Self-Thinking Machines in Autonomous Driving

Poondru Prithvinath Reddy

August 1, 2019

Self-Thinking Machines in Autonomous Driving

Poondru Prithvinath Reddy

ABSTRACT

A truly autonomous machine , therefore, needs to be able to learn and adapt its own models. A machine learning about itself and its environment is in the position of being an active part of the system it is trying to learn about; this situation draws interesting parallels with learning in human infants. A system is presented here that enables a machine to autonomously learn a model with no prior knowledge of its own system or the external environment. The autonomous machine sends out random motor commands to its motor system and receives information back from the vision system. This set of evidence is used to learn the structure and parameters of the machine in an environment which are then used as input to 3D Convolutional Neural Network, a machine learning technique, to create a its own internal model. This model can then be used to enable the autonomous machine to predict movements.

INTRODUCTION

Self-driving car, also known as a robot car, autonomous car, or driverless car, is a vehicle that is capable of sensing its environment and moving with little or no human input. Autonomous cars combine a variety of sensors to perceive their surroundings, such as radar, Lidar, sonar, GPS, A odometry and inertial measurement units. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles .

CONCEPT OF AUTONOMOUS DRIVING

A car capable of autonomous driving should be able to drive itself without any human input. To achieve this, the autonomous car needs to sense its environment, navigate and react without human interaction. A wide range of sensors, such as LIDAR, RADAR, GPS, wheel odometry sensors and cameras are used by self-driving cars to perceive their surroundings. In addition, the autonomous car must have a control system that is able to understand the data received from the sensors and make a difference between traffic signs, obstacles, pedestrian and other expected and unexpected things on the road .

For a vehicle to operate autonomously several real-time systems must work tightly together . These real-time systems, include environment mapping and understanding, localisation, route planning and movement control. For these real-time systems to have a platform to work on, the self-driving car itself needs to be equipped with the appropriate sensors, computational HW, networking and SW infrastructure .

For a machine to be called a robot, it should satisfy at least three important capabilities: to be able to sense, plan, and act . For a car to be called an autonomous car, it should satisfy the same requirements . Self-driving cars are essentially robot cars that can make decisions about how to get from point A to point B. The passenger only needs to specify the destination, and the autonomous car should be able to take him or her there safely.

SELF – DRIVING VEHICLES

The following sensors should be present in all self-driving cars:

Global positioning system (GPS). Global positioning system is used to determine the position of a self-driving car by triangulating signals received from GPS satellites . It is often used in combination with data gathered from an IMU and wheel odometry encoder for more accurate vehicle positioning and state using sensor fusion algorithms.

Light detection and ranging (LIDAR). A core sensor of a self-driving car, this measures the distance to an object by sending a laser signal and receiving its reflection . It can provide accurate 3D data of the environment, computed from each received laser signal. Self-driving vehicles use LIDAR to map the environment and detect and avoid obstacles .

Camera. Camera on board of a self-driving car is used to detect traffic signs, traffic lights, pedestrians, etc. by using image processing algorithms .

RADAR. RADAR is used for the same purposes as LIDAR. The advantages of RADAR over LIDAR are that it is lighter and has the capability to operate in different conditions .

Ultrasound sensors. Ultrasound sensors play an important role in the parking of self-driving vehicles and avoiding and detecting obstacles in blind spots, as their range is usually up to 10 metres .

Wheel odometry encoder. Wheel encoders provide data about the rotation of car's wheels per second. Odometry makes use of this data, calculates the speed, and estimates the car's position and velocity based on it. Odometry is often used with other sensor's data to determine a car's position more accurately.

Inertial measurement unit (IMU). An IMU consists of gyroscopes and accelerometers. These sensors provide data on the rotational and linear motion of the car, which is then used to calculate the motion and position of the vehicle regardless of speed .

On-board computer. This is the core part of any self-driving car. As any computer, it can be of varying power, depending on how much sensor data it has to process and how efficient it needs to be. All sensors connect to this computer, which has to make use of sensor's data by understanding it, planning the route and controlling the car's actuators. The control is performed by sending the control commands such as steering angle, throttle and braking to the wheels, motors and servo of the autonomous car.

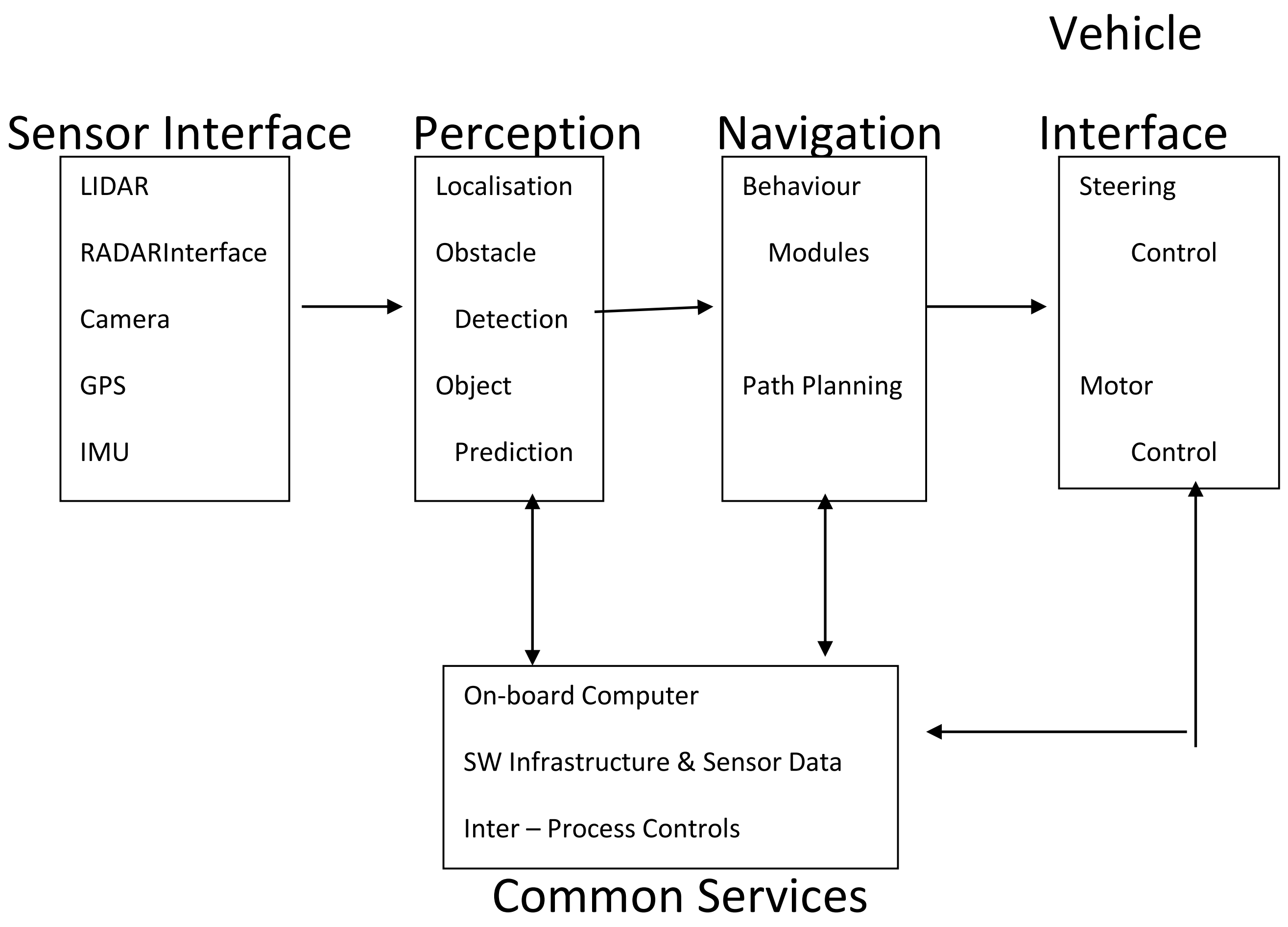

Figure 1 illustrates the SW block diagram of the standard self-driving car.

Each block seen in Figure 1 can interact with other blocks using inter-process communication (IPC) and identified the following blocks for the SW block diagram of a typical self-driving car:

Sensor interface modules. All communication between sensors and the car is performed in this block, as it enables data acquired from sensors to be shared with other blocks.

Perception modules. These modules process perception data from sensors such as LIDAR, RADAR and cameras, then segment the processed data to locate different objects that are staying still or moving.

Navigation modules. Navigation modules determine the behaviour of the self-driving car, as they have route and motion planners, as well as a state machine of car's behaviour .

Vehicle interface. This interface's goal is to send control commands such as steering, throttle and braking to the car after the path has been plotted in the navigation module.

Common services. Common services module controls the car's SW reliability by allowing logging and time-stamping of car's sensor data.

AGENT & ITs ENVIRONMENT

An agent: "it is a computer system situated in some environment, and capable of autonomous action in this environment in order to meet its design objectives". First, an agent has to be autonomous; it means that it must be able to act on its own. Second, an agent has design objectives, goals that it tries to reach. Indeed, we can distinguish between two key characteristics of an agent: its reactivity, its ability to perceive the environment and to respond to changes in it, and its ability to take initiatives and act towards its goal.

To create **synthetic** self awareness we need to create an "observer agent " that evaluates machine learning answers (from multiple ML applications using different ML techniques) so it **learns** and **retains** the derived knowledge.

The "observer agent "must be independant from the ML applications and it must able to explain what they are learnings and how they determined best answers.

The information coming from the robot car senses, especially the sense of moving objects from which the Autonomous Vehicle will learn troubles as well dangers.

Eventually you will have to define an artificial agent, with sensors , and define an environment in which the agent will live (whether it will be the real world or an artifcial world). The sensors will transfer the perceptions of the agent from the environment to the mind of the agent, to create a mental state.

Given a mental state, the agent may select one or more predefined actions to perform in the environment (including the DoNothing action). Some of the algorithms (techniques) we have learned will be applied to the agent to improve its ability to select appropriate actions given its mental state. These are the learning algorithms.

REINFORCEMENT LEARNING

Reinforcement Learning is a type of machine learning that allows us to create AI agents that learn from the environment by interacting with it. Just like how we learn to ride a bicycle, this kind of AI learns by trial and error. The brain represents the AI agent, which acts on the environment. After each action, the agent receives the feedback. The feedback consists of the reward and next state of the environment. The reward is usually defined by a human. We can define reward as the distance from the original starting point of the bicycle.

DEEP REINFORCEMENT LEARNING

Google's DeepMind in which they introduced a new algorithm called **Deep Q Network** (DQN for short) . It demonstrated how an AI agent can learn to play games by just observing the screen without any prior information about those games. This opened the era of what is called 'deep reinforcement learning', a mix of deep learning and reinforcement learning. In *Q-Learning Algorithm*, there is a function called **Q Function**, which is used to approximate the reward based on a state. We call it **Q(s,a)**, where Q is a function which calculates the expected future value from state *s* and action *a*. Similarly in *Deep Q Network* algorithm, we use a neural network to approximate the reward based on the state.

AUTONOMOUS DRIVING ON A HIGHWAY TO PRODUCE SPECIFIC DRIVING BEHAVIOUR

This is an application of Self-Learning for autonomous vehicle control to produce specific driving behavior on a Highway.

The first step is to set the highway environment; an arbitrary number of straight lanes with exits

The goal is to study the behavior of autonomous cars – individual self-learning cars – in traffic, who make decisions based on their current situation on the highway.

The key part of our environment is the highway. It is defined by its structure – that is, the lanes, the exits, the size of the lanes, etc – and by the cars driving on it and also the dynamics of cars movement and happening of car crashes.

The highway can be seen as a group of lanes and a lane can be seen as a group of cells. The position of a car on the highway is defined by the lane and the cell of that lane that it is in. To simplify the highway is defined by the lane index as the y coordinate and the cell index as the x coordinate.

The right-end lane of the highway is the exit lane. It is a lane where drivers go only to take an exit. As soon as a car reaches an exit, it is considered out of the highway and consequently out of the environment. The same goes for cars that reach the end of a lane This is an attempt to mimic real life where highway exits are indicated multiple times with the remaining distance to that exit. The distance between two consecutive exits is always the same, as well as the distance between the start of the highway and the first exit.

There are four types of cells: car cells where agents drive, exit cells where agents can leave the highway, exit indication cells that indicate the current and next exits, and off-road cells, cells on the exit lane that are not exits nor indications. Cars cannot go on these cells and movements passing by or going to these cells are never considered by the drivers.

Each car has two attributes: its speed and its acceleration that can change at every time step t. The speed v_t and acceleration α_t are bounded; the speed cannot be negative or greater than the cars' maximum speed v^{max} and the acceleration cannot be greater, in absolute value, than the cars' maximum acceleration α^{max}. These attributes are discrete variables and can only take integer values. The speed at some time step *t* depends on both the speed and the acceleration of the previous time step $t - 1$. For example, when the speed $v_t = v^{max}$, the acceleration α_t can only be negative or equal to zero.

Movements

At each time step, a car *c*'s *x* position, denoted by $x_{c,t}$, is updated according to its speed $v_{c,t}$ and its acceleration $\alpha_{c,t}$ while its *y* position (i.e. the lane), denoted $y_{c,t}$, depends on their chosen direction, denoted $d_{c,t}$.

Regardless of the direction, the length $l_{c,t}$ of the car's move at the time step *t* is equal to the car's updated speed, $v_{c,t+1}$ of that time step and corresponds to the number of cells that it will advance.

$$l_t = v_{t+1} = v_t + \alpha_t$$

The *x* position is then updated as follows:

$$X_{c,t+1} = x_{c,t} + l_{c,t}$$

When the new speed v_{t+1} is equal to 0, the destination cell will be the initial position of the car at that time step.

The y position is updated according to the direction $d_{c,t}$. The car can only change lane when the movement length $l_{c,t}$ is greater than 0. Otherwise, the car is still and does not move.

$$y_{c,t+1} = \begin{cases} y_{c,t} - 1 \text{ if } d_{c,t} = LEFT \text{ and } l_{c,t} > 0 \\ y_{c,t} + 1 \text{ if } d_{c,t} = RIGHT \text{ and } l_{c,t} > 0 \\ y_{c,t} \text{ if } d_{c,t} = FORWARD \text{ or } l_{c,t} = 0 \end{cases}$$

All directions are not always possible; when a car is in the left-end lane, it can only go forward or to the right, as turning left would lead outside the highway. Similarly, when it is on the right-end lane, it can only go forward or turn left unless it is taking an exit when going to the right. Finally, we add an attribute to the car, the status of their blinkers, denoted bc,t, that indicates which direction the car is taking:

$$b_{c,t} = \begin{cases} LEFT \text{ if } d_{c,t} = LEFT \text{ and } l_{c,t} > 0 \\ RIGHT \text{ if } d_{c,t} = RIGHT \text{ and } l_{c,t} > 0 \\ null \text{ if } d_{c,t} = FORWARD \text{ or } l_{c,t} = 0 \end{cases}$$

Crashes

When considering two cars c_1 and c_2 and their respective initial position $(x_{c1,t}, y_{c1,t})$ and $(x_{c2,t}, y_{c2,t})$ and destination $(x_{c1,t+1}, y_{c1,t+1})$ and $(x_{c2,t+1}, y_{c2,t+1})$, we have to determine whether they are crossing paths at the same time step and crashing.

When a car moves from the position (x_t, y_t) to the position (x_{t+1}, y_{t+1}) at some time step, it does not disappear from its initial position to appear in the destination afterwards. The car passes by intermediary positions or *passing cells*, denoted *PC*. For the forward direction, these passing cells are easily defined as the cells located between the starting and destination cells of the move:

$$PC_{t+1} = {}^{,} (x, y_t) \mid x \in [x_t + 1, x_{t+1} - 1]$$

For the other directions, when the car changes lanes, we consider that the car effectively crosses the lane lines in the middle of the movement; at this moment, the car is considered to be in both lanes. Before that, the car is still in its initial lane, and after that it is in the new lane. The passing cells are

$$PC_{t+1} = \begin{array}{c} :\{.\ (x,y_t) \mid x \in (x_t + 1, x_t + (l_t + 1)/2\ )\ \} \\ \cup \\ \{\ (x,y_{t+1}) \mid x \in [x_t + (l_t / 2), x_{t+1} - 1]\ \} \end{array}$$

Two cars crash when they are in or pass by the same cell during the same time step t. A car c1 and a car c2 will crash when

$$\{PC_{c_1,t+1} \cup (x_{c_1,t+1}, y_{c_1,t+1})\} \cap \{PC_{c_2,t+1} \cup (x_{c_2,t+1}, y_{c_2,t+1})\} \neq \emptyset$$

There are two possible situations where cars cross paths and consequently crash. One situation could be when a car's destination is one of another car's passing cells.

Environment's States

A state s of the highway is a list of all the cells of this highway, with information about what is in the cell at that moment or about what type of cell it is. We denote Hx,y the cell located at the xth position on the yth lane. For off-road and exit cells, the only information is the type of the cell. For exit indication cells, the information is the exits indicated: the number of this exit ei and of the next one ei+1. Finally, for car cells, the information can be one of three things. If there is a car, the information is the speed, acceleration, and blinkers' status of the car. If there is a crash, the information is simply an indication that there is a crash. If there is nothing, the information indicates as much. By unifying everything, we have:

A highway cell Hx,y is a tuple (o,e,v,α,b,i1,i2) where

• o indicates if the cell is an off-road cell (1) or not (0);

• e indicates if the cell is an exit cell (1) or not (0);

• v indicates the speed of the car currently in the cell, if any;

• α indicates the acceleration of the car currently in the cell, if any;

• b is the blinkers' status of the car currently in the cell, if any;

• i1 is the number of the indicated exit if the cell is an indication cell;

• i2 is the number of the next exit, if any, if the cell is an indication cell

We can now formally define a state s at a time step t: $s_t = \{H_{t,x,y} \mid 0 \leq x < X, 0 \leq y \leq Y\}$

Where Ht,x,y is the highway cell at the position x,y for the time step t, X is the size of the lanes, and Y is the number of lanes.

Actions

The car can choose a direction directly as it corresponds to turning the wheel in a certain way. This direction can be left, right or forward. However, the car cannot choose the speed they want to have; changing speed means accelerating or decelerating.

An action is consequently composed of a direction dt and an acceleration αt. This acceleration is bounded. In conclusion, the set of possible actions A is:

$$A = \{(d,\alpha) \mid d \in \{LEFT, RIGHT, FORWARD\}, -\alpha_{max} \leq \alpha \leq \alpha_{max}\}$$

At a given time step t, some actions can be "impossible". Namely, when an action's corresponding move is forbidden in our system, this action will not be considered.. A forbidden move is a situation where the destination or one of the passing cells of that move is outside the highway; this includes the off-road cells of the exit lane. Moreover, actions whose acceleration will make the updated speed outside the speed bounds defined are also considered impossible. For example, when αmax = 1, the set of actions is:

$$A = \left\{ \begin{array}{c} (\text{LEFT}, -1), \\ (\text{LEFT}, 0), \\ (\text{LEFT}, +1), \\ (\text{FORWARD}, -1), \\ (\text{FORWARD}, 0), \\ (\text{FORWARD}, +1), \\ (\text{RIGHT}, -1), \\ (\text{RIGHT}, 0), \\ (\text{RIGHT}, +1) \end{array} \right\}$$

Furthermore, the length of the action set depends on the number of directions, which is always 3, and the number of possible accelerations. $|A| = 3 \times (2\alpha max\ + 1)$.

Lanes' Preferred Speed

To overtake another car, a car must go to the lane on the left and drive faster. This leads to the observation that the average speed in a lane should be higher the farther to the left this lane is.The preferred speed of a lane between these two extremes will depend on both the number of lanes (excluding the exit lane) and the number of possible speeds (excluding 0). We denote the i^{th} lane L_i and its preferred speed v^{L_i} . We want to guarantee a fair and balanced distribution of the preferred speeds across the lanes. The simplest case is when the number of lanes Y is a multiple of the number of possible speeds m. In such cases, each speed v^k will be the preferred speed of $n_V k$ = Y/ m lanes, ordered from highest (left-end lane) to lowest (right-end lane). For example, if Y = 6 and m = 3, the preferred speeds of the lanes, from left to right, will be 3, 3, 2, 2, 1, 1. When Y is not a multiple of m, $n_V k$ is not the same for all speeds v^k. Some speeds will be the preferred speeds of more lanes. We decided that these speeds should be the higher ones. For example, if Y = 4 and m = 3, the preferred speeds of the lanes will be 3, 3, 2, 1. The number of speeds that appear more times as a preferred speed is given by the rest of the division of Y by m..

With this construction, the preferred speeds' distribution will always respect the following conditions

$$n_{v^k} = \{\ [\ Y/m\ ] \text{ if } k \leq (Y \bmod m\ \},\ [Y/m] + 1 \text{ otherwise}$$

In conclusion, once we know how many times each speed should appear as a lane's preferred speed, we just need to distribute them correctly, by respecting the first condition.

Generating Traffic

A driver arrives actually in Highway refers to when – which time step – and where – which lane – they arrive in the environment. To do so, we use a new parameter for the highway, the traffic density, denoted τ, that serves as the probability, at each time step and for each lane, that a new driver is arriving in this lane.

The traffic generation process is summarized as below :

```
For all lane do
        if lane’s initial position is free then
            if random < τ, with probability τ then
                driver ← randomly initialized driver
                driver enters lane
            end
        end
end
```

Highway Steps

The highway is updated sequentially from right to left. To implement this, we divide a highway time step into two “sub-steps”, that we call the observation step where the drivers observe the environment and choose an action, and the update step where we update the highway’s state by executing the drivers’ actions. The observation step is done sequentially from right to left. During this step, the drivers choose their action; if they are going to change lanes, their blinkers are turned on so that drivers behind them can know which direction they are going to take. After the observation step, all drivers have chosen their action, and we can now update the highway: this is the update step, which is done sequentially from left to right. We update the highway from left to right because we want the crashes that can happen in different cells to actually happen in the first possible cell – starting from the left.

Highway Parameters

We can identify the parameters of the highway.

• Number of lanes $X > 0$, $L \in N$

• Size of the lanes $Y > 0$, $C \in N$

• Number of exits $E \geq 0$, $E \in N$

• Size of the exits $Se > 0$, $Se \in N$

• Size of the space between two exits $Ss > 0$, $Ss \in N$

• Crash duration $Tf > 0$, $Tf \in N$

• Traffic density $\tau \in [0,1]$

• Cars’ maximum speed $vmax > 0$, $vmax \in N$

• Cars’ maximum acceleration $\sigma max > 0$, $\sigma max \in N$

Highway Performance

We have a fully functional Highway with traffic and, we would like to see how well it performs and mirrors real-world traffic. First, we have to define what a good performance is; we want to maximize the ratio of cars that reach their goal and minimize the number of crashes. It is clear that the result of a Highway depends on the chosen parameters . Hence, we focus on what we consider to be the most important ones: the traffic density. We run a series of tests to analyze the percentage of each outcome and. to do that, we fixed the other parameters:

• Number of lanes = 5

• Size of the lanes = 80

• Number of exits = 2

• Size of the exits = 5

• Size of the space between two exits = 7

• Crash duration = 10

• Cars' maximum speed = 3

• Cars' maximum acceleration = 2

The possible outcomes are called goal if the driver reaches their goal, crash if they crash and missed goal if the driver does not reach their goal (e.g. misses their exit). We run the Highway for 2000 time steps and repeat it 20 times to compute the average.

The autonomous car enters the Highway without any prior knowledge of highway dynamics, other cars and its own position. After a brief period of stay on the highway, the autonomous agent appears experimenting with different movements but does not yet produce any recognizable output.

The Highway has been implemented in Python .

Autonomous Cars

Agents – Attributes

We opt for the agents; they have the attributes: the sight and the goal. While the goal is chosen randomly when an agent arrives on the highway, the sight is always fixed to the some value. The other noticeable fact is that our learning agents do not have a desired speed. We define the autonomous cars as entities whose primary concern is to avoid crashing; they should consequently not exhibit any preference for a certain speed as long as they are driving safely. Furthermore, we add an attribute ϵ to these learning agents; this is their probability of choosing a random action at each time step.

Given that we define learning agents the same way as the human drivers, we can seamlessly add them to the highway. The only difference is how they will choose an action: by using their learning model, a neural network. We can therefore adapt the highway's time step's algorithm to take the learning agent into account for the observation step. To decide what action it should take, the reinforcement learning agent uses a neural network to approximate the Q-function . Thus, at every time step t, the agent c observes its state $s_{c,t}$; this state is then processed in some way so that it can be passed to a neural network whose outputs correspond to all the possible actions. The values of these outputs are the estimated Q-values, Q($s_{c,t}$, a); as it is using a neural network θ, we denote the Q-function approximated with that network by Q(s,t;θ). The agent then uses an ϵ-greedy strategy to choose the action $a_{c,t}$.

Rewards

We need to define the reward function R; there are three different final states the agents can be in. First, they can reach their goal, whether it is taking an exit or not. Second, they can miss their goal; they either took a wrong exit or missed their exit. Finally, they can crash. Hence, we define the following rewards:

- ρ_ω , the reward received by the agent when it reached its goal
- $\rho_{\bar{\omega}}$, the reward received by the agent when it failed to reach its goal but did not crash
- ρ_f , the reward received by the agent when it crashed

Since reaching the goal and crashing are completely opposite outcomes, we define $\rho_\omega = - \rho_f$. Moreover, since missing the goal but not crashing is preferable to causing an accident but is a less desirable outcome than reaching the goal, the value of this reward should be smaller, such that $\rho_f = -\rho_\omega < \rho_{\bar{\omega}} < \rho_\omega$. The agent gets these final rewards at the time step that effectively ends its run on the highway. For all the other previous time steps, it receives a default reward that is always equal to 0. However, the agent receives another reward ρ0, a small penalty, when the agent's speed is 0; our environment is a highway and cars should not be still, unless there is congestion.

Required Information

We start by defining the minimum amount of information that an autonomous agent should have. Consequently, the encoding on the highway will possess these pieces of information. They are:

• The current lane that the agent is in

• The current speed of the agent

• The goal of the agent

• What the agent sees; cars, crashes and exit indications

Time-Step Encoding

The encoding is based on the idea as the cars presence at different time steps; we use information about the previous time step (the cars' presence represented by the observation matrix O) instead of the current speed of the cars. The observation matrix of the previous time step t−1, denoted Ot−1, is not additional inputs, but it forms, along with the current observation matrix, a 3-dimensional matrix with time as the third dimension. We then pass this matrix through a 3-dimensional convolutional neural network . We also keep decreasing the number of inputs by including the learning agent itself in its observation matrix. To differentiate itself from the other cars, the value is not 1 but 0.5. This way, the only additional information required are the exits and the agent's goal. Figure 2 illustrates time-step encoding, as it encodes the cars' presence at different time steps.

The learning agent is shown in orange while cars in its field of view are in grey. All the input vectors are concatenated and passed to the network.

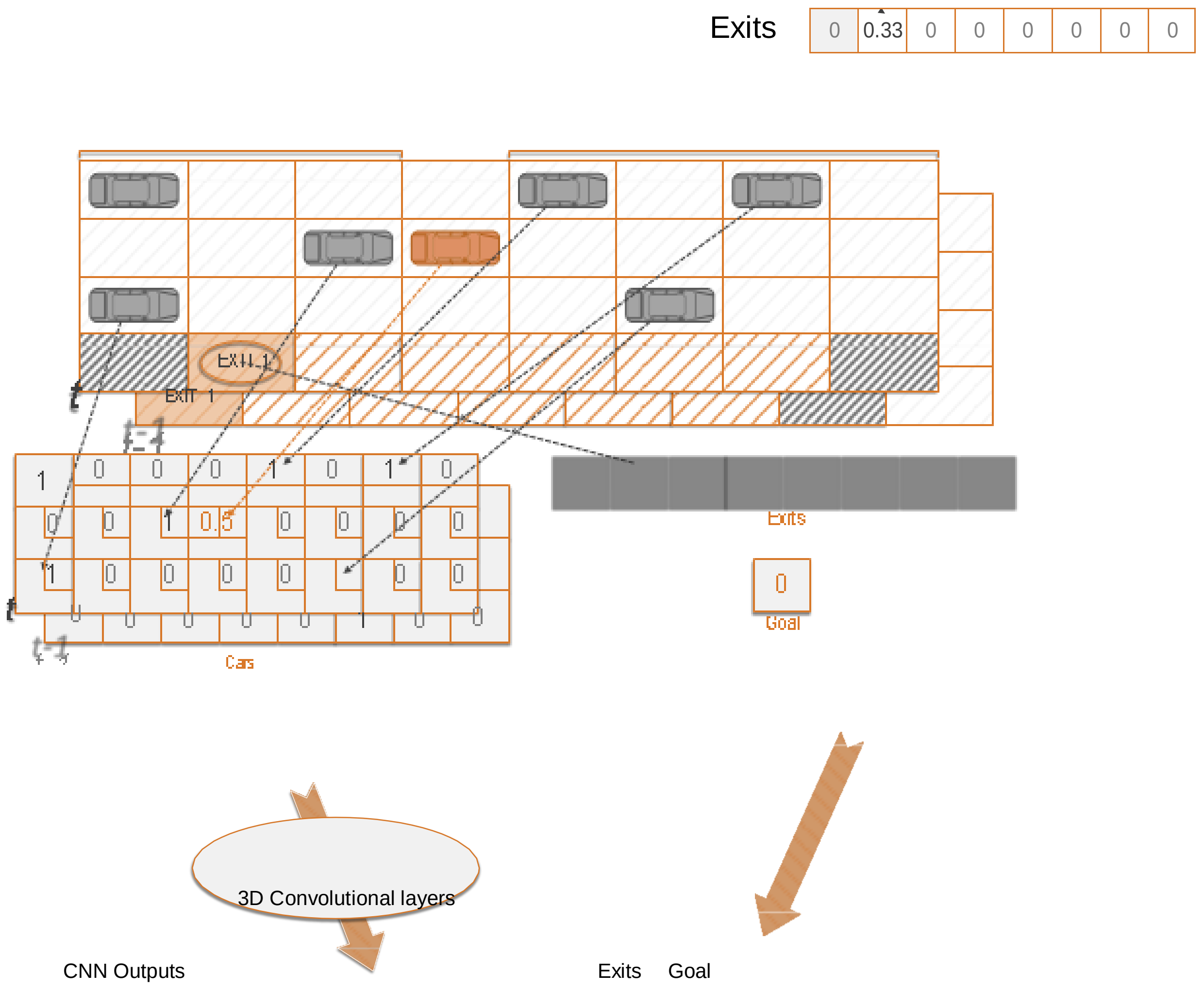

Figure 2 : Time-Step Encoding.

In an attempt to decrease the number of inputs, we now encode the exits in a relative way; instead of having a vector for each exit, we have only one vector, and the value is not binary anymore. If the exit indication in the field of view is, for example, the first exit out of 3, the value will be 1/3 = 0.33. Likewise, the encoding of the goal of the agent is now a single real-value that is 0 if the goal is to continue on the highway, or the relative number of the exit it wants to take.

Single-Agent Setting

In a single-agent environment; there is only one learning agent on the highway, other cars are our simulated human drivers. This has been implemented in Python with the module Keras with TensorFlow as back-end.

Now that we have defined our neural network, a machine learning technique, and the highway, it is time to tackle our main problem: making an autonomous car learn on its own how to drive.

The training environment set up for our learning agent is divided into episodes. One episode consists of a full run of the agent on the highway.

First, as the agent could potentially stay still indefinitely, if it always chooses to stay at the speed zero, we need to guarantee that its run on the highway will come to an end. We thus define a maximum number of steps T^{max}; if the agent's run on the highway reaches this upper bound, we consider the run over: the overtime outcome. Furthermore, the value of T^{max} must be chosen wisely.

The bound must not be too low nor too great. As the time required for an agent to leave the highway depends on the size of the lane X, we need to define the overtime bound based on this value. We could then use twice that value, 2×X, but it means that the agent could potential stay still half of the time and still finish its run. Finally, we define arbitrarily T^{max} as 2×X −10; we want the agent to actually drive more often than it is not moving.

Second, we want our learning agent to arrive in a highway after some time, to ensure that it arrives in a situation where it is not the only car on the highway. To do that, we perform an arbitrary number of highway time step updates, usually 20, before adding the agent in the system. Then, we randomly initialize the attributes – speed v and lane y – of the agent and add it on the highway. Note that if it cannot, for some reason, be placed in its lane, we perform other highway time step updates until it is possible.

Finally, we use experience replay to train the network in an online fashion; at each time step, we sample a batch of experiences from the memory and train the network with it.

This whole process of learning is formalized in below Algorithm.

```
Algorithm : Single-agent learning algorithm
  Initialize replay memory D of length $M$
for $episode = 1, ..., N$ do
    Initialize highway $H$ according to fixed highway parameters
    for $t = 1, ..., 20$ do
        Perform one time step update of the highway $H$
    end
    Initialize learning agent $c$ with a fixed sight $\varphi_c^+$ and $s$
    Choose randomly lane $y$ in which to add the agent, and speed $v$ of the agent
    while agent cannot enter lane $y$ do
        Perform one time step update of the highway $H$
    end
    Add the learning agent $c$ on the highway
    for $t = 1, ..., T^{max}$ do
        Perform one time step update of the highway $H$
        /* Experience replay */
        Observe the state $s_t$, action $a_t$ and reward $r_t$ of the agent for that time step, and the
          next state $s_{t+1}$
        Store experience $(s_t, a_t, r_t, s_{t+1})$ in D
        Create empty neural net model with parameters
        if replay memory D is full then
            Sample minibatch of size $B$ from experience memory D
            Initialize batch learning buffer of size $B$
            Load the neural net model
            for experience $(s_k, a_k, r_k, s_{k+1})$ in minibatch do
                Get network outputs $\mathbf{y}$ according to $Q(s_k, a; \theta)$
```

$$\text{Set } y_{a_k} = \begin{cases} r_k & \text{if state } s_{k+1} \text{ is final} \\ r_k + \gamma \max_{a^j} Q(s_{k+1}, a^j; \theta) & \text{if state } s_{k+1} \text{ is not final} \end{cases}$$

```
                Store $(s_k, \mathbf{y})$ in batch learning buffer
```

```
            End
            Train network $\theta$ against batch learning buffer
          end
        end
     end
```

Training Parameters

A training experiment is defined by numerous parameters of the highway, and all the parameters regarding the learning agent and the training algorithm. These parameters are:

- φ_c^+ , the sight of the learning agent (its backsight $\varphi_c^- = \varphi_c^+ - 1$)
- s, the agent's probability of choosing a random action
- R, the reward function; which corresponds to defining the different rewards ρ_ω, $\rho_{\bar{\omega}}$, ρ_f, ρ_0 and ρ_T
- μ_c, the neural network model used by the agent
- γ, the discount factor as defined for the MDP
- α, the learning rate for the neural network training
- N , the number of training episodes
- M , the size of the experience memory buffer
- B, the size of the batches of experiences

Moreover, the hidden structure of the neural network model μ_c can also be chosen; this includes the number of hidden layers, the number of neurons in these layers, their activation function, and the dropout rate δ_c (0 if we do not want to use dropout) to apply to each layer. Finally, if the chosen model is a CNN one, the structure of the convolutional networks can also be set: the number of convolutional layers with their stride and the number and size of the filters.

Learning – sampling batches of experiences to update the network's weights – starts once the experience memory buffer is full: after encountering M experiences.

Experiments

For our experiments, we trained our model with TensorFlow as back-end. We also considered different possible hidden structures for the neural networks.

Settings

For our experiments, most of the parameters are fixed. We present them in Table 1. The structure of the CNNs are also fixed: there are two convolutional layers, the first with a stride of 1 and 16 filters of size 4×4, and the second with a stride of 1 and 32 filters of size 2×2.
Finally, the reward system of our learning agent is fixed, and the values are shown in Table 2.

Results

The learning results are the evolution of the percentage of outcomes (accumulated) throughout the training process.

Highway parameters	
Number of lanes Y	3
Lane size X	40
Number of exits E	0
Exit size S_e	5
Space size S_s	7
Crash duration D	10
Traffic density τ	0.25
Cars' maximum speed v^{max}	3
Cars' maximum acceleration α^{max}	2

Learning parameters	
Number of episodes N	50000
Batch size B	25
Experience memory size M	50
Learning rate α	0.01
Discount factor γ	0.9
Agent's sight φ_c^+	6
Agent's s	0.05

Table 1: Single-agent training's fixed parameters

	Reward ρ	**Value**
Goal	ρ_ω	+1
Missed goal	$\rho_{\bar{\omega}}$	-0.15
Crash	ρ_f	-1
No speed penalty	ρ_0	-0.01
Overtime	ρ_T	-0.4

Table 2: Single-agent training's reward system

We tested three different configurations for the hidden structures of the networks:

• 2 hidden layers: the first with 60 neurons and a tanh activation function; the second with 30 neurons and a linear activation fucntion. No dropout.

• 2 hidden layers: the first with 150 neurons and a tanh activation function; the second with 75 neurons and a linear activation fucntion. No dropout.

• 2 hidden layers: the first with 300 neurons and a tanh activation function; the second with 150 neurons and a linear activation fucntion. No dropout.

The results for our CNN based model – a modern machine learning technique – are mixed and the learning agent is able to reach the goal with low success rate. The networks that do not use dropout seem to learn well. The percentage of goal reached for the networks (without dropout) is high and increases with the number of hidden neurons. As the training is done for fewer episodes, the models aptitude to learn and adapt on its own is limited.

CONCLUSION

Autonomous Driving systems are complex and present a challenging environment. Self-Thinking for Autonomous Systems development is very promising where the required behaviours to be learned from scratch and further improved from deep learning technique. An overview of autonomous vehicles and its components is provided. Designed neural network model with convolutions and Q-learning in order to solve the problem of autonomous driving to produce specific driving behavior on a highway with deep learning. At first the autonomous agent enters the highway without prior knowledge of highway dynamics, other cars and its own position. After a brief period of stay on the highway, the autonomous agent experiments with different movements and then used deep learning, a modern machine learning technique, to create internal model to accurately predict movements. As the training is done for fixed number of episodes, the models aptitude to learn and adapt on its own is limited.

REFERENCES

1. R. Kwiatkowski el al., "Task-agnostic self-modeling machines," *Science Robotics* (2019).
2. Manon Legrand : “Deep Reinforcement Learning for Autonomous Vehicle Control among Human Drivers “. Universitas Bruxellensis