Numpy implementation

Temperature and Humidity

Dr. Abraham Aldaco

June 6, 20205

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
```

In [3]:
```
# Load dataset
df = pd.read_csv("data_Temperature_Humidity.csv")
```

In [4]:
```
df.head
```

Out[4]:
```
<bound method NDFrame.head of    Temperature  Humidity Class
0     3.393533  2.331273     A
1     3.110073  1.781540     A
2     1.343809  3.368361     A
3     3.582294  4.679179     A
4     2.280362  2.866990     A
5     7.423437  4.696523     B
6     5.745052  3.533990     B
7     9.172169  2.511101     B
8     7.792783  3.424089     B
9     7.939821  0.791637     B>
```

In [5]:
```
# Extract inputs and target
X = df[['Temperature', 'Humidity']].values
y = df['Class']
```

In [6]:
```
# Encode class (e.g., 'A' -> 0, other -> 1)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y).reshape(-1, 1)
```

In [8]:
```
y_encoded
```

Out[8]:
```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1]])
```

In [9]:
```
# Normalize input features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

In [10]:
```python
X_scaled
```

Out[10]:
```
array([[0.26183319, 0.39428457],
       [0.22562385, 0.25350356],
       [0.        , 0.65987175],
       [0.28594562, 0.99555844],
       [0.11963599, 0.53147601],
       [0.77661583, 1.        ],
       [0.56221779, 0.70228755],
       [1.        , 0.44033653],
       [0.8237964 , 0.6741431 ],
       [0.84257905, 0.        ]])
```

In [11]:
```python
# Define sigmoid and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
```

In [12]:
```python
# Initialize network parameters
np.random.seed(42)
input_dim = 2      # Temperature, Humidity
hidden_dim = 2     # Layer L1: 2 neurons
output_dim = 1     # Final output: y_pred
learning_rate = 0.1
epochs = 1000
```

In [13]:
```python
# Weight and bias initialization
W1 = np.random.randn(input_dim, hidden_dim)
b1 = np.zeros((1, hidden_dim))
W2 = np.random.randn(hidden_dim, output_dim)
b2 = np.zeros((1, output_dim))
```

In [14]:
```python
print(W1)
print(b1)
print(W2)
print(b2)
```

```
[[ 0.49671415 -0.1382643 ]
 [ 0.64768854  1.52302986]]
[[0. 0.]]
[[-0.23415337]
 [-0.23413696]]
[[0.]]
```

In [19]:
```python
# Training Loop
for epoch in range(epochs):
    # Forward pass
    z1 = np.dot(X_scaled, W1) + b1
    a1 = sigmoid(z1)                    # Output of L1
    z2 = np.dot(a1, W2) + b2
    y_pred = sigmoid(z2)               # Final prediction

    # Compute loss (mean squared error)
    loss = np.mean((y_encoded - y_pred) ** 2)

    # Backpropagation
```

```python
    error_output = y_encoded - y_pred
    d_output = error_output * sigmoid_derivative(y_pred)

    error_hidden = d_output.dot(W2.T)
    d_hidden = error_hidden * sigmoid_derivative(a1)

    # Gradient descent updates
    W2 += a1.T.dot(d_output) * learning_rate
    b2 += np.sum(d_output, axis=0, keepdims=True) * learning_rate
    W1 += X_scaled.T.dot(d_hidden) * learning_rate
    b1 += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate
```

In [21]:

```python
# Final prediction for first 5 samples
z1 = np.dot(X_scaled, W1) + b1
a1 = sigmoid(z1)
z2 = np.dot(a1, W2) + b2
final_output = sigmoid(z2)

print("Predictions for first 5 samples:\n", final_output[:8])
```

```
Predictions for first 5 samples:
 [[0.06078301]
 [0.03831295]
 [0.00324477]
 [0.06752689]
 [0.00903354]
 [0.98772266]
 [0.90013004]
 [0.99540184]]
```