Internal Assessment Test 2 – November 2025

Sub:	Deep Learning and Reinforcement Learning					Sub Code:	BAI701	Branch:	AIML
Date:		Duration:	90 min	Max Marks:	50	Sem/Sec:	VII (A& B)		OBE

	Scheme of Evaluation	**MARKS**	**CO**	**RBT**
1	Explain the working mechanism of a Recurrent Neural Network (RNN) and explain why it is effective for handling sequential or time-dependent data.	**10**	**CO2**	**L2**

Solution: Explanation 10M

A **Recurrent Neural Network (RNN)** is a type of neural network specially designed for sequential data. Unlike feed-forward networks, an RNN has **loops in its architecture**, which allow information from previous time steps to influence the current output. As stated in the PDF, *"RNN has recurrence in it and the current output is influenced not only from the current input but also from the past inputs."*

How an RNN Works

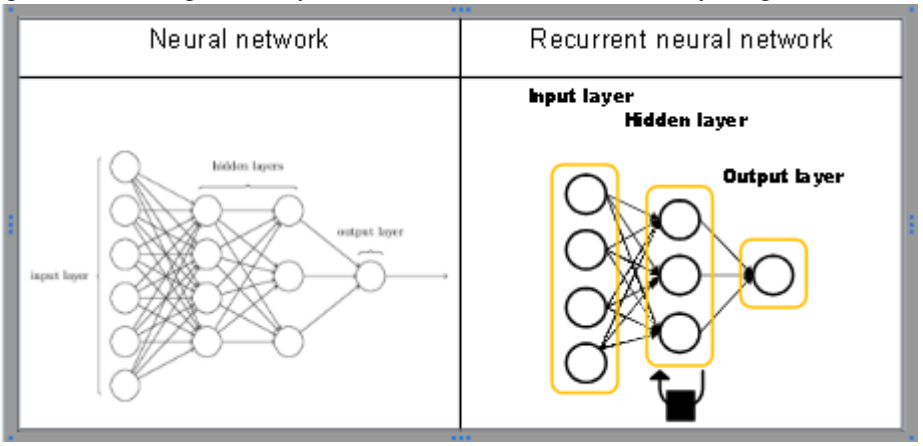
An RNN processes inputs **one time-step at a time**:

1. At each time-step *t*, the RNN receives:
   - Current input xtx_txt
   - Hidden state from previous step ht−1h_{t-1}ht−1
2. It combines them to produce:
   - New hidden state hth_tht
   - Output ytyty_tyt

The hidden state acts as the **memory**, storing relevant information from earlier steps. This allows the "unfolded computational graph" of the RNN to propagate information across time. The PDF states that the *"unfolded computational graph in an RNN allows considering the historical context or information"*

1. RNN - Specialized for processing a sequence data

eg. This movie is good, really not bad vs. This movie is bad, really not good.



2. RNN - Sharing parameters across different parts of a model

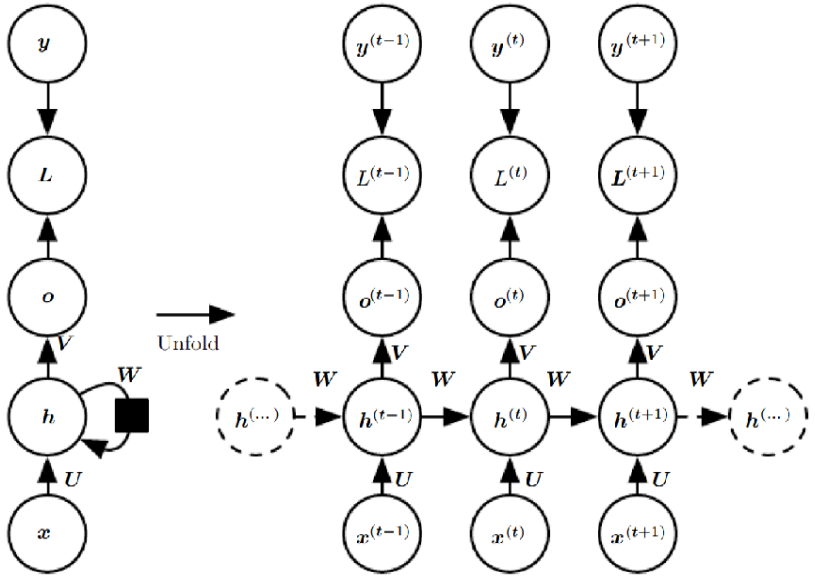
- Such sharing is particularly important when a specific piece of information can occur at multiple positions within the sequence.
- For example, consider the two sentences "I went to Nepal in 2009" and "In 2009, I went to Nepal."
- If we ask a machine learning model to read each sentence and extract the year in which the narrator went to Nepal, we would like it to recognize the year 2009 as the relevant piece of information, whether it appears in the sixth word or the second word of the sentence.

• Feed Forward

- Separate Parameter for each Input
- Needs to Learn ALL of the Rules of Languages

• CNN

- Output of CNN is a Small Number of Neighboring Member of Inputs.
- The idea of parameter sharing manifests in the application of the same convolution

	kernel at each time step. • **Recurrent networks share parameters in a different way.** • **Each member of the output is a function of the previous members of the output.** • **Each member of the output is produced using the same update rule applied to the previous outputs.** • **This recurrent formulation results in the sharing of parameters through a very deep computational graph.**			
2	Analyze the role of Gradient Descent in training RNNs and explain how vanishing and exploding gradients influence learning performance and stability. Solution: 10 The recurrent neural network of figure 10.3 and equation 10.8 is universal in the sense that any function computable by a Turing machine can be computed by such a recurrent network of a finite size. We now develop the forward propagation equations for the RNN depicted in figure 10.3. The figure does not specify the choice of activation function for the hidden units. Here we assume the hyperbolic tangent activation function. Also, the figure does not specify exactly what form the output and loss function take.  Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)*The RNN and its loss drawn with recurrent connections. *(Right)*The same seen as an time-unfolded computational graph, where each node is now associated with one particular time instance. We can then apply the softmax operation as a post-processing step to obtain a vector $\hat{y}$ of normalized probabilities over the output. Forward propagation begins with a specification of the initial state $h^{(0)}$. Then, for each time step from $t = 1$ to $t = \tau$ we apply the following update equations: $a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$ (10.8) $h^{(t)} = \tanh(a^{(t)})$ (10.9) $o^{(t)} = c + Vh^{(t)}$ (10.10) $\hat{y}^{(t)} = \text{softmax}(o^{(t)})$ (10.11) where the parameters are the bias vectors b and c along with the weight matrices U, V and W, respectively for input-to-hidden, hidden-to-output and hidden-to-hidden connections. This is an example of a recurrent network that maps an input sequence to an output sequence of the same length. **The total loss for a given sequence of *x* values paired with a sequence of y values would then be just the sum of the losses over all the time steps.**	10	CO2	L3

For example, if $L^{(t)}$ is the negative log-likelihood of $y^{(t)}$ given $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}$, then

$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \tag{10.12}$$

$$= \sum_t L^{(t)} \tag{10.13}$$

$$= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right), \tag{10.14}$$

where $p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{\boldsymbol{y}}^{(t)}$.

Computing the gradient of this loss function with respect to the parameters is an expensive operation.

The gradient computation involves performing a forward propagation pass moving left to right through our illustration of the unrolled graph in figure 10.3, followed by a backward propagation pass moving right to left through the graph.

The runtime is $O(\tau)$ and cannot be reduced by parallelization because the forward propagation graph is inherently sequential; each time step may only be computed after the previous one.

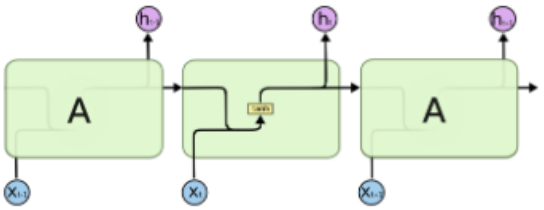
States computed in the forward pass must be stored until they are reused during the backward pass, so the memory cost is also $O(\tau)$.

The back-propagation algorithm applied to the unrolled graph with $O(\tau)$ cost is called **back-propagation through time** or BPTT.

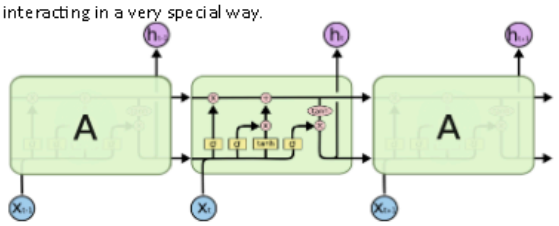
The network with recurrence between hidden units is thus very powerful but also expensive to train.

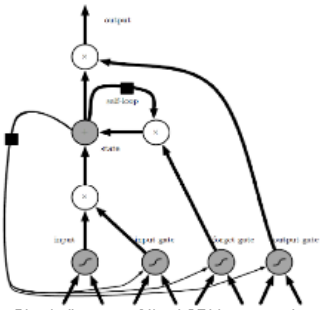
3	Describe how an LSTM network improves learning in long-term sequential data compared to a conventional RNN. Solution: 10M	10	CO2	L2

- Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable o learning long-term dependencies.
- LSTMs are explicitly designed to avoid the long-term dependency problem.
- Remembering information for long periods of time is practically their default behavior.
- All recurrent neural networks have the form of a chain of repeating modules of neural network.
- In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



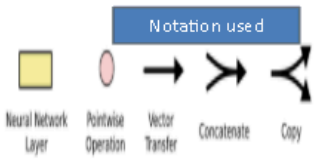
- LSTMs also have this chain like structure, but the repeating module has a different structure.
- Instead of having a single neural network layer, there are four, interacting in a very special way.



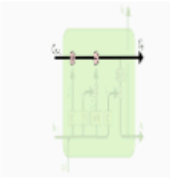


Block diagram of the LSTM recurrent network "cell."

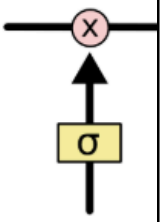
In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.



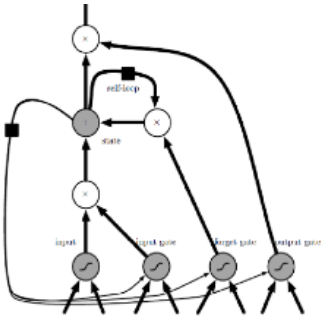
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt.
- It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.



- Gates are a way to optionally let information through. Consider this example - They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.
- A value of zero means "let nothing through," while a value of one means "let everything through!"
- An LSTM has three of these gates, to protect and control the cell state.



- The LSTM has been found extremely successful in many applications, such as handwriting recognition, speech recognition, handwriting generation, machine translation, image captioning and Parsing.
- LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN.
- Each cell has the same inputs and outputs as an ordinary recurrent network, but has more parameters and a system of gating units that controls the flow of information.
- The most important component is the state unit $s_i^{[t]}$ that has a self-loop. However, here, the self-loop weight is controlled by a **forget gate** unit $f_i^{[t]}$ (for time step t and cell i), that sets this weight to a value between 0 and 1 via a sigmoid unit.



Block diagram of the LSTM recurrent network "cell."

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state.

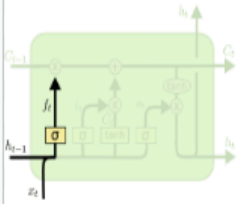
This decision is made by a sigmoid layer called the "forget gate layer."

It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

Let's go back to our example of a language model trying to predict the next word based on all the previous ones.

In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

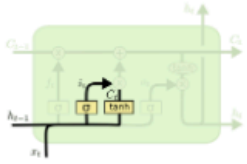
$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

where $\boldsymbol{x}^{(t)}$ is the current input vector and $\boldsymbol{h}^{(t)}$ is the current hidden layer vector containing the outputs of all the LSTM cells, and $\boldsymbol{b}^f$, $\boldsymbol{U}^f$, $\boldsymbol{W}^f$ are respectively biases, input weights and recurrent weights for the forget gates.

Step-by-Step LSTM Walk Through

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

4	Explain the concept of stateless algorithms in the context of reinforcement learning and describe how they function in decision-making systems. Also, discuss their suitability along with their basic advantages and limitations. Solution: 3 methods - 10 M **Stateless Algorithms in Reinforcement Learning — Explanation Based on the Screenshots** **Concept of Stateless Algorithms** From the screenshots, the discussion focuses on the **multi-armed bandit problem**, which is the *simplest case of reinforcement learning*. In this setting: • Every trial provides **the same probabilistic reward distribution for a given action**. • There is **no notion of state**, unlike games such as chess where the next decision depends on the board state. Thus, **stateless algorithms** are decision-making methods where: • The agent **does not track or update a system state**. • Each action's reward is assumed to depend only on the action itself, not on any external or changing state. The agent simply learns the **expected payoff of each action** (slot machine) and bases its decision on that.	10	CO3	L2

How Stateless Algorithms Function

The screenshots describe three stateless strategies:

1. Naïve Algorithm

- Play each machine a fixed number of times (exploration).
- Then permanently select the machine with the highest observed payoff (exploitation).
- Very simple but rigid.

2. ε-Greedy Algorithm

- With probability **ε**, choose a random machine (exploration).
- With probability **1 − ε**, choose the machine with the best current average payoff (exploitation).
- Ensures one is **not trapped forever** in a wrong strategy.

3. Upper-Bounding (UCB-like) Methods

- For each machine, compute an **upper confidence bound**:
  Ui=Qi+CiU_i = Q_i + C_iUi=Qi+Ci
  where
  - QiQ_iQi = estimated mean reward
  - CiC_iCi = confidence interval (bonus for uncertainty)
- Machines with fewer trials get a **larger bonus**, encouraging exploration.
- Merges exploration and exploitation automatically.

All these methods use **only past rewards of actions**, not any environmental state.
Hence they are stateless.

Suitability of Stateless Algorithms

Stateless algorithms are suitable when:

- The environment **does not change with actions**.
- Rewards depend only on **which action is chosen**, not on any world state.
- The problem resembles a **multi-armed bandit** rather than a full RL environment.

Examples include:

- Choosing advertisements
- Selecting network routing paths
- Online recommendation systems

Advantages (Based on the Screenshots)

Naïve Method

- Simple to implement.

	• Conceptually easy. **ε-Greedy** • Prevents being "trapped" in a wrong strategy. • Uses the best strategy for most trials. • Balances exploration and exploitation. **Upper-Bounding Methods** • More efficient learning of payoffs. • Encourages exploration of uncertain actions. • Allows explicit control of confidence through constant **K**. • Does not require a separate exploration phase; both aspects are integrated. --- **Limitations (From the Screenshots)** **Naïve Method** • Hard to decide how many trials are enough for accurate payoff estimation. • May require **many exploratory trials**, wasting effort. • If the final choice is wrong, the gambler uses the wrong machine forever. • Unrealistic for real problems due to its fixed strategy. **ε-Greedy** • Choosing ε is difficult and context-dependent. • Small ε may take long to find the correct best option. • Large ε wastes trials on random exploration. **Upper-Bounding (UCB)** • Requires maintaining statistical confidence intervals. • Larger K values cause excessive exploration. • Still stateless, so unsuitable for problems requiring state-dependent decisions.			
5 a	Explain the basic framework of reinforcement learning with a neat diagram. Solution: Diagram- 2m Explanation- 3M	5	CO3	L2

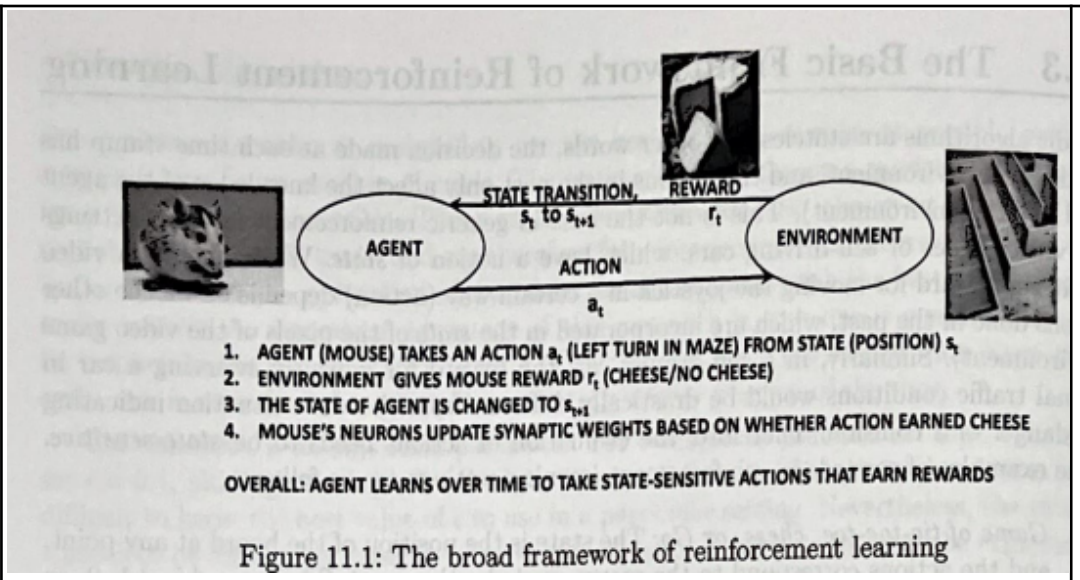


Figure 11.1: The broad framework of reinforcement learning

Framework of Reinforcement Learning (Based on the Figure)

The figure presents the **broad framework of reinforcement learning**, which describes how an *agent* interacts with an *environment* to learn actions that maximize rewards over time.

1. Agent Takes an Action (a□)

- The **agent** (mouse) observes its current state sts_tst (its position in the maze).
- It chooses and performs an **action**—for example, *turn left*.

2. Environment Returns a Reward (r□)

- After the action, the **environment** provides feedback.
- The reward tells whether the action was good or bad—for example, *cheese = reward*, *no cheese = no reward*.

3. State Transition (s□ → s□+1)

- The environment then moves the agent to a **new state** st+1s_{t+1}st+1.
- Example: After turning left, the mouse enters a new part of the maze.

4. Agent Updates Its Knowledge

- Based on the reward, the agent **updates its internal knowledge** (synaptic weights).
- Good actions get strengthened; bad actions get weakened.

5 b	Discuss the reasons for using Reinforcement Learning in dynamic and real-time environments. Solution: any 3 reason **1. RL Learns Directly From Interaction With the Environment** As shown in the framework: ● The agent takes an action ● The environment returns a reward ● The agent updates its strategy This makes RL ideal for environments that **cannot be fully modeled beforehand** and must be learned through trial-and-error. --- **2. Ability to Adapt to Uncertain and Changing Conditions** The text about bandit problems explains that **payoffs are unknown** and must be learned by exploring machines. This reflects real-time environments where: ● system behavior changes, ● rewards vary over time, ● past observations may not be sufficient. RL continues to adjust actions based on the latest rewards, making it effective in **non-static** environments. --- **3. Handles the Exploration–Exploitation Trade-off** The screenshots emphasize this trade-off: ● Too much exploration wastes resources ● Too much exploitation risks getting stuck with the wrong choice Algorithms like ε-greedy and Upper Confidence Bound (UCB) are designed to continuously **balance discovering new strategies (exploration)** with **using the best-known strategy (exploitation)**. Dynamic environments require this constant balance because conditions may shift. --- **4. Works Even Without a Full State Model** The multi-armed bandit is introduced as the **simplest case of reinforcement learning**, where there is *no notion of state*. This shows that RL can operate even when: ● full system models are unavailable, ● transitions are unknown, ● only rewards from actions are visible. This is realistic for many **real-time problems** such as online recommendation, network routing, robotics, etc. ---	5	CO3	L2

	5. Enables Continuous Improvement Over Time The framework diagram shows how: ● rewards influence the agent's future decisions ● the agent gradually selects actions that lead to higher long-term reward In dynamic systems, RL provides **ongoing learning**, not one-time optimization. It adapts behavior as new data arrives. --- **6. Suitable for Delayed and Sequential Rewards** The framework illustrates that the reward for an action may not be immediate. RL is built for situations where: ● actions influence future outcomes, ● rewards come after several steps, ● the agent must learn the long-term consequences of actions. Dynamic environments frequently have such dependencies.			
6	Apply the principles of reinforcement learning to demonstrate how a self-driving car can be modeled as an intelligent agent. **Solution: 10M** **Modeling a Self-Driving Car as an RL Intelligent Agent** A self-driving car fits perfectly into the RL agent-environment interaction loop. **Agent** The intelligent control system inside the car: ● perception module ● decision-making module ● motion planner **Environment** Everything outside the car: ● road structure ● traffic signals ● pedestrians ● other vehicles ● weather conditions The environment changes continuously—ideal for RL. **State (sts_tst)** Information available to the car at any moment: ● lane position	10	CO3	L3

- speed
- distance to nearby vehicles
- camera/LiDAR perceptions
- traffic light status

Actions (ata_tat)
The car's possible decisions:

- accelerate
- brake
- steer left/right
- change lane
- overtake
- stop

Reward (rtr_trt)
Rewards guide learning:

- positive: staying in lane, smooth driving, obeying signals, reaching destination
- negative: collisions, sharp braking, lane departure, traffic violations

Learning Process

1. The car observes its current state.
2. It selects an action (e.g., brake, steer left).
3. The environment reacts (slows down, avoids collision).
4. Reward is given.
5. The system updates its policy to improve future actions.

Faculty Signature CCI Signature HOD Signature