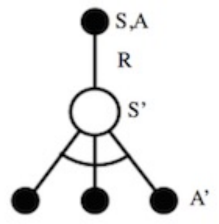
$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r)\,\in\, B} \mathcal{L}(\delta) \qquad \mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \le 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

DRL Algorithm: DQN

Pros and Cons of Q-learning

- Q-target: $R_{t+1} + \gamma max_{a'} Q(S_{t+1}, a')$
- **Pros: Off-policy**

  ✓ The policy to update Q-function (evaluation policy) is different to the policy used to produce action samples (behavior policy)



Behavior policy: ε-greedy (ε probability to select a_{t+1})

Evaluation policy: greedy (by max $Q(S_{t+1}, a')$)

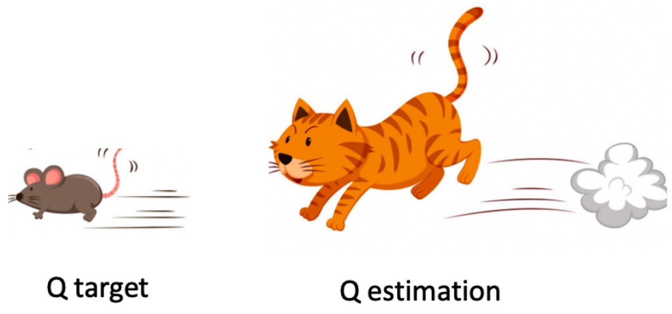
$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S,A) \right)$$

- **Cons:**

  ✓ The target Q is the same as the the Q to be optimized!

  e.g., a cat is chasing a string tied to itself surrounding a table.

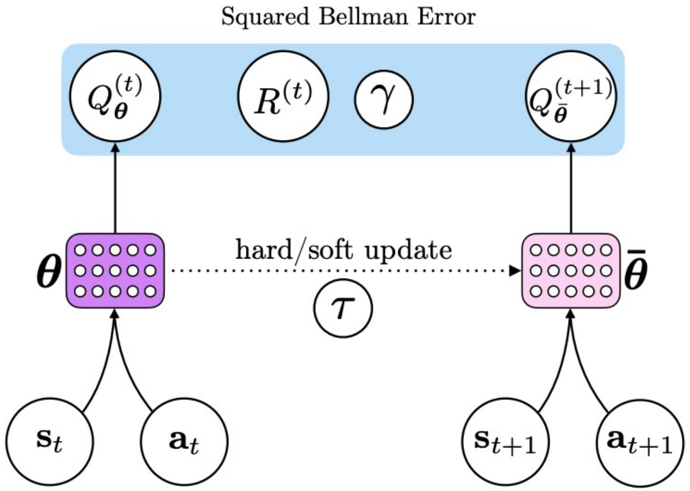
  ✓ Imagine that your labels are always changing in supervised training.



DRL Algorithm: DQN

❑ How does DQN improve the Q-learning?

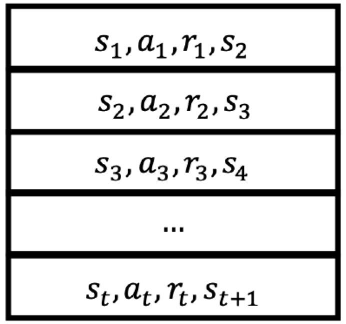
- **Target Q Network**: parameters are updated delayed.
  - ✓ The "mouse" is kept fixed for a period of time, so that the "cat" could catch up.



Update the Target Q Network weights by **running average**

$$\bar{\boldsymbol{\theta}} \leftarrow (1-\tau)\bar{\boldsymbol{\theta}} + \tau\boldsymbol{\theta},$$

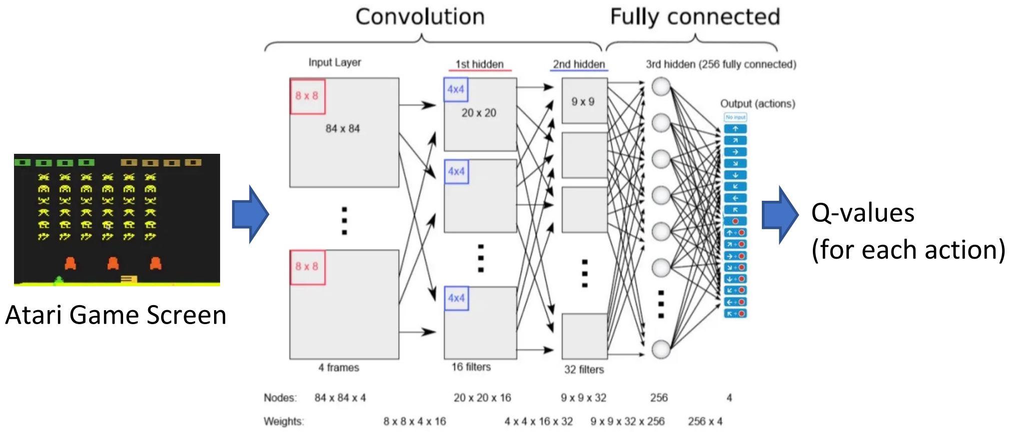
- **Experience Replay:** stabilize the training by reducing the sample correlation
  - ✓ Store the transition (s_t, a_t, r_t, s_{t+1}) in a large replay buffer, from which samples are randomly drawn to update the neural networks.



DRL Algorithm: DQN

❑ Summary of the DQN

- Q-Network Architecture:



- Loss Function

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

- Training Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$ ε-greedy behavior policy

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D experience replay

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

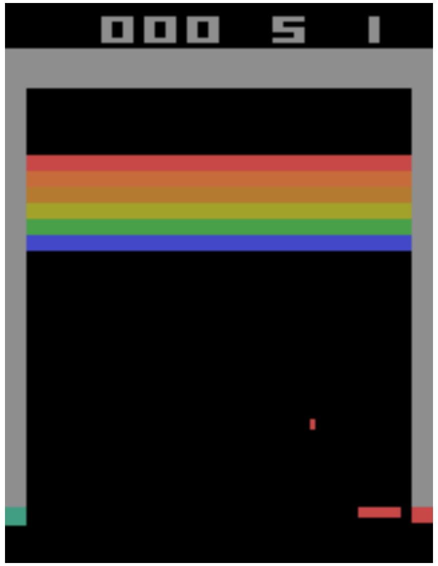
    Every C steps reset $\hat{Q} = Q$ delayed Q-target update

  End For

End For

DRL Algorithm: DQN

❑ Atari Demos[1]



Breakout

Kangaroo

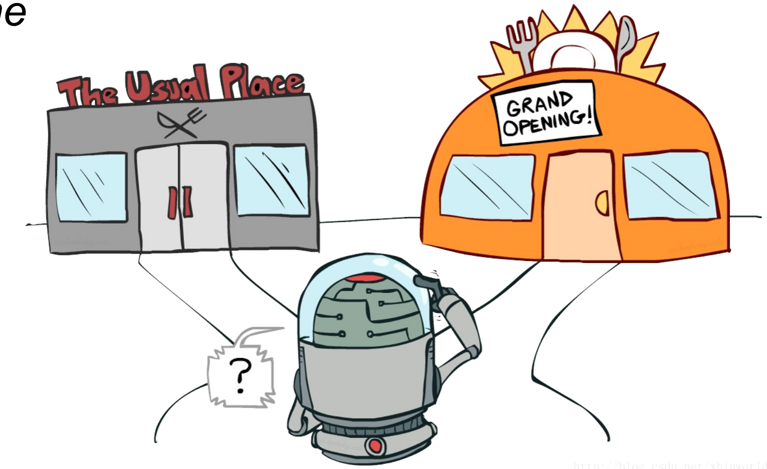
Journey Escape

[1] Demos from: https://github.com/nik-sm/dqn

DRL Algorithm: SAC

❑ Exploration and Exploitation Dilemma

- A simple example: Decide a restaurant to eat
  - ✓ "*Suppose there are 10 restaurants around you, and you have ever tried 8 of them, knowing that the best of the 8 is scored 80, while the rest 2 may be scored 20 or 100.*"
  - ✓ *"Will you choose the best restaurant (80) that you tried?"*
  - ✓ *"Or will you explore a new restaurant from the two?"*



- Fundamental concepts that guide the design of most modern DRL algorithms.
- <u>How to Explore the Action Space?</u>
  - ✓ Q-learning / DQN: ε-greedy method
  - ✓ Maximum Entropy RL: **maximize the entropy** of action distribution, e.g., SAC.

DRL Algorithm: SAC

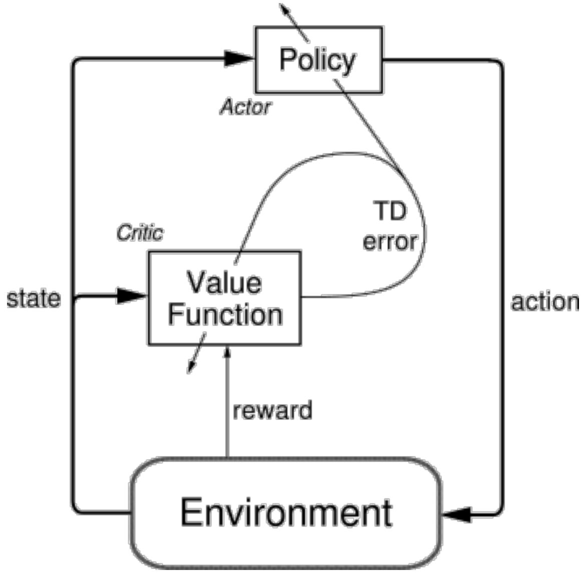
❑ Soft Actor-Critic (SAC)[1]

- Training Goal: maximize the total expected reward and the entropy of actions.

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t, a_t) + \alpha\mathcal{H}(\pi_\theta(.|s_t))]$$

- Different to DQN that only learns the value, SAC learns both value and policy.
- Core Elements:
  - ✓ Soft Policy Network (**Actor**): produce action by giving a state (π_θ)
  - ✓ Soft Value Network (**Critic**): state value (V_ψ), and state-action value (Q_w)

- Central Idea of Actor-Critic:
  - A **policy gradient** method: directly optimize policy by gradient descent.
  - **Actor**: decide which action should be taken.
  - **Critic**: inform the actor how “good” was the action, and how it should adjust.



[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In ICML, 2018.

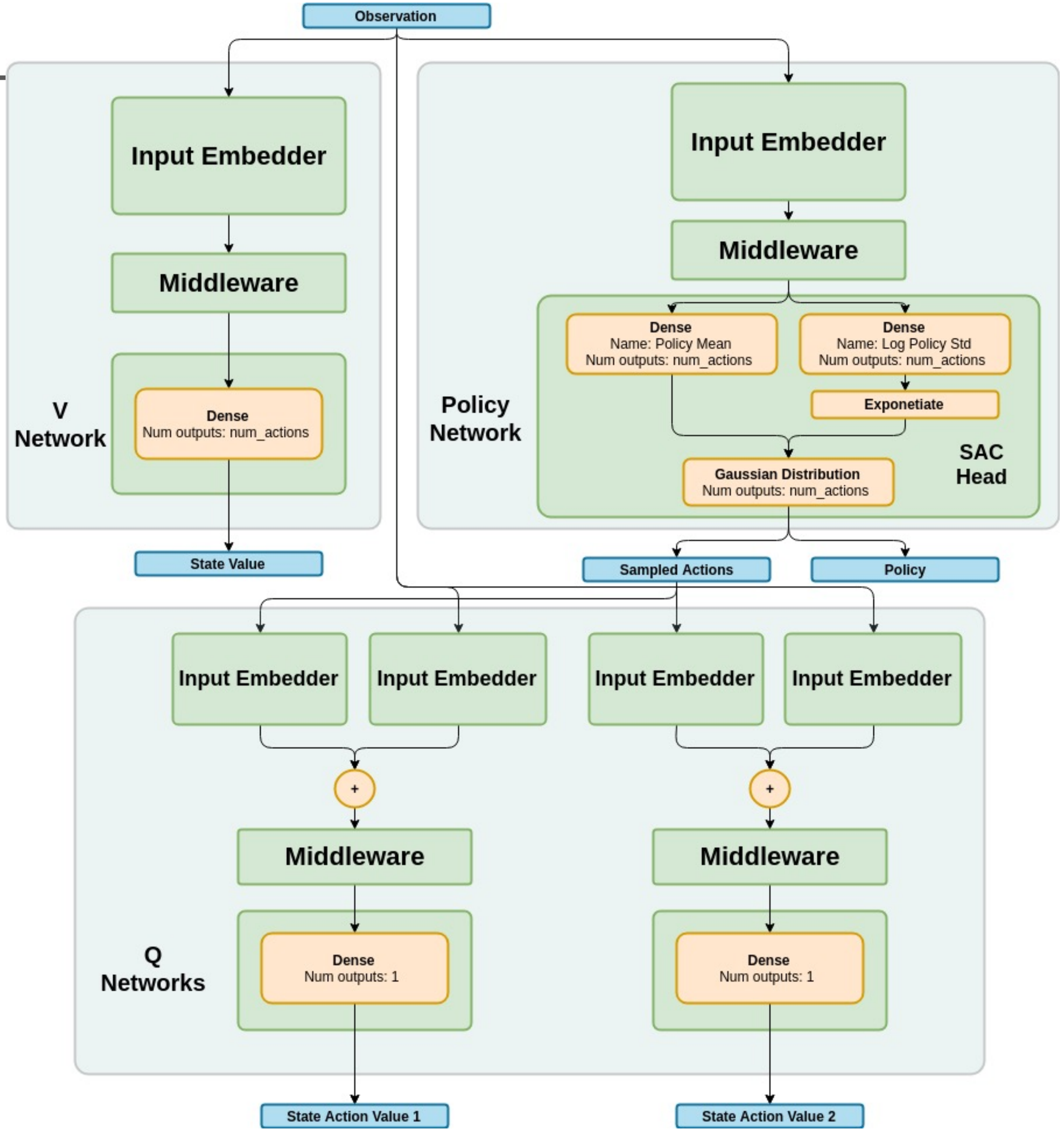
DRL Algorithm: SAC

❑ Network Architectures

- Policy Network: $\boldsymbol{a} = \pi(\boldsymbol{s}; \theta)$
- State Value Network: $v = V(\boldsymbol{s}; \psi)$
- Q Networks: $q = Q(\boldsymbol{s}, \boldsymbol{a}; w)$

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for



DRL Algorithm: SAC

❑ Loss Functions

- Policy Network: $\boldsymbol{a} = \pi(\boldsymbol{s}; \theta)$, updated by minimizing the KL divergence between action distributions and energy distribution of Q values.

$$J_\pi(\theta) = \nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta(.\,|s_t) \| \exp(Q_w(s_t,.) - \log Z_w(s_t))\right)$$

- State Value Network: $v = V(\boldsymbol{s}; \psi)$, updated by using Q value and entropy as the target

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}[Q_w(s_t, a_t) - \log \pi_\theta(a_t|s_t)]\right)^2]$$

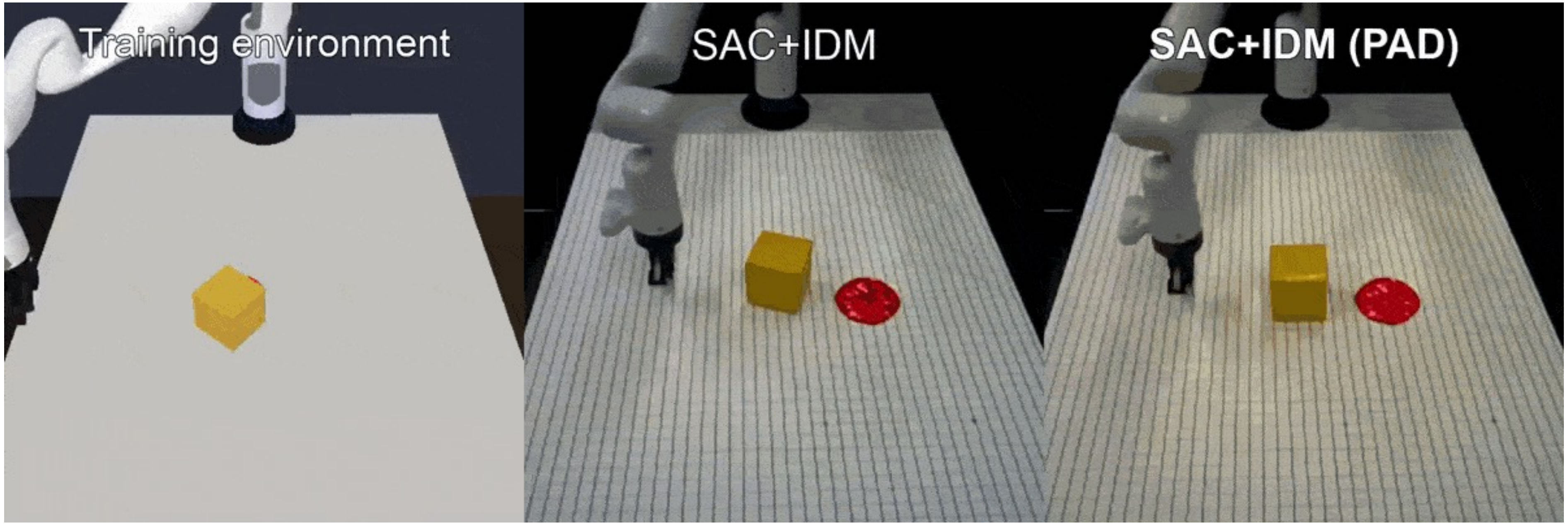
- Q Network: $q = Q(\boldsymbol{s}, \boldsymbol{a}; w)$, updated by minimizing the TD error

$$J_Q(w) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}\left(Q_w(s_t, a_t) - (r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V_{\bar{\psi}}(s_{t+1})])\right)^2]$$

DRL Algorithm: SAC

❑ Some online resources

- Sample code of SAC: https://github.com/pranz24/pytorch-soft-actor-critic/blob/master/sac.py
- SAC is known as SOTA for Robot Learning[1]



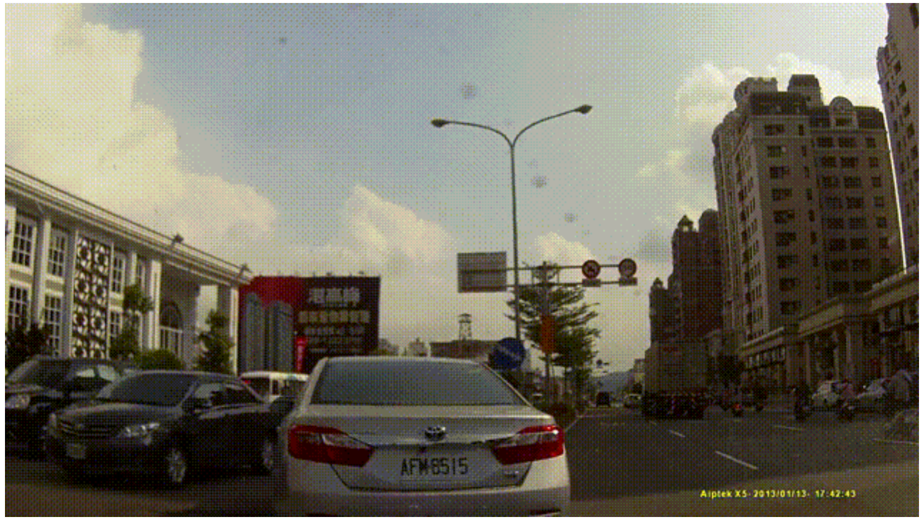
[1] Demos are from BAIR's ICLR 2021 work: https://bair.berkeley.edu/blog/2021/02/25/ss-adaptation

DRL for Traffic Accident Anticipation[1]

Traffic Accident Anticipation

- Anticipate the traffic accidents before they happen **as early as possible (AEAP)**.
- Given a dashcam video, we need to know **if and when** an accident will happen.

Accident Video (positive)



Non-accident Video (negative)

[1] **Wentao Bao**, Qi Yu, and Yu Kong. "Deep Reinforced Accident Anticipation with Visual Explanation." in ICCV, 2021.

DRL for Traffic Accident Anticipation

Challenges

- Visual cues of future accidents are difficult to be captured
  - Explicitly model the **human visual attention**

***Where do drivers look** when predicting future accidents?*

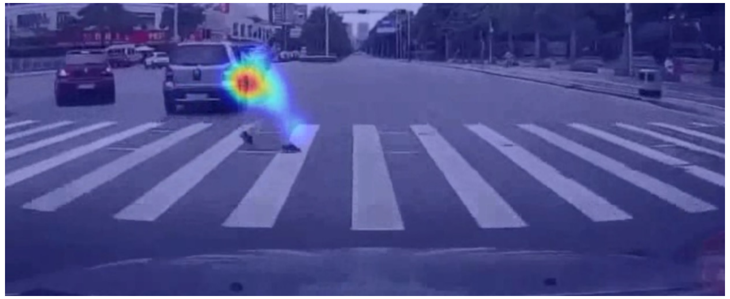
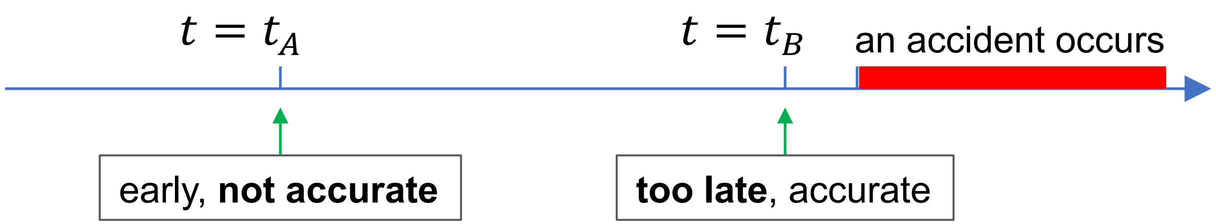
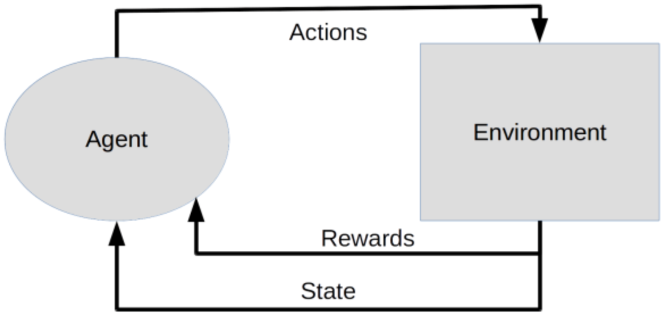


Illustration for drivers' visual attention

- Trade-off between **early** and **accurate** decisions
  - Formulate the task as a Reinforcement Learning problem

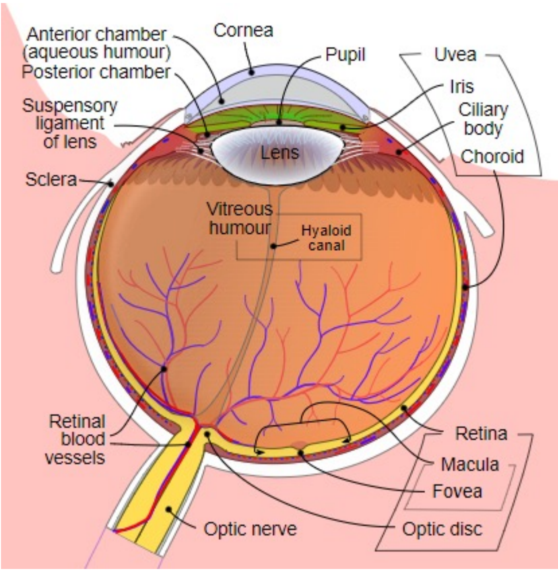




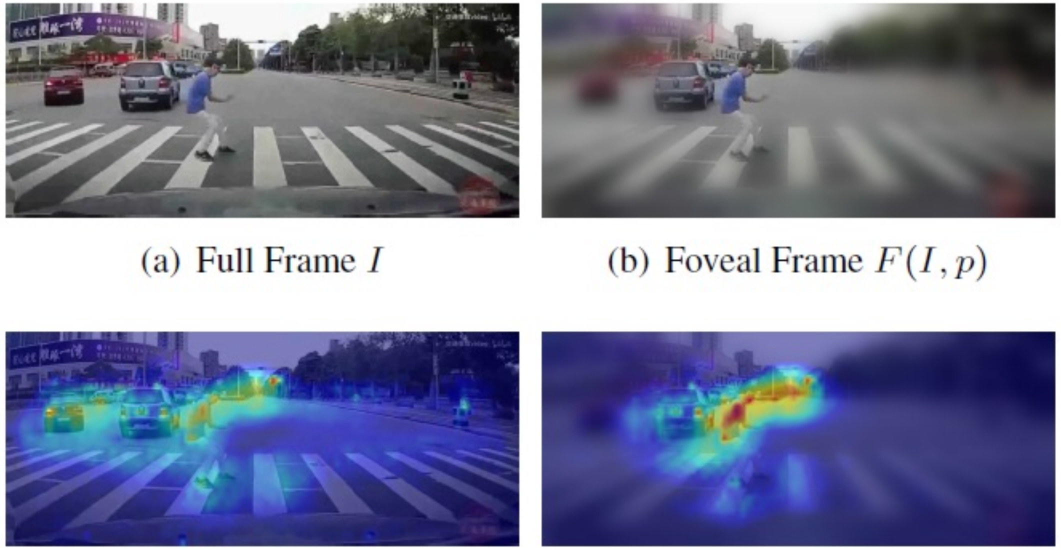
DRL for Traffic Accident Anticipation

Preliminary: Human Visual Attention

- Biological Vision System
  - Foveal vision: recognizing object semantics.
  - Peripheral vision and working memory: drives visual exploration.
- Human visual attention is computationally simulated by saliency map.
- With eye-tracking data, saliency map could be predicted by DNN.



Fovea on Retina (Wikipedia)



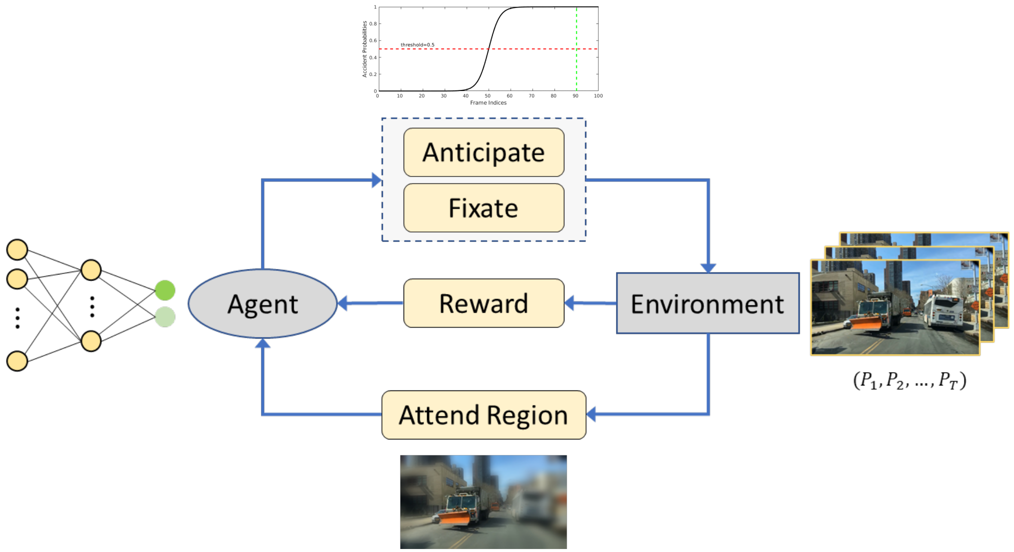
(a) Full Frame I (b) Foveal Frame $F(I,p)$

(c) Bottom-up Attention $G(I)$ (d) Top-down Attention $G(F(I,p))$

DRL for Traffic Accident Anticipation

DRL Setup

- Basic Elements:
  - ✓ **Agent**: deep neural networks (DNN).
  - ✓ **Environment**: dashcam traffic videos.
  - ✓ **Goal**: predict accident AEAP, visually explainable
- Main Elements:
  - ✓ **State**: 1) attended local region 2) historical memory
  - ✓ **Action:** 1) fixation location of the agent 2) probability of a future accident
  - ✓ **Reward**: 1) earliness, 2) correctness, and 3) attentiveness



DRL for Traffic Accident Anticipation

Methodology Overview

- The **traffic observation environment** identifies representative features as observation state.
- The **stochastic multi-task agent** predicts both the accident score and the next fixation point.
- Improves the SAC for training the DRIVE model.

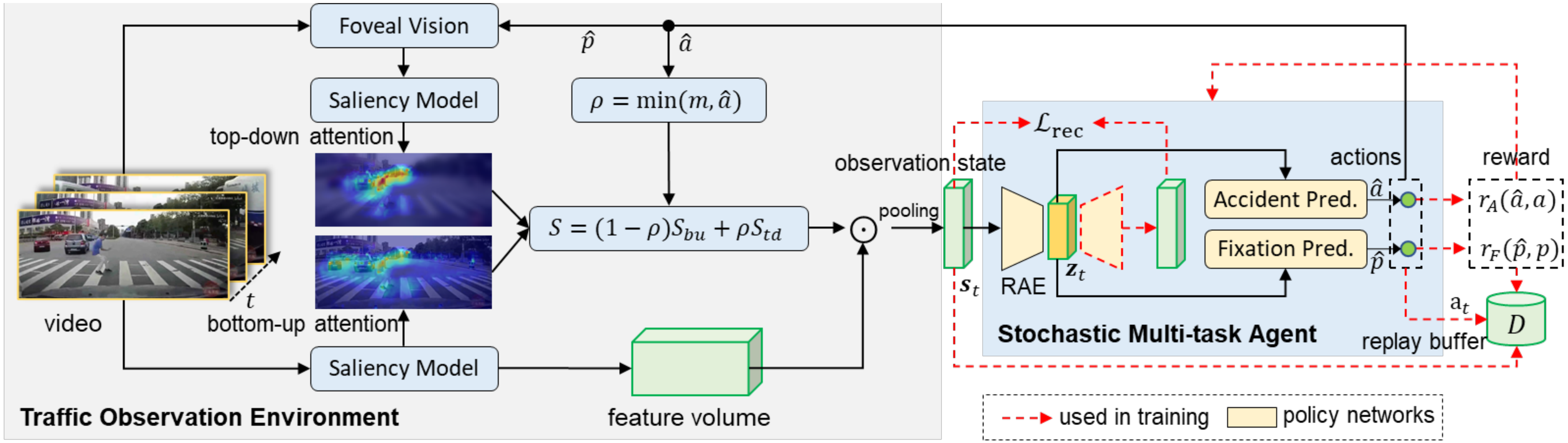
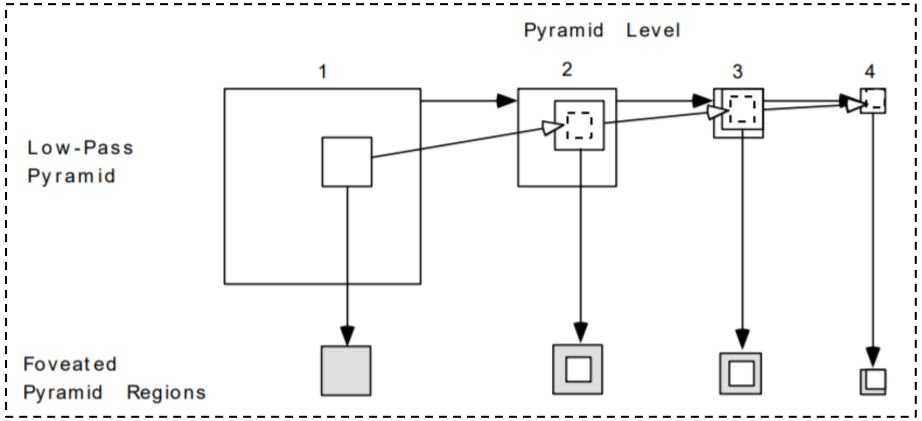


Fig: The proposed DRIVE model

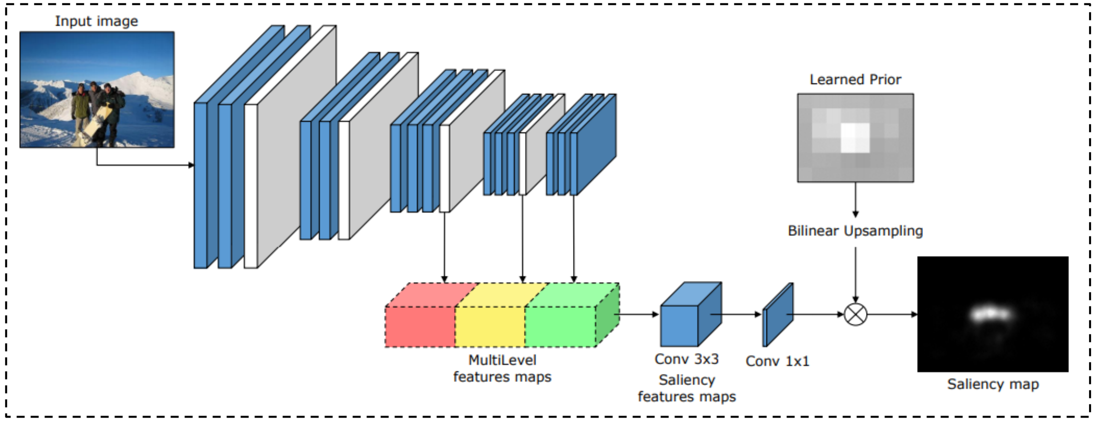
DRL for Traffic Accident Anticipation

Traffic Observation Environment

- Traffic visual attention modeling by CNNs
  - Foveation is implemented by the multi-level low-pass pyramid method[1].
  - MLNet[2] with VGG-16 backbone.
  - MLNet is trained on DADA-2000 dataset.



Foveation by multi-level low-pass pyramid[1]



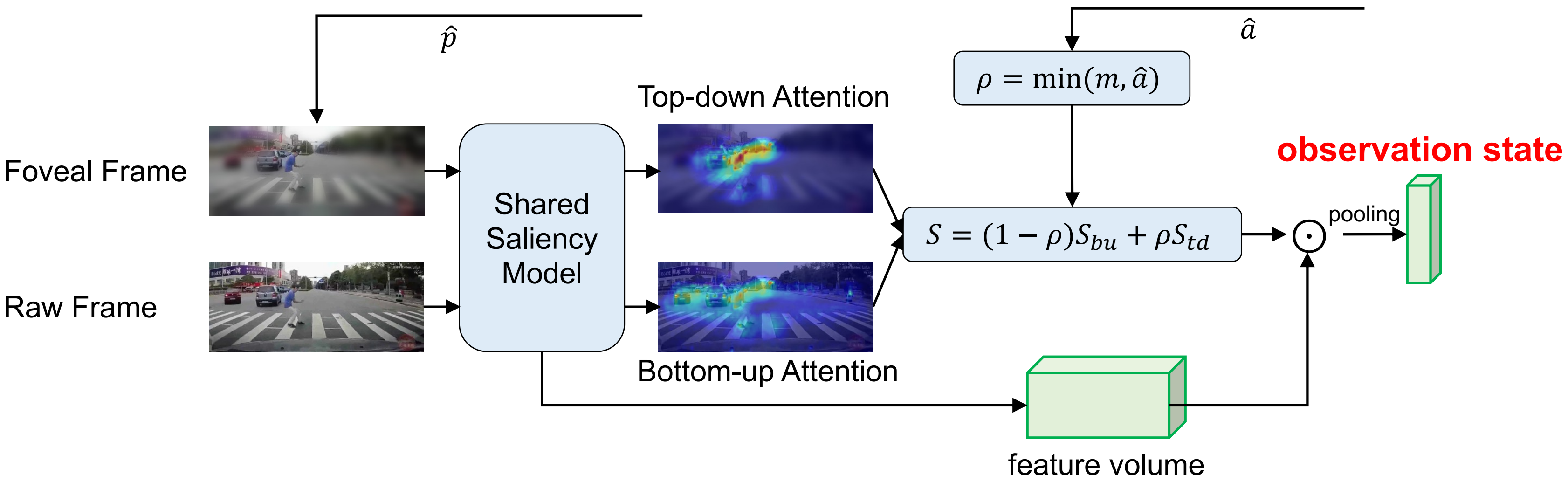
Saliency prediction by MLNet[1]

[1] Wilson S Geisler and Jeffrey S Perry. Real-time foveated multiresolution system for low-bandwidth video communication. In Human Vision and Electronic Imaging III, volume 3299, pages 294–305, 1998.
[2] Cornia, Marcella, et al. "A deep multi-level network for saliency prediction." in *ICPR*, 2016.

DRL for Traffic Accident Anticipation

Traffic Observation Environment

- State representation



  - Dynamic Attention Fusion (DAF)

$$S^t = (1-\rho^t)S^t_{bu} + \rho^t S^t_{td},$$

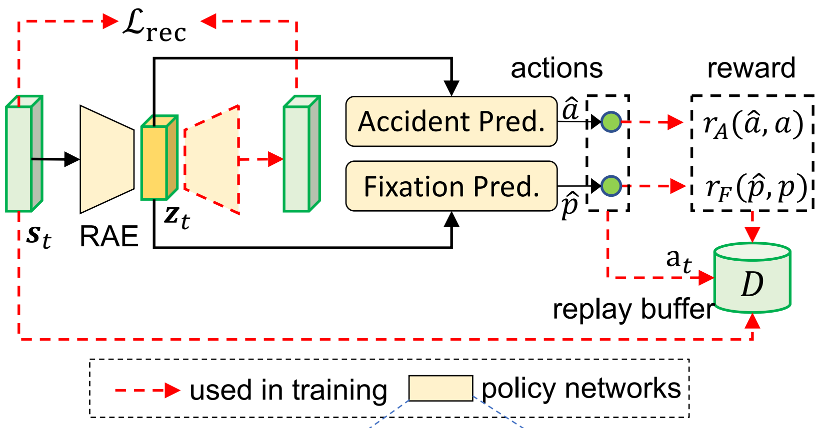
  - Feature pooling and concatenation

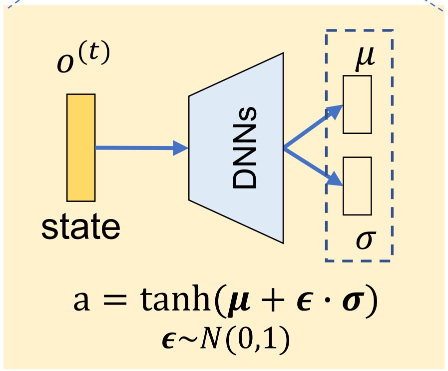
$$\mathbf{s}^i_t = \text{cat}\left(\tilde{f}_{GMP}(S^t \odot V^t_i), \tilde{f}_{GAP}(S^t \odot V^t_i)\right),$$

DRL for Traffic Accident Anticipation

Stochastic Multi-task Agent

- Multi-task Agent Architecture
  - Regularized AutoEncoder (RAE)
  - Two stochastic policy networks





- Action representation

$$\hat{\mathbf{a}}_t = \operatorname{cat}\left(\phi_A\left(\mathcal{E}(\mathbf{s}_t)\right), \phi_F\left(\mathcal{E}(\mathbf{s}_t)\right)\right).$$

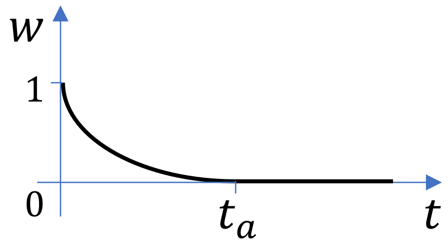
DRL for Traffic Accident Anticipation

Reward Functions and Training

- Dense Anticipation Reward

$$r_A^t = w_t \cdot \text{XNOR}\left[\mathbb{I}[a^t > a_0], y\right],$$

$$w_t = \frac{1}{e^{t_a} - 1}\left(e^{\max(0, t_a - t)} - 1\right),$$



- Sparse Fixation Reward.

$$r_F^t = \mathbb{I}\left[t > t_a\right] \exp\left(-\frac{||\hat{p}^t - p^t||^2}{\eta}\right),$$

- The model is trained by our improved SAC algorithm.

$$-\mathcal{H}(\pi_\phi(\hat{\mathrm{a}}|\mathrm{s})) = \log\left[\pi_{\phi_A}(\hat{a}|\mathrm{s}) \cdot \pi_{\phi_F}(\hat{p}|\mathrm{s})\right].$$

DRL for Traffic Accident Anticipation

Improved SAC Algorithm

- Optimize the **Critic** by $J(\theta)$

$$J(\theta_i) = \mathbb{E}\left[\left(Q_{\theta_i}(\mathbf{s},\mathbf{a}) - y(r,\mathbf{s}',\mathbf{a})\right)^2\right], \qquad (15)$$

$$y(r,\mathbf{s}',\mathbf{a}) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\bar{\theta}_j}(\mathbf{s}',\hat{\mathbf{a}}') - \alpha \log \pi_\theta(\hat{\mathbf{a}}'|\mathbf{s}')\right)$$

- Optimize the **Actor** by $J(\emptyset)$

$$J_o(\phi) = \mathbb{E}\left[\alpha \log \pi_\phi(\hat{\mathbf{a}}|\mathbf{s}) - \min_{j=1,2} Q_{\theta_j}(\mathbf{s},\hat{\mathbf{a}})\right] + w_0||\phi||^2,$$

$$\begin{aligned} J(\phi_A) &= J_o(\phi) + w_1\mathbb{E}\left[\mathcal{L}(\hat{a}^t, t_a, y)\right] \\ J(\phi_F) &= J_o(\phi) + w_2\mathbb{E}\left[\mathbb{I}[t > t_a] d(\hat{p}^t, p^t)\right], \end{aligned} \qquad (11)$$

- Optimize the **Temperature** by $J(\alpha)$

$$J(\alpha) = \mathbb{E}\left[-\alpha \log \pi_\phi(\hat{\mathbf{a}}|\mathbf{s}) - \alpha\mathcal{H}_0\right], \qquad (12)$$

$$\alpha \leftarrow \max(\alpha - \lambda_\alpha \hat{\nabla}_\alpha J(\alpha), \alpha_0)$$

- Optimize the **RAE** by $J(\beta)$

$$J_{RAE}(\beta) = \mathcal{L}_{rec}(\mathbf{s};\beta) + w_0||\beta||^2 + w_{\mathbf{s}}||\mathbf{z}||^2, \qquad (14)$$

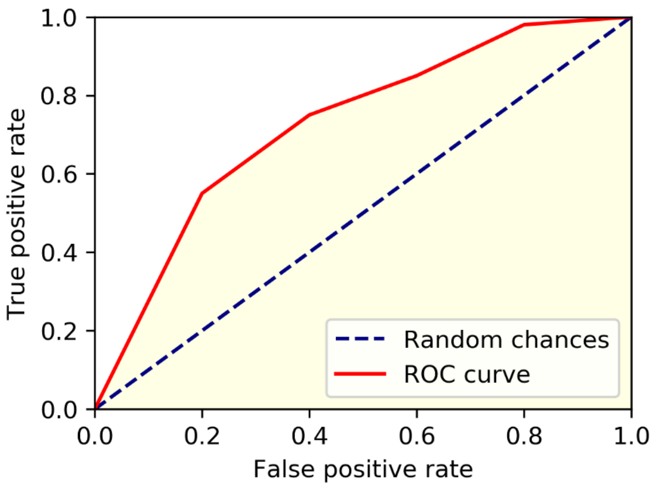
Algorithm 1 Improved SAC for the DRIVE Model Training

Require: $\theta_1, \theta_2, \phi, \beta$ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target networks
$\mathcal{D} \leftarrow \emptyset$, $\mathbf{h}_0 \leftarrow \mathbf{0}$ ▷ Replay buffer and hidden states
for each iteration **do**
  for each environment step **do**
    Sample actions $(\mathbf{a}_t, \mathbf{h}_t) \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_{t-1})$
    Compute state $\mathbf{s}_t$ with actions ▷ See Eq. 2
    Compute reward $r_t = r_A^t + r_F^t$ ▷ See Eq. 4-6
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{h}_t, \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    for each critic update **do**
      $\theta \leftarrow \theta - \lambda\hat{\nabla}_\theta J_Q(\theta)$ ▷ Update by Eq. 15
    end for
    $\phi \leftarrow \phi - \lambda\hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update by Eq. 11
    $\alpha \leftarrow \max(\alpha - \lambda_\alpha\hat{\nabla}_\alpha J(\alpha), \alpha_0)$ ▷ See Eq. 12
    $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ ▷ Update Q-target
    $\beta \leftarrow \beta - \lambda\hat{\nabla}_\beta J_{\mathrm{RAE}}(\beta)$ ▷ Update by Eq. 14
  end for
end for
Ensure: $\theta_1, \theta_2, \phi, \beta$

DRL for Traffic Accident Anticipation

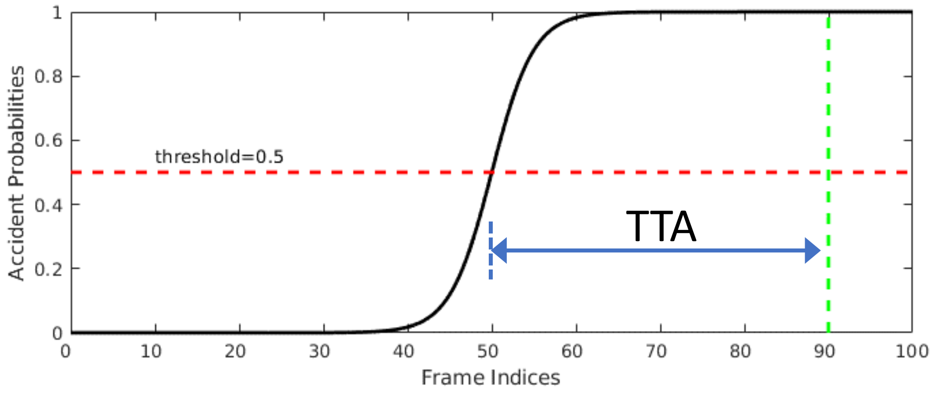
Evaluation Metrics

- Area under ROC (AUC)



	Real Pos.	Real Neg.
Pred. Pos.	TP	FP
Pred. Neg.	FN	TN

- Time-to-Accident (TTA)



DRL for Traffic Accident Anticipation

☐ Datasets

- DADA-2000[1]
  - Provides ~2,000 dashcam videos containing traffic accidents.
  - Drivers’ eye fixation points are captured in lab by eye-tracking device.
  - Spatial resolution: 660 x 1584
  - 30 frames per second
  - Videos are untrimmed, from which the negative video clips are sampled.

- DAD[2]
  - Provides 620 positive (accident) and 1130 negative (normal) dashcam videos
  - Videos are trimmed to 5 seconds long.
  - Spatial resolution: 720 x 1080

[1] J. Fang, D. Yan, J. Qiao, J. Xue, H. Wang and S. Li, "DADA-2000: Can Driving Accident be Predicted by Driver Attention? Analyzed by A Benchmark," 2019 IEEE Intelligent Transportation Systems Conference (ITSC), 2019, pp. 4303-4309.
[2] Fu-Hsiang Chan, Yu-Ting Chen, Yu Xiang, and Min Sun. Anticipating accidents in dashcam videos. In ACCV, 2016.

DRL for Traffic Accident Anticipation

Comparison with SOTA baselines

- Our model achieves the **best AUC** score and competitive TTA performance.
- Our method is flexible to be extended **without fixation annotations**.
- Ablation studies show the contributions from bottom-up (BU) and top-down (TD) attentions, as well as the RAE.

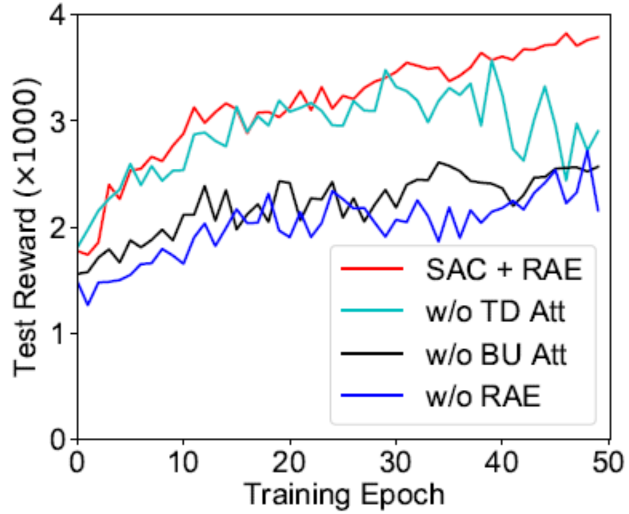
Methods	DADA-2000 [11]		DAD [3]	
	AUC (%)	TTA (s)	AUC (%)	TTA (s)
DSA-RNN [3]	47.19	3.095	71.57	1.169
AdaLEA [40]	55.05	**3.890**	58.06	2.228
UString [2]	60.19	3.849	65.96	0.915
DRIVE (ours)	**72.27**	3.657	**93.82**	**2.781**

DRL for Traffic Accident Anticipation

Ablation Study

- We analyzed the contributions of each major component, including:
  - Training algorithm (SL vs. RL).
  - Vanilla SAC-based RL algorithm vs. SAC+RAE method.
  - With vs. without human fixations as ground truth in training.
- AUC results and reward curves of training process

Type	SAC	RAE	Fixations	AUC (%)
RL	✓	✓	✗	61.91
RL	✓	✗	✓	66.21
SL	✗	✓	✓	63.96
RL	✓	✓	✓	**72.27**



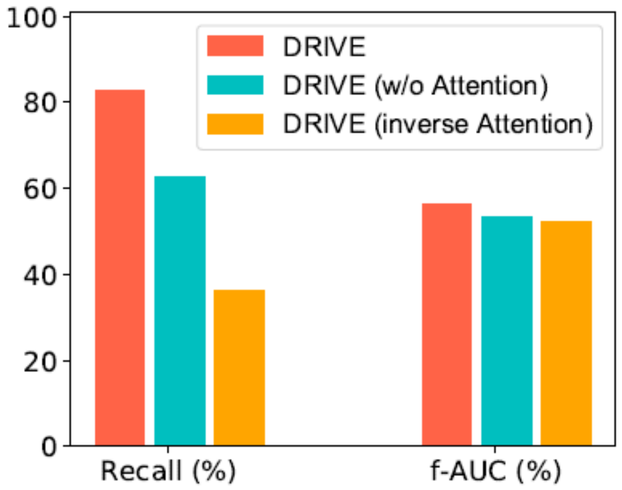
DRL for Traffic Accident Anticipation

Visual Explanation Results

- Correlation between Visual Attention and Accident Anticipation

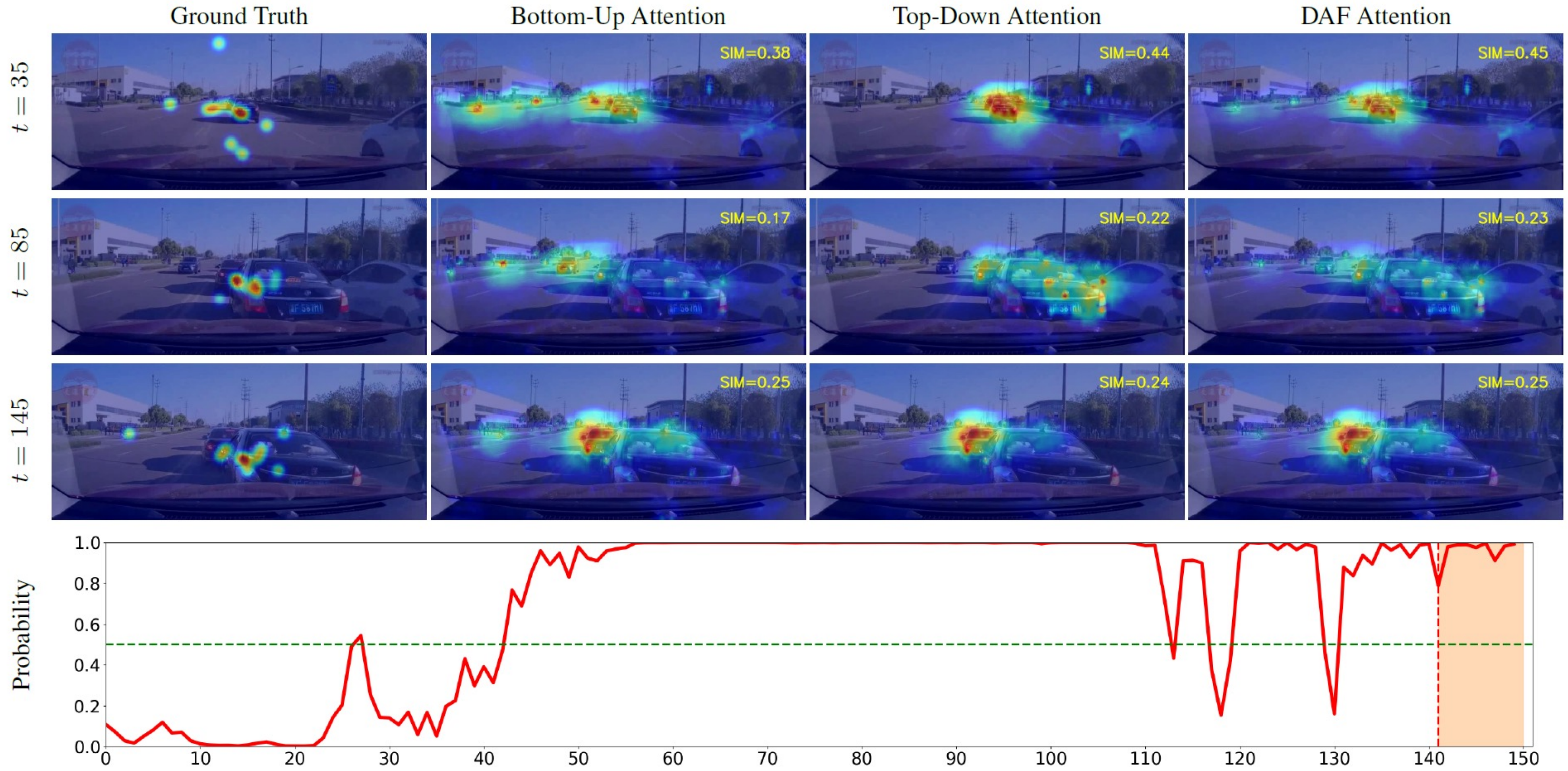
Params	Methods	AUC	SIM	CC	KLD (↓)
0.5	SAF	0.645	0.188	0.322	2.679
	DAF	**0.659**	**0.192**	**0.331**	**2.654**
0.8	SAF	0.691	0.144	0.190	3.087
	DAF	**0.726**	**0.158**	**0.226**	**2.986**
1.0	SAF	0.632	0.080	0.079	12.948
	DAF	**0.679**	**0.112**	**0.143**	**7.836**

- Explainable Results by Attention Intervention



DRL for Traffic Accident Anticipation

Visualization



DRL for Traffic Accident Anticipation

■ Our Online Resources

Paper & Supp.: https://arxiv.org/abs/2107.10189

Code: https://github.com/Cogito2012/DRIVE

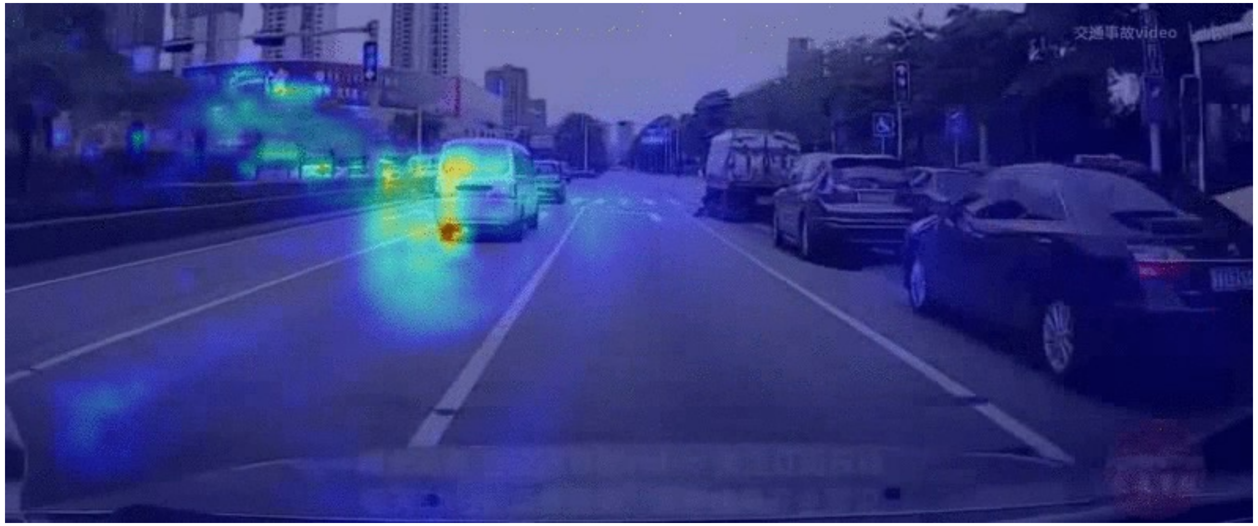
Project: https://www.rit.edu/actionlab/drive

Project

Code

■ Demo

YouTube Demo: https://www.youtube.com/watch?v=A3bTWejzUwM



Feel free to contact me via wb6219@rit.edu

Q & A