# Design of a Fuzzy Neural Network for Optimizing Energy Consumption and Reducing $CO_2$ Emissions

## 1 Problem

In this work, we present a novel fuzzy neural network (FNN) architecture designed to minimize energy consumption and reduce $CO_2$ emissions. This hybrid approach combines the flexibility of fuzzy logic with the learning capabilities of neural networks. A clear and detailed system design is proposed, including rule construction, membership functions, and neural layers. The method is evaluated on a simple application related to smart energy control in a building, showing its potential in achieving sustainability goals.

Growing concerns about global warming and energy inefficiency necessitate intelligent solutions that can adapt to uncertain, nonlinear environments. Fuzzy neural networks offer a promising path by merging human-like reasoning and data-driven learning. This paper proposes a self-adaptive FNN to dynamically adjust energy usage and minimize $CO_2$ output in real-time applications.

**Fuzzy Neural Network Design:**

Architecture Overview:

1. **Input Layer:** Accepts real-time data (e.g., temperature, occupancy, device load).
2. **Fuzzification Layer:** Converts crisp inputs into fuzzy values using membership functions.
3. **Rule Layer:** Applies fuzzy rules of the form: *IF temperature is high AND occupancy is low THEN reduce AC power.*
4. **Neural Adaptation Layer:** Learns optimal rule weights and adjusts outputs through gradient descent.
5. **Defuzzification Layer:** Converts fuzzy outputs back to crisp control signals (e.g., HVAC adjustment level).

**Mathematical Representation** Let $x_1, x_2, ..., x_n$ be the input features. Membership functions $\mu_i(x)$ map these inputs into fuzzy sets. Rules are defined as:

$$R_j : \text{IF } x_1 \text{ is } A_{1j} \text{ AND } x_2 \text{ is } A_{2j} \text{ THEN } y_j = w_j$$

The final output is:

$$y = \frac{\sum_j w_j \cdot \mu_{A_{1j}}(x_1) \cdot \mu_{A_{2j}}(x_2)}{\sum_j \mu_{A_{1j}}(x_1) \cdot \mu_{A_{2j}}(x_2)}$$

**Application: Smart Building Energy Control**

In a simple application, the system receives temperature, time of day, and occupancy data. The FNN adjusts lighting and HVAC systems to ensure minimal energy usage while maintaining comfort. It learns patterns like:

1. Reducing cooling when the room is unoccupied.
2. Dimming lights when natural sunlight is sufficient.

## 2 Solution

This architecture presents a Fuzzy Neural Network (FNN) designed to optimize energy efficiency and reduce carbon emissions in FPGA-based machine learning systems. The FNN dynamically adjusts ML algorithm parameters based on current environmental and operational conditions using fuzzy inference to manage uncertainty and imprecision.

## 1. Inputs and Fuzzification Layer

The **Fuzzification Layer** converts crisp real-world input values into fuzzy linguistic variables by assigning degrees of membership to fuzzy sets. These inputs fall into three categories:

### 1.1 Environmental Variables

#### 1.1.1 Temperature

1. Low: Very cold temperatures
2. Medium: Moderate temperatures
3. High: Very warm or hot temperatures

#### 1.1.2 Humidity

1. Low: Dry air
2. Medium: Comfortable humidity
3. High: Humid or wet conditions

#### 1.1.3 Time of Day

1. Morning: Early hours
2. Afternoon: Midday to early evening
3. Evening: Late evening and night

### 1.2 Operational Variables

#### 1.2.1 Energy Consumption

1. Low: Minimal energy usage
2. Medium: Moderate usage
3. High: High energy demand

#### 1.2.2 Training Progress

1. Low: Initial stages
2. Medium: Intermediate progress
3. High: Advanced training

## 1.3 Historical Data

### 1.3.1 Past Performance Metrics

1. Poor: Suboptimal outcomes
2. Moderate: Acceptable performance
3. Good: Highly efficient results

### 1.3.2 Past Environmental Conditions

1. Unfavorable: Previously difficult conditions
2. Moderate: Average past states
3. Favorable: Ideal past conditions

## 1.4 ML Algorithm Metrics

### 1.4.1 Speed of ML Algorithm

1. Slow: Long processing time
2. Moderate: Balanced speed
3. Fast: High-speed execution

### 1.4.2 Energy Consumption of ML Algorithm

1. Low: Efficient energy profile
2. Moderate: Average energy use
3. High: Energy intensive

# 2. Rule Base

The fuzzy rule base defines the logical relationships between input conditions and actions to reduce energy use and $CO_2$ emissions. Sample rules include:

## 2.1 Energy Consumption Rules

1. IF (Temperature is High) AND (Energy Consumption is High) THEN **Reduce Energy Consumption**
2. IF (Humidity is High) AND (Energy Consumption is High) THEN **Reduce Energy Consumption**
3. IF (Time of Day is Evening) AND (Energy Consumption is High) THEN **Reduce Energy Consumption**

## 2.2 $CO_2$ Emissions Rules

1. IF (Training Progress is High) AND (Past Performance is Good) THEN **$CO_2$ Emissions is Low**
2. IF (Speed of ML Algorithm is Fast) AND (Energy Consumption of ML Algorithm is Low) THEN **$CO_2$ Emissions is Low**

These rules guide the FNN's decision-making to prioritize environmental sustainability while maintaining performance.

## 3. Membership Functions

Membership functions define how each input value maps to fuzzy sets.

Each variable's membership function can be triangular, trapezoidal, or Gaussian, depending on the complexity and required interpretability.

## 4. Environmental Variables

### 4.1 Temperature:

1. Low Temperature (LT):

$$\mu_{LT}(x) = \begin{cases} 1 & \text{if } x \leq 0 \\ \frac{b-x}{b-a} & \text{if } 0 \leq x \leq b \\ 0 & \text{if } x \geq b \end{cases}$$

   where $a = 0$ and $b = 10$.

2. Medium Temperature (MT):

$$\mu_{MT}(x) = \begin{cases} 0 & \text{if } x \leq a \text{ or } x \geq c \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ \frac{c-x}{c-b} & \text{if } b \leq x \leq c \end{cases}$$

   where $a = 15$, $b = 25$, and $c = 35$.

3. High Temperature (HT):

$$\mu_{HT}(x) = \begin{cases} 0 & \text{if } x \leq b \\ \frac{x-b}{c-b} & \text{if } b \leq x \leq c \\ 1 & \text{if } x \geq c \end{cases}$$

   where $b = 30$ and $c = 50$.

### 4.2 Humidity:

1. Low Humidity (LH):

$$\mu_{LH}(y) = \begin{cases} 1 & \text{if } y \leq a \\ \frac{b-y}{b-a} & \text{if } a \leq y \leq b \\ 0 & \text{if } y \geq b \end{cases}$$

   where $a = 0$ and $b = 20$.

2. Medium Humidity (MH):

$$\mu_{MH}(y) = \begin{cases} 0 & \text{if } y \leq a \text{ or } y \geq c \\ \frac{y-a}{b-a} & \text{if } a \leq y \leq b \\ \frac{c-y}{c-b} & \text{if } b \leq y \leq c \end{cases}$$

   where $a = 30$, $b = 50$, and $c = 70$.

3. High Humidity (HH):

$$\mu_{HH}(y) = \begin{cases} 0 & \text{if } y \leq b \\ \frac{y-b}{c-b} & \text{if } b \leq y \leq c \\ 1 & \text{if } y \geq c \end{cases}$$

   where $b = 60$ and $c = 80$.

## 4.3 Time:

1. Morning Time (MT):
$$\mu_{MT}(t) = \begin{cases} 1 & \text{if } t \leq a \\ \frac{b-t}{b-a} & \text{if } a \leq t \leq b \\ 0 & \text{if } t \geq b \end{cases}$$
where $a = 0$ and $b = 4$.

2. Afternoon Time (AT):
$$\mu_{AT}(t) = \begin{cases} 0 & \text{if } t \leq a \text{ or } t \geq c \\ \frac{t-a}{b-a} & \text{if } a \leq t \leq b \\ \frac{c-t}{c-b} & \text{if } b \leq t \leq c \end{cases}$$
where $a = 6$, $b = 12$, and $c = 18$.

3. Evening Time (ET):
$$\mu_{ET}(t) = \begin{cases} 0 & \text{if } t \leq b \\ \frac{t-b}{c-b} & \text{if } b \leq t \leq c \\ 1 & \text{if } t \geq c \end{cases}$$
where $b = 16$ and $c = 20$.

## 4.4 Energy Consumption:

1. Low Energy Consumption (LEC):
$$\mu_{LEC}(x) = \begin{cases} 1 & \text{if } x \leq 0 \\ \frac{50-x}{50-0} & \text{if } 0 \leq x \leq 50 \\ 0 & \text{if } x \geq 50 \end{cases}$$

2. Medium Energy Consumption (MEC):
$$\mu_{MEC}(x) = \begin{cases} 0 & \text{if } x \leq 50 \text{ or } x \geq 150 \\ \frac{x-50}{100-50} & \text{if } 50 \leq x \leq 100 \\ \frac{150-x}{150-100} & \text{if } 100 \leq x \leq 150 \end{cases}$$

3. High Energy Consumption (HEC):
$$\mu_{HEC}(x) = \begin{cases} 0 & \text{if } x \leq 100 \\ \frac{x-100}{150-100} & \text{if } 100 \leq x \leq 150 \\ 1 & \text{if } x \geq 150 \end{cases}$$

## 4.5 Training Progress:

1. Low Training Progress (LTP):
$$\mu_{LTP}(t) = \begin{cases} 1 & \text{if } t \leq 0 \\ \frac{30-t}{30-0} & \text{if } 0 \leq t \leq 30 \\ 0 & \text{if } t \geq 30 \end{cases}$$

2. Medium Training Progress (MTP):
$$\mu_{MTP}(t) = \begin{cases} 0 & \text{if } t \leq 20 \text{ or } t \geq 80 \\ \frac{t-20}{50-20} & \text{if } 20 \leq t \leq 50 \\ \frac{80-t}{80-50} & \text{if } 50 \leq t \leq 80 \end{cases}$$

3. High Training Progress (HTP):

$$\mu_{HTP}(t) = \begin{cases} 0 & \text{if } t \leq 70 \\ \frac{t-70}{100-70} & \text{if } 70 \leq t \leq 100 \\ 1 & \text{if } t \geq 100 \end{cases}$$

## 4.6 Performance:

1. Poor Performance (PP):

$$\mu_{PP}(m) = \begin{cases} 1 & \text{if } m \leq 0 \\ \frac{30-m}{30-0} & \text{if } 0 \leq m \leq 30 \\ 0 & \text{if } m \geq 30 \end{cases}$$

2. Moderate Performance (MP):

$$\mu_{MP}(m) = \begin{cases} 0 & \text{if } m \leq 20 \text{ or } m \geq 80 \\ \frac{m-20}{50-20} & \text{if } 20 \leq m \leq 50 \\ \frac{80-m}{80-50} & \text{if } 50 \leq m \leq 80 \end{cases}$$

3. Good Performance (GP):

$$\mu_{GP}(m) = \begin{cases} 0 & \text{if } m \leq 70 \\ \frac{m-70}{100-70} & \text{if } 70 \leq m \leq 100 \\ 1 & \text{if } m \geq 100 \end{cases}$$

## 4.7 Conditions:

1. Unfavorable Conditions (UC):

$$\mu_{UC}(e) = \begin{cases} 1 & \text{if } e \leq 0 \\ \frac{30-e}{30-0} & \text{if } 0 \leq e \leq 30 \\ 0 & \text{if } e \geq 30 \end{cases}$$

2. Moderate Conditions (MC):

$$\mu_{MC}(e) = \begin{cases} 0 & \text{if } e \leq 20 \text{ or } e \geq 80 \\ \frac{e-20}{50-20} & \text{if } 20 \leq e \leq 50 \\ \frac{80-e}{80-50} & \text{if } 50 \leq e \leq 80 \end{cases}$$

3. Favorable Conditions (FC):

$$\mu_{FC}(e) = \begin{cases} 0 & \text{if } e \leq 70 \\ \frac{e-70}{100-70} & \text{if } 70 \leq e \leq 100 \\ 1 & \text{if } e \geq 100 \end{cases}$$

## 4.8 Speed:

1. Slow Speed (SS):

$$\mu_{SS}(s) = \begin{cases} 1 & \text{if } s \leq 0 \\ \frac{30-s}{30-0} & \text{if } 0 \leq s \leq 30 \\ 0 & \text{if } s \geq 30 \end{cases}$$

2. Moderate Speed (MS):

$$\mu_{MS}(s) = \begin{cases} 0 & \text{if } s \le 20 \text{ or } s \ge 80 \\ \frac{s-20}{50-20} & \text{if } 20 \le s \le 50 \\ \frac{80-s}{80-50} & \text{if } 50 \le s \le 80 \end{cases}$$

3. Fast Speed (FS):

$$\mu_{FS}(s) = \begin{cases} 0 & \text{if } s \le 70 \\ \frac{s-70}{100-70} & \text{if } 70 \le s \le 100 \\ 1 & \text{if } s \ge 100 \end{cases}$$

### 4.9 Energy Consumption:

1. Low Energy Consumption (LEC):

$$\mu_{LEC}(e) = \begin{cases} 1 & \text{if } e \le 0 \\ \frac{50-e}{50-0} & \text{if } 0 \le e \le 50 \\ 0 & \text{if } e \ge 50 \end{cases}$$

2. Moderate Energy Consumption (MEC):

$$\mu_{MEC}(e) = \begin{cases} 0 & \text{if } e \le 30 \text{ or } e \ge 100 \\ \frac{e-30}{70-30} & \text{if } 30 \le e \le 70 \\ \frac{100-e}{100-70} & \text{if } 70 \le e \le 100 \end{cases}$$

3. High Energy Consumption (HEC):

$$\mu_{HEC}(e) = \begin{cases} 0 & \text{if } e \le 70 \\ \frac{e-70}{100-70} & \text{if } 70 \le e \le 100 \\ 1 & \text{if } e \ge 100 \end{cases}$$

## Rule Processing Layer: (Neural Network Hidden Layer)

The rule processing layer interprets fuzzy inputs and applies the defined rules to generate output fuzzy sets. Each neuron in this layer corresponds to a specific fuzzy rule, and its output represents the degree to which the rule is satisfied. The layer integrates information from environmental variables, operational variables, historical data, and ML algorithm metrics.

### Energy Consumption Rule 1

1. IF (Temperature is High) AND (Energy Consumption is High) THEN (Reduce Energy Consumption) Apply the min operation to find the minimum membership degree for each rule. Combine the rules using the max operation.

```
import numpy as np

# Membership functions for temperature
def low_temperature(x, a=0, b=10, c=20):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [1, lambda x: (b - x) / (b - a), lambda x: (c - x) / (c - b), 0])

def medium_temperature(x, a=15, b=25, c=35):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - a) / (b - a), lambda x: (c - x) / (c - b), 0])
```

```
def high_temperature(x, a=30, b=40, c=50):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - b) / (c - b), 1, 0])

# Membership functions for energy consumption
def low_energy_consumption(x, a=0, b=50):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), x >= b],
        [1, lambda x: (b - x) / (b - a), 0])

def medium_energy_consumption(x, a=50, b=100, c=150):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - a) / (b - a), lambda x: (c - x) / (c - b), 0])

def high_energy_consumption(x, b=100, c=150):
    return np.piecewise(x,
        [x <= b, (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - b) / (c - b), 1])

# Membership functions for humidity
def low_humidity(y, a=0, b=20):
    return np.piecewise(y,
        [y <= a, (a <= y) & (y <= b), y >= b],
        [1, lambda y: (b - y) / (b - a), 0])

def medium_humidity(y, a=30, b=50, c=70):
    return np.piecewise(y,
        [y <= a, (a <= y) & (y <= b), (b <= y) & (y <= c), y >= c],
        [0, lambda y: (y - a) / (b - a), lambda y: (c - y) / (c - b), 0])

def high_humidity(y, b=60, c=80):
    return np.piecewise(y,
        [y <= b, (b <= y) & (y <= c), y >= c],
        [0, lambda y: (y - b) / (c - b), 1])

# Generate input ranges
x_values = np.linspace(0, 150, 1000)
y_values = np.linspace(0, 100, 1000)

# Fuzzy rule: IF (Temperature is High) AND (Energy Consumption is High)
rule1 = np.minimum(high_temperature(x_values), high_energy_consumption(x_values))
output = np.maximum(rule1, 0)
```

Fuzzy output of 0 typically means that, based on the fuzzy rules and the given input values, there is no support or very weak support for the corresponding conclusion or linguistic term.

### Energy Consumption Rule 2

1. IF (Humidity is High) AND (Energy Consumption is High) THEN (Reduce Energy Consumption)

   Apply the min operation to find the minimum membership degree for each rule. Combine the rules using the max operation if there are multiple.

```
import numpy as np

# Membership functions for humidity
def low_humidity(y, a=0, b=20):
    return np.piecewise(y,
        [y <= a, (a <= y) & (y <= b), y >= b],
        [1, lambda y: (b - y) / (b - a), 0])

def medium_humidity(y, a=30, b=50, c=70):
    return np.piecewise(y,
        [y <= a, (a <= y) & (y <= b), (b <= y) & (y <= c), y >= c],
        [0, lambda y: (y - a) / (b - a), lambda y: (c - y) / (c - b), 0])

def high_humidity(y, b=60, c=80):
    return np.piecewise(y,
        [y <= b, (b <= y) & (y <= c), y >= c],
        [0, lambda y: (y - b) / (c - b), 1])

# Membership functions for energy consumption
def low_energy_consumption(x, a=0, b=50):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), x >= b],
        [1, lambda x: (b - x) / (b - a), 0])

def medium_energy_consumption(x, a=50, b=100, c=150):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - a) / (b - a), lambda x: (c - x) / (c - b), 0])

def high_energy_consumption(x, b=100, c=150):
    return np.piecewise(x,
        [x <= b, (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - b) / (c - b), 1])

# Generate input ranges
x_values = np.linspace(0, 150, 1000)
y_values = np.linspace(0, 100, 1000)

# Compute membership degrees
humidity_high = high_humidity(y_values)
energy_high = high_energy_consumption(x_values)

# Apply fuzzy rule: IF (Humidity is High) AND (Energy Consumption is High)
combined_membership = np.minimum(humidity_high, energy_high)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(y_values, humidity_high, label='High Humidity', linestyle='--')
plt.plot(x_values, energy_high, label='High Energy Consumption', linestyle='--')
plt.plot(y_values, combined_membership, label='Combined Rule (Min)', color='red')

# Show last three membership values
print(combined_membership[-3:])
```

**Output:**

```
[0.99399399, 0.996997, 1]
```

In the highlighted region (between 60 and 80 for humidity and between 100 and 150 for energy consumption), both conditions—high humidity and high energy consumption—coincide, resulting in a non-zero membership value for the combined rule.

The last three output values of the membership degree are:

$$[0.99399399, 0.996997, 1]$$

The higher the combined membership value, the stronger the support for the conclusion **"Reduce Energy Consumption"** based on the fuzzy rules for the specific input values in this region.

### Energy Consumption Rule 3

1. IF (Time of Day is Evening) AND (Energy Consumption is High) THEN (Reduce Energy Consumption)

   Apply the min operation to find the minimum membership degree for each rule. Combine the rules using the max operation if multiple rules apply.

```python
import numpy as np

# Membership functions for time of day
def morning_time(t, a=0, b=4):
    return np.piecewise(t,
        [t <= a, (a <= t) & (t <= b), t >= b],
        [1, lambda t: (b - t) / (b - a), 0])

def afternoon_time(t, a=6, b=12, c=18):
    return np.piecewise(t,
        [t <= a, (a <= t) & (t <= b), (b <= t) & (t <= c), t >= c],
        [0, lambda t: (t - a) / (b - a), lambda t: (c - t) / (c - b), 0])

def evening_time(t, b=16, c=20):
    return np.piecewise(t,
        [t <= b, (b <= t) & (t <= c), t >= c],
        [0, lambda t: (t - b) / (c - b), 1])

# Membership functions for energy consumption
def low_energy_consumption(x, a=0, b=50):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), x >= b],
        [1, lambda x: (b - x) / (b - a), 0])

def medium_energy_consumption(x, a=50, b=100, c=150):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - a) / (b - a), lambda x: (c - x) / (c - b), 0])

def high_energy_consumption(x, b=100, c=150):
    return np.piecewise(x,
        [x <= b, (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - b) / (c - b), 1])

# Generate input ranges
x_values = np.linspace(0, 20, 1000)       # Time of day from 0 to 20 hours
y_values = np.linspace(0, 150, 1000)      # Energy consumption from 0 to 150

# Compute membership degrees
evening_time_values = evening_time(x_values)
energy_high = high_energy_consumption(y_values)

# Apply fuzzy rule: IF (Time of Day is Evening) AND (Energy Consumption is High)
combined_membership = np.minimum(evening_time_values, energy_high)

print(combined_membership[-3:])

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(x_values, evening_time_values, label='Evening Time', linestyle='--')
plt.plot(y_values, energy_high, label='High Energy Consumption', linestyle='--')
plt.plot(x_values, combined_membership, label='Combined Rule (Min)', color='red')
```

**Output:**

```
[0.98998999, 0.99499499, 1]
```

In the highlighted region (around 16 to 20 hours for time and high energy consumption levels), both conditions—Evening time and High Energy Consumption—coincide, resulting in a non-zero membership value for the combined rule.

The last three membership degree values are:

$$[0.98998999,\ 0.99499499,\ 1]$$

The higher the combined membership value, the stronger the support for the conclusion **"Reduce Energy Consumption"** based on the fuzzy rules for the specific input values in this region.

## CO2 Emissions Rule 1

1. IF (Training Progress is High) AND (Past Performance Metrics is Good Performance) THEN CO2 Emissions is Low

   Apply the min operation to find the minimum membership degree for each rule. Combine the rules using the max operation if multiple rules apply.

```python
import numpy as np
import matplotlib.pyplot as plt

# Membership functions for Training Progress
def low_training_progress(t, a=0, b=30):
    return np.piecewise(t,
        [t <= a, (a <= t) & (t <= b), t >= b],
        [1, lambda t: (b - t) / (b - a), 0])

def medium_training_progress(t, a=20, b=50, c=80):
    return np.piecewise(t,
        [t <= a, (a <= t) & (t <= b), (b <= t) & (t <= c), t >= c],
        [0, lambda t: (t - a) / (b - a), lambda t: (c - t) / (c - b), 0])

def high_training_progress(t, b=70, c=100):
    return np.piecewise(t,
        [t <= b, (b <= t) & (t <= c), t >= c],
        [0, lambda t: (t - b) / (c - b), 1])

# Membership functions for Past Performance
def poor_performance(m, a=0, b=30):
    return np.piecewise(m,
        [m <= a, (a <= m) & (m <= b), m >= b],
        [1, lambda m: (b - m) / (b - a), 0])

def moderate_performance(m, a=20, b=50, c=80):
    return np.piecewise(m,
        [m <= a, (a <= m) & (m <= b), (b <= m) & (m <= c), m >= c],
        [0, lambda m: (m - a) / (b - a), lambda m: (c - m) / (c - b), 0])

def good_performance(m, b=70, c=100):
    return np.piecewise(m,
        [m <= b, (b <= m) & (m <= c), m >= c],
        [0, lambda m: (m - b) / (c - b), 1])

# Generate input ranges
x_values = np.linspace(0, 150, 1000)  # Past performance metrics
y_values = np.linspace(0, 100, 1000)  # Training progress

# Compute membership degrees
high_training = high_training_progress(y_values)
past_performance = good_performance(x_values)

# Apply fuzzy rule: IF (Training Progress is High) AND (Past Performance is Good)
combined_membership = np.minimum(high_training, past_performance)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(y_values, high_training, label='High Training Progress', linestyle='--')
plt.plot(x_values, past_performance, label='Good Past Performance', linestyle='--')
plt.plot(y_values, combined_membership, label='Combined Rule (Min)', color='red')

# Highlight peak points
plt.scatter([70, 100], [1, 1], color='red')
plt.scatter([70, 100], [1, 1], color='red')
plt.title('Combined Fuzzy Logic Rule')
plt.xlabel('Training Progress / Past Performance')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()

# Print last three membership degrees
print(combined_membership[-4:])
```

**Output:**

```
[0.98998999 0.99332666 0.99666333 1.        ]
```

The higher the combined membership value, the stronger the support for the conclusion **"CO2 Emissions is Low"** based on the fuzzy rules for the specific input values in the highlighted region.

### CO2 Emissions Rule 2

1. IF (Speed of ML Algorithm is Fast) AND (Energy Consumption of ML Algorithm is Low) THEN CO2 Emissions is Low

   Apply the min operation to find the minimum membership degree for each rule. Combine the rules using the max operation if multiple rules apply.

```
import numpy as np
import matplotlib.pyplot as plt

# Membership functions for Speed
def slow_speed(s, a=0, b=30):
    return np.piecewise(s,
        [s <= a, (a <= s) & (s <= b), s >= b],
        [1, lambda s: (b - s) / (b - a), 0])

def moderate_speed(s, a=20, b=50, c=80):
    return np.piecewise(s,
        [s <= a, (a <= s) & (s <= b), (b <= s) & (s <= c), s >= c],
        [0, lambda s: (s - a) / (b - a), lambda s: (c - s) / (c - b), 0])

def fast_speed(s, b=70, c=100):
    return np.piecewise(s,
        [s <= b, (b <= s) & (s <= c), s >= c],
        [0, lambda s: (s - b) / (c - b), 1])

# Membership functions for Energy Consumption
def low_energy_consumption(x, a=0, b=50):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), x >= b],
        [1, lambda x: (b - x) / (b - a), 0])

def medium_energy_consumption(x, a=50, b=100, c=150):
    return np.piecewise(x,
        [x <= a, (a <= x) & (x <= b), (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - a) / (b - a), lambda x: (c - x) / (c - b), 0])

def high_energy_consumption(x, b=100, c=150):
    return np.piecewise(x,
        [x <= b, (b <= x) & (x <= c), x >= c],
        [0, lambda x: (x - b) / (c - b), 1])

# Generate input ranges
x_values = np.linspace(0, 150, 1000)  # Energy consumption
y_values = np.linspace(0, 100, 1000)  # Speed

# Compute membership degrees
fast_speed_values = fast_speed(y_values)
```

```
low_energy_consumption_values = low_energy_consumption(x_values)

# Apply fuzzy rule: IF (Speed is Fast) AND (Energy Consumption is Low)
combined_membership = np.minimum(fast_speed_values, low_energy_consumption_values)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(y_values, fast_speed_values, label='Fast Speed', linestyle='--')
plt.plot(x_values, low_energy_consumption_values, label='Low Energy Consumption',
    linestyle='--')
plt.plot(x_values, combined_membership, label='Combined Rule (Min)', color='red')

# Highlight peak points
plt.scatter([30, 70], [1, 1], color='red')
plt.scatter([0, 50], [1, 1], color='red')

plt.title('Combined Fuzzy Logic Rule')
plt.xlabel('Speed / Energy Consumption Level')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
```

**Note:** A fuzzy output of 0 typically means that, based on the fuzzy rules and the given input values, there is no or very weak support for the corresponding conclusion or linguistic term.

### Aggregation Layer

Neurons aggregate the outputs of the Rule Layer. This can involve summing the membership values from different rules to determine the overall support for a conclusion.

**1. Energy Consumption** Sum the contributions from each rule for the conclusion **"Reduce Energy Consumption"** based on the given membership values:

- Rule 1: Membership value = 0 (weak support).
- Rule 2: Summed membership values = 167.0
- Rule 3: Summed membership values = 100.40040040040043

The aggregated support for the conclusion **"Reduce Energy Consumption"** is calculated by summing these values. Rule 2 has the highest aggregated support with a sum of 167.0.

**2. CO2 Emissions** Sum the contributions from each rule for the conclusion **"Reduce CO2 Emissions"**:

- Rule 1: Summed membership values = 150.3503503503504
- Rule 2: Membership value = 0 (weak support).

The aggregated support for **"Reduce CO2 Emissions"** is highest from Rule 1 with a sum of 150.3503503503504.

### Output Layer

Select the best membership values from the aggregation layer:

$$\text{best\_membership\_energy} = 167.0$$

$$\text{best\_membership\_co2} = 150.3503503503504$$

Calculate the mean of these values:

$$\text{output\_layer\_mean} = \frac{\text{best\_membership\_energy} + \text{best\_membership\_co2}}{2} = 158.67517517517518$$

Print the output:

```python
import numpy as np

# Example combined_membership array from fuzzy logic outputs (replace with actual values)
combined_membership = np.array([/* fuzzy membership values from rules */])

# Sum the combined membership values
sum_combined_membership = np.sum(combined_membership)
print("Sum of combined membership values:", sum_combined_membership)

# Given best membership values from aggregation
best_membership_energy = 167.0
best_membership_co2 = 150.3503503503504

# Calculate the mean output of the output layer
output_layer_mean = (best_membership_energy + best_membership_co2) / 2
print("Output Layer Mean:", output_layer_mean)

# Activation function (sigmoid) applied to output layer mean to get probability
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

output_probability = sigmoid(output_layer_mean)
print("Output Layer Probability (after sigmoid):", output_probability)
```

**Output:**

```
Sum of combined membership values: <value depending on input>
Output Layer Mean: 158.67517517517518
Output Layer Probability (after sigmoid): 1.0
```

## Notes

The aggregation layer sums the support from all relevant fuzzy rules to provide a consolidated measure of confidence for each conclusion. The output layer calculates a mean of the best membership values and applies an activation function such as the sigmoid to convert this measure into a normalized probability between 0 and 1. This allows the fuzzy inference system to output a probabilistic interpretation of conclusions like "Reduce Energy Consumption" or "Reduce CO2 Emissions" based on the input data and fuzzy logic rules.

*In this example, the high output probability (close to 1) indicates strong overall support from the fuzzy logic system for the conclusions based on the given inputs and rule evaluations.*