Mountain Car: A study on the cooperative relationship between intelligent control systems

Arthur Matola Moreira
Control and Automation Engineering
UCL - Universidade do Centro Leste
Serra, Brazil
arthurmoreira@ucl.br

Maycon Gomes do Rosário
Computer Engineering
UCL - Universidade do Centro Leste
Serra, Brazil
maycong@ucl.br

João Lucas Conrado
Computer Engineering
UCL - Universidade do Centro Leste
Serra, Brazil
joaolucasc@ucl.br

Mateus Altoé da Silva
Computer Engineering
UCL - Universidade do Centro Leste
Serra, Brazil
mcsuetam@ucl.br

***Abstract*—This article is a study on the relationship between Fuzzy intelligent systems and artificial neural networks. Based on an academic challenge, an analysis was developed on how the union between different intelligent language methods can generate more overwhelming results. The final objective was to improve the systems through ANN training. The conclusion resulted in the understanding that the union between Fuzzy systems and ANN can generate great evolutions in the areas of artificial intelligence.**

***Index Terms*—Fuzzy, RNA, programming, computing, automation.**

I. Introduction

In recent years, there has been a growing interest in exploring the potential of artificial intelligence to enhance decision-making processes across various domains.using the well-known Mountain Car game as a study project. From our fuzzy logic-based project, we created a database of its results and trained the neural network using this data. for its ability to handle uncertainty and approximate reasoning, making it suitable for systems that rely on human-like decision-making. However, with the advent of advanced neural network architectures, there is an opportunity to achieve higher levels of accuracy, scalability, and adaptability. This adaptation involves reinterpreting the fuzzy rules and membership functions into a framework compatible with neural networks, while ensuring the core functionality and purpose of the original system are preserved. The objective is to leverage the learning capabilities of neural networks to improve system performance while maintaining interpretability and robustness.

II. Creating our Data Bank

It's interesting to understand that, in this context, Fuzzy Logic has been employed to create a comprehensive database for decision-making, while Artificial Neural Networks (ANN) have been utilized to refine and optimize the solution. Fuzzy Logic is particularly suitable for handling ambiguous data, allowing for more flexible and nuanced decision processes. By incorporating approximate reasoning, it enables the system to process incomplete or contradictory information, which is often encountered in real-world applications. The integration of Fuzzy Logic in constructing the database allows for the representation of uncertain or qualitative data, which can be essential for modeling complex systems. Through fuzzy sets and membership functions, data points are categorized into degrees of membership, providing a more granular and adaptable approach compared to traditional binary classification systems. This process is particularly beneficial when dealing with variables that are difficult to quantify, such as human preferences, environmental factors, or subjective measurements.

On the other hand, Artificial Neural Networks (ANN) offer a powerful tool for learning from data and making predictions. ANN models, inspired by biological neural networks, consist of interconnected layers of nodes (neurons) that process input data and adjust their weights during training to minimize errors in predictions. By utilizing ANN, the database constructed with Fuzzy Logic can be refined and improved, as neural networks can learn complex patterns and optimize the system's performance based on historical data.

In this study, the combination of Fuzzy Logic and Artificial Neural Networks aims to leverage the strengths of both approaches. Fuzzy Logic contributes to handling uncertainty in the data, while ANN enhances the system's ability to learn from patterns and improve predictions over time. This hybrid methodology has the potential to provide more accurate and robust solutions to problems in various fields, including decision support, forecasting, and control systems.

A. The problem: Mountain Car

The Mountain Car problem is a classic challenge used in the field of reinforcement learning, often adopted as an example to illustrate the application of control algorithms and decision-making processes. The environment consists of a car placed between two hills, with the goal of reaching the higher hill. However, the car lacks sufficient power to climb the hill directly, and thus, it must alternate acceleration techniques to gain enough speed and reach the top. This problem is commonly used in reinforcement learning (RL) studies, particularly to demonstrate the effectiveness of methods such as Q-learning, neural networks, and as we are using, Fuzzy Logic.

B. The first solution: Fuzzy logic

Firstly, as we are working in Python (Google Colaboratory), we can use external libraries to develop Fuzzy. The choice of Fuzzy to create the database is due to the fact that it is possible, using it, to make customizations based on knowledge of the need and the problem.

There are some libraries that are vital for the code to work, so they must be executed from the beginning.

Explaining some important references:

- The scikit-fuzzy library is an extension of the popular scikit-learn and provides tools to work with Fuzzy Logic in Python. It allows for the construction of fuzzy inference systems, fuzzy control systems, and other applications dealing with uncertainties and imprecision in data. Fuzzy Logic is essential in many engineering and scientific fields where decision-making involves ambiguous or qualitative data [1].
- The gym library, developed by OpenAI, is a framework for creating reinforcement learning environments, providing interactive simulations where agents can be trained. The [atari] option installs environments based on classic Atari games, and the [box2d] option installs environments with 2D physics, useful for more realistic simulations like robot and vehicle control. These options are essential for researchers working with reinforcement learning and intelligent agent simulations [2].

```
!pip install scikit-fuzzy

# Gym Dependencies
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
!pip install gym[box2d] > /dev/null 2>&1

# Rendering Dependencies
# !pip install gym pyvirtualdisplay > /dev/null 2>&1
# !apt-get install -y xvfb python-opengl ffmpeg > /
    dev/null 2>&1

!apt-get install xvfb # install Xvfb
!pip install pyvirtualdisplay # install required
    package
```

In the sequence, the presented code snippet uses the scikit-fuzzy library to define the variables of a fuzzy control system applied to the Mountain Car problem. The input variables include the position (posicao) and velocity (velocidade) of the car, represented as Antecedent objects, which are the variables influencing the system's decision. The position is defined within the range of -1.2 to 0.6, discretized at intervals of 0.1, while the velocity varies from -0.07 to 0.07 with a finer resolution of 0.001, allowing for greater precision in control. The output variable, action, is defined as a Consequent object and represents the car's acceleration commands, ranging from 0 to 3, with a discretization of 0.1. This set of variables forms the foundation for developing a fuzzy control system, where the agent's actions are determined based on the values of position and velocity.

```
#Creating the systeam variables
posicao = ctrl.Antecedent(np.arange(-1.2,0.6,0.1), '
    Posicao')
velocidade = ctrl.Antecedent(np.arange
    (-0.07,0.07,0.001), 'Velocidade')

acao = ctrl.Consequent(np.arange(0,3,0.1), 'Acao')
```

The next step is to define the Fuzzy's universes of activity and the membership functions. In the context of Fuzzy Logic, universes and membership functions are fundamental concepts for representing and handling uncertainties. The universe of discourse refers to the complete set of values that a fuzzy variable can take. Membership functions, on the other hand, describe the degree of membership of a value to a fuzzy set within a universe of discourse [3].

There is no need to specifically discuss all the parameters used, as it is not the intention of the study, but, for the sake of knowledge, three positions were used (left, center and right), three speeds (negative, neutral and positive) and three resulting actions (forward, neutral and backward).

To define the actions, an evaluation by what we call an expert is necessary. The expert will define the consequences that he deems necessary based on the occurrence of certain factors, whether joint or specific. For example, if the position is 'center' and the speed is 'neutral', the cart should go 'forward'. The limit of rules is infinite, and everything must be done considering the final objective you want to achieve.

Finally, the 'ctrl.ControlSystem' command is used to unite all the rules created, so that they are applied as the system identifies certain combinations of situations specified by the specialist.

Originally, this Fuzzy system was able to solve the Mountain Car exercise in 170 steps. Therefore, we know that it is functional. Therefore, we can start creating the database from there. But why create a database if the solution is already there? Fuzzy solutions are very restricted for simpler problems and depend entirely on human evaluation to work. Obviously, it is the specialist who defines the final result he wants to obtain, but, with the creation of a database, an ANN can help to increasingly improve the solution, based on training and reward systems.

To create the database, we used 'loop'-based code systems, that is, we asked the system to run the Mountain Car solution several times, until we obtained the number of lines we wanted. All of this was saved in a .csv and will be used in the future in the development of the neural network.

III. The second solution: artificial neural network

In this section, we present a solution using an artificial neural network (ANN) to predict actions for the Mountain

Car environment from OpenAI Gym. The implementation involves data processing, model training, and interaction with the environment.

Before we get into the technical explanation I will go through some of the functions and parameters that we use in our code.

A. Explanation of Parameters and Functions

In this subsection, we provide an explanation of the key parameters and functions used in the implementation, focusing on their roles and the rationale behind their selection.

1) Dataset Splitting Parameters: - `test_size=0.17`: This parameter specifies that 17% of the dataset is reserved for testing, ensuring a good balance between training and validation data. A smaller test size leaves more data for training, which is important when working with limited data.

2) Neural Network Parameters:

- `input_dim=2`: Specifies the number of features (`Posicao` and `Velocidade`) in the input layer. This ensures that the network correctly aligns with the input data dimensions.
- `activation='relu'`: The ReLU activation function is used in hidden layers for its simplicity and effectiveness in avoiding vanishing gradients. It allows the network to learn complex relationships by introducing non-linearity.
- `activation='softmax'`: Applied in the output layer to generate probabilities for each action class (0, 1, 2). This ensures that the model outputs a valid probability distribution for multi-class classification.

3) Model Compilation and Training Parameters:

- `optimizer='adam'`: The Adam optimizer is selected for its efficiency and adaptability in handling sparse gradients and varying learning rates during training.
- `loss='sparse_categorical_crossentropy'`: This loss function is ideal for multi-class classification tasks where class labels are integers.
- `metrics=['accuracy']`: Accuracy is chosen as the evaluation metric to measure the proportion of correctly predicted classes during training and testing.
- `batch_size=10`: Small batch sizes allow for frequent updates to model weights, potentially improving convergence, especially with limited data.
- `epochs=100`: A relatively high number of epochs ensures sufficient learning, though early stopping may be applied to prevent overfitting.

4) Environment Interaction Parameters:

- `time.sleep(0)`: Introduced for simulation pacing but set to zero here to prioritize speed during environment interaction.
- `terminate`: The loop ends when the environment signals that the episode has concluded, ensuring proper simulation control.
- `env.step(a)`: Executes the chosen action `a` in the environment, returning the new state, reward, and termination status.

5) Rationale for Activation Function Choices:

- **ReLU:** Avoids issues like vanishing gradients, enabling deep networks to train more effectively. It introduces sparsity, making the network computationally efficient.
- **Softmax:** Ensures the output values are interpretable as probabilities for each action, aiding in decision-making for classification tasks.

These parameters and function choices contribute to the robustness and efficiency of the ANN-based solution and environment interaction logic.

B. Code Explanation

1) Setup and Data Loading: The necessary libraries are installed, and the dataset is loaded from Google Drive. The features `Position` and `Velocity` are used as inputs, and the target is the `Action`. The data is split into training and testing sets.

```
# Install dependencies
!pip install pyvirtualdisplay
!apt-get install -y xvfb

# Mount Google Drive and load data
drive.mount('/content/gdrive')
dados = pd.read_csv('/content/gdrive/MyDrive/Colab␣
    Notebooks/dftotal.csv')

# Prepare predictors and target
previsores = dados[['Posicao', 'Velocidade']]
classe = dados['Acao']

# Split data
previsores_treinamento, previsores_teste,
    classe_treinamento, classe_teste =
    train_test_split(previsores, classe, test_size
    =0.17)
```

2) Building and Training the Neural Network: A neural network with two hidden layers (64 and 32 units) and a softmax output layer is constructed. The model is compiled using the Adam optimizer and trained with the training dataset.

```
# Create the ANN model
classificador = Sequential()

# Hidden layers
classificador.add(Dense(64, input_dim=2, activation=
    'relu'))
classificador.add(Dense(32, activation='relu'))

# Output layer
classificador.add(Dense(3, activation='softmax'))

# Compile and train
classificador.compile(optimizer='adam', loss='
    sparse_categorical_crossentropy', metrics=['
    accuracy'])
classificador.fit(previsores_treinamento,
    classe_treinamento, batch_size=10, epochs=100)
```

3) Interacting with the Environment: The trained neural network is used to control the agent in the `MountainCar-v0` environment. The agent observes the state, predicts the action, and interacts with the environment until the episode ends.

```
# Create environment
env = RecordVideo(gym.make('MountainCar-v0'), './
    video')
obs = env.reset()

# Simulate with the ANN
total_reward = 0
while True:
    entrada = np.array([obs]).reshape(1, -1)
    acao = np.argmax(classificador.predict(entrada,
        verbose=0))
    obs, reward, terminate, _ = env.step(acao)
    total_reward += reward
    if terminate:
        break
```

4) Interacting with the Environment Using Rules-Based Logic: A rules-based controller is also implemented to interact with the `MountainCar-v0` environment. The system evaluates the car's position and velocity to decide the next action.

```
# Initialize environment
env = wrap_env(gym.make('MountainCar-v0'))
obs = env.reset()

disco.input['Posicao'] = obs[0]
disco.input['Velocidade'] = obs[1]

passo = 0
while True:
    time.sleep(0)
    passo += 1

    env.render()  # Render environment

    disco.compute()
    a = disco.output['Acao']
    if a < 1:
        a = 0
    elif a > 2:
        a = 2
    else:
        a = 1

    obs, reward, terminate, _ = env.step(a)

    disco.input['Posicao'] = obs[0]
    disco.input['Velocidade'] = obs[1]

    print(f"Passo␣:␣{passo}")
    print(f"Posicao␣do␣passo␣:␣{obs[0]}")
    print(f"Velocidade␣do␣passo␣:␣{obs[1]}")
    print(f"Acao␣do␣passo␣:␣{a}\n")

    if terminate:
        break

env.close()
show_video()
```

This section demonstrates how both the ANN and rules-based approaches are utilized to predict actions and control the agent in the environment.

C. Results

After conducting extensive tests, iterative executions, and refinements, our neural network-based approach yielded results comparable to or exceeding the performance of the Fuzzy system. While the Fuzzy logic system achieved an average of 170 steps in solving the Cart Pole problem, our neural network consistently performed within the range of 170 to 180 steps. These outcomes underscore the potential of neural network architectures in control tasks, provided they are well-designed and effectively trained.

During the project's development, several technical challenges emerged:

Hardware limitations: The available computational resources constrained the generation of a sufficiently large and diverse dataset, adversely impacting the system's ability to generalize. Low-dimensional input data: The Fuzzy system relied on only two input parameters, offering limited information for effective decision-making and subsequent performance comparisons. Overfitting in neural training: The rapid convergence during training caused the model to overfit on the training data, reducing its generalization capability and stability in unseen scenarios. To overcome these challenges, we propose the following technical improvements:

Enhanced computational resources: Using hardware with greater memory and processing power would allow the creation of a larger dataset, ideally containing between 200,000 and 300,000 entries. A larger dataset would improve the model's capacity to learn diverse patterns and avoid over-reliance on a narrow input range. Data augmentation and preprocessing: Introducing techniques like noise injection, data normalization, and synthetic data generation could further enrich the training dataset, ensuring better representation of edge cases and reducing overfitting tendencies. Optimized neural architecture: Experimentation with advanced neural network topologies, such as deeper networks with dropout layers, regularization techniques (L2 regularization), and batch normalization, would help address overfitting and enhance model robustness. Hybrid approaches: Combining neural networks with Fuzzy systems in a hybrid structure might leverage the interpretability of Fuzzy logic while benefiting from the adaptability of neural networks. These proposed measures are expected to enhance the robustness, scalability, and generalization capabilities of the system, allowing it to handle more complex variations of control tasks like Cart Pole and beyond.

References

[1] Hammad, A., Hossain, M., "Scikit-Fuzzy: A Library for Fuzzy Logic in Python," arXiv preprint arXiv:1911.10374, 2019.

[2] Brockman, G., et al., "OpenAI Gym.," arXiv:1606.01540., 2016.

[3] Zadeh, L. A., "Fuzzy sets. Information and Control," 8(3), p. 338–353., 1965.