Neural Networks in Autonomous Driving

Cristina Tîrnăucă, José L. Montaña, Zoraida Mediavilla
Universidad de Cantabria
Santander 39005, Spain

Abstract
In a society in which more and more complex tasks are undertaken by robotics, and the use of electronic devices continues to grow, the ability of computers to learn by observing human behavior becomes crucial. In this paper, we show how artificial intelligence and, in particular, neural networks, can be used to imitate the behavior of a human while piloting a miniature vehicle via a remote controller.

Introduction
Recently, the words smart and intelligent has been associated with a variety of devices to indicate that they are now endowed with “intelligence”. Many companies develop their own products, sometimes imitating others and other times proposing something new. Watches, cities, houses, cars and even home appliances... many of them already include this term, and it is not only the companies that are manufacturing their products, but it has become increasingly common for people in their homes to manufacture their own machines, in the so-called “maker movement” which follows the motto of “do it yourself” (DIY).

This technology is partly developed thanks to the large advances in hardware of the last years. The number of low cost small-sized powerful devices is increasing day by day, and this is a huge impulse in their proliferation. This is also the case for electronics such as Arduino, Raspberry Pi, UDOO, Banana Pi, BeagleBone, ODroid, etc., which allows the set up of home-made projects with the desired characteristics within a small budget. In this article we show how a homemade toy-sized vehicle powered by a Raspberry Pi can be used to observe and record the behavior of a human that pilots it via a remote controller. In the end of this observation phase, the vehicle should possess enough knowledge to become independent and drive autonomously, as long as the environment in which the training took place does not change substantially.

Learning from observation is an attractive and promising alternative to handcrafting or simulating behaviors, and has lately attracted a lot of attention (see, for example, (Bentivegna, Atkeson, and Cheng 2004; Fernlund et al. 2006; Mehta et al. 2009; Moriarty and Gonzalez 2009; Dereszynski et al. 2011; Floyd, Bicakci, and Esfandiari 2012; Tîrnăucă et al. 2016)).

In the automobile industry, many of the biggest car producers are already developing their own autonomous vehicles, endowed with the ability to drive by themselves, while interacting with all kinds of objects (traffic lights, other cars, road markings, etc.) or people (pedestrians), and obeying certain traffic rules. They mostly rely on hardware technologies to achieve good results. Commonly used devices are: radars (to compute the speed of objects), GPSs (to have the exact location), cam recorders (to identify traffic signs, pedestrians or other vehicles), accelerometers and other specific mechanisms (for example, to record the behavior of the wheels). In addition, some models have added the ability to communicate with other vehicles that are relatively close to their position.

We propose an easy to implement/use approach, very similar in spirit to the ALVINN (Autonomous Land Vehicle in a Neural Network) project (Pomerleau 1989), in which the output of the neural network is given by the wheels’ movement and not by a linear representation of the direction the vehicle should travel, as in the original project. Also, in the training phase, we do not require fixed, predefined features (like the road’s center line) for accurate driving. Artificial neural networks were used in other related settings, such as learning to control cars in a racing game by observation (Muñoz, Gutierrez, and Sanchis 2009) or learning the player’s behavior by observing a first person shooter (FPS) game (Thurau, Bauckhage, and Sagerer 2003).

Prototype Design
The main component of the vehicle we build in order to simulate a human driver in action is a Raspberry Pi, which is a credit-card sized affordable single-board computer, originally created as an education tool, nowadays very much used in home automation and robotics.

The Raspberry Pi was glued on a chassis with two driving wheels and a third stability wheel, assembled as a differentially steered drive system; steering the vehicle is acquired by varying the speeds of the drive wheels. Two DC (direct current) motors to convert electrical power into mechanical power, one additional controller and an external battery (needed to amplify the insufficient power provided by the 5V GPIO pins of the Raspberry Pi) were also added to the
chassis. The device was equipped with an HC-SR04 ultrasonic sensor that measures the distance to the nearest object and a basic web-cam. The final prototype is presented in Figure 1.

The operating system installed on the Raspberry Pi is a Linux distribution. All coding necessary for the project was done in Python 2.7, and the OpenCV library was used for image processing.

Learning from Observation

In this project, we propose the use of artificial neural networks (ANN) to learn the behavior of a human driver that pilots a vehicle via a remote controller (see Zhang 2010 for a detailed survey on ANN). More precisely, we will use a particular type of ANN called multi-layer perceptrons (MLP), which is one of the most widely studied and used ANN in practice. An MLP is formed by a large connection of interconnected computing units called neurons, which are organized in layers as in Figure 2. The first one is the Input Layer, the middle ones are Hidden Layers and the last one is called Output Layer.

![Figure 2: The structure of an MLP](image)

The input of the network in this case is given by the image captured by the web-cam and the distance to the nearest obstacle measured by the ultrasonic sensor. The hidden layers can contain an arbitrary number of neurons. The output consists of the state of the two motors (turned on/off) and the type of movement produced (stop, move forward or move backward).

Training an MLP with a very large input layer (as in the case of an image) can be time consuming and can lead to the overloading of the existing memory capacity. An image can be shaped into an array of $p \times q$ pixels, each pixel being a 3-dimensional vector in $[0,255]^3$. Reducing the size of the image can be done using one or more of the following strategies:

- by gray-scaling the image, each pixel becomes a number in $[0,255]$;
- by defining a grid and replacing each block of $k \times k$ pixels with one unique pixel that holds the average of the $k^2$ values;
- by performing principal component analysis (PCA), as described below.

The objective of PCA is reducing the dimensionality of the input data from $\mathbb{R}^n$ to $\mathbb{R}^k$, while being in control of the amount of information lost in the process. For that, one needs to compute first the co-variance matrix,

$$
\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)}) (x^{(i)})^T
$$

where each $x^{(i)}$ is an $n$-dimensional vector representing one image.

Then, let $U = (u_1, u_2, \ldots, u_n)$ be the square matrix containing as columns the $n$ eigenvectors of $\Sigma$. Out of these $n$ vectors, the first $k$ are used to compute the projection $z^{(i)} \in \mathbb{R}^k$ of an original point $x^{(i)} \in \mathbb{R}^n$ with the formula $z^{(i)} = (U_k)^T \ast x^{(i)}$, where $U_k = (u_1, \ldots, u_k)$. Then, the approximate value $x^{(i)}_{\text{approx}}$ can be obtained as $U_k \ast z^{(i)}$.

Choosing the right $k$ depends on the quantity of information that one wants to preserve. The next formula corresponds to 1% error rate, which is what we used in our experiments:

$$
\frac{1}{m} \sum_{i=1}^{m} \left\| x^{(i)} - x^{(i)}_{\text{approx}} \right\|^2 \leq 0.01
$$

Experiments and Results

In the experimentation phase, we guided the vehicle through distinct environments, avoiding the obstacles and driving backwards whenever the prototype was approaching a wall. Data was saved to file every four seconds. Each record line would contain as independent attributes a 320x232 RGB image and one real number representing the distance to the nearest object. The two dependent variables recorded are the state of the two motors and the type of movement produced; these values are the ones to be predicted by the MLP. The particular algorithm employed to learn the weights of the MLP (one hidden layer with 10 neurons) was RProp (Riedmiller and Braun 1993).
Reducing the dimensionality of the image was done using all three strategies described in the previous section:

- we first gray-scaled the image to obtain a 320x232 array of natural numbers in [0,255],
- we used an 8 unit grid to further decrease the dimensionality of the array to 40x29,
- we performed PCA while preserving 99% of the variance.

The variable indicating the state of the motors has four possible values: 00, 01, 10 and 11. The left bit corresponds to the left motor and the right bit to the right motor, with a value 1 if the motor is running and 0 otherwise. The type of movement has three possible values: 0 if the vehicle is not moving, 1 if it is moving forward and 2 if it is going backwards. Note that using a -1/0/1 representation for each motor is not the same as using the above mentioned variables, because the directionality of the vehicle depends on the speed of the two wheels.

We performed two types of tests. In the first one, we trained the ANN using the image and the distance. The results were rather poor, indicating that the ANN was unable to learn using this information only. Thus, we performed a second test in which we considered as input the other dependent variable. Out of the total of 1989 entries we made a selection of 628 rows for the prediction of the state of the motors and 618 for the prediction of the type of movement, in order to have equally distributed values in the training and testing sets. A 65% of these entries were used for training (408 and 401, respectively), and 35% for testing (220 and 217, respectively). A confusion matrix with the results obtained for each of the two learning settings is presented in Figures 1 and 2, respectively.

The conclusions we draw is that going backwards can be easily predicted with this kind of information while moving forward seems to be a very hard task.

In the future, we plan to extend our experiments to include information extracted from an accelerometer and one gyroscope. This should allow us a finer analysis with respect to the angle of steering and the speed of the vehicle. Also, an evaluation of the performance of our approach as a learning from observation algorithm remains to be done (see Ontañón, Montaña, and Gonzalez 2014) for relevant evaluation metrics. Moreover, we will consider the use of convolutional neural networks, with the drawback that training cannot occur on the mobile robot due to its limited memory capacity.

Acknowledgments

This work was partially supported by project PAC::LFO (MTM2014-55262-P) of Ministerio de Ciencia e Innovación (MICINN), Spain.

References


