EXPERIMENT 7

Aim: To implement union, intersection, complement, and difference operations on fuzzy sets. Also, to create fuzzy relations using the Cartesian product and perform max-min composition on fuzzy relations using Python and NumPy.

Theory:

Fuzzy set theory, introduced by Lotfi Zadeh in 1965, extends classical set theory by allowing elements to have degrees of membership. Unlike classical (crisp) sets where an element either belongs or doesn't belong to a set, in fuzzy sets, membership is a gradual concept represented by a value between 0 and 1.

The core operations in fuzzy set theory mimic classical set operations but are defined differently to accommodate the partial membership:

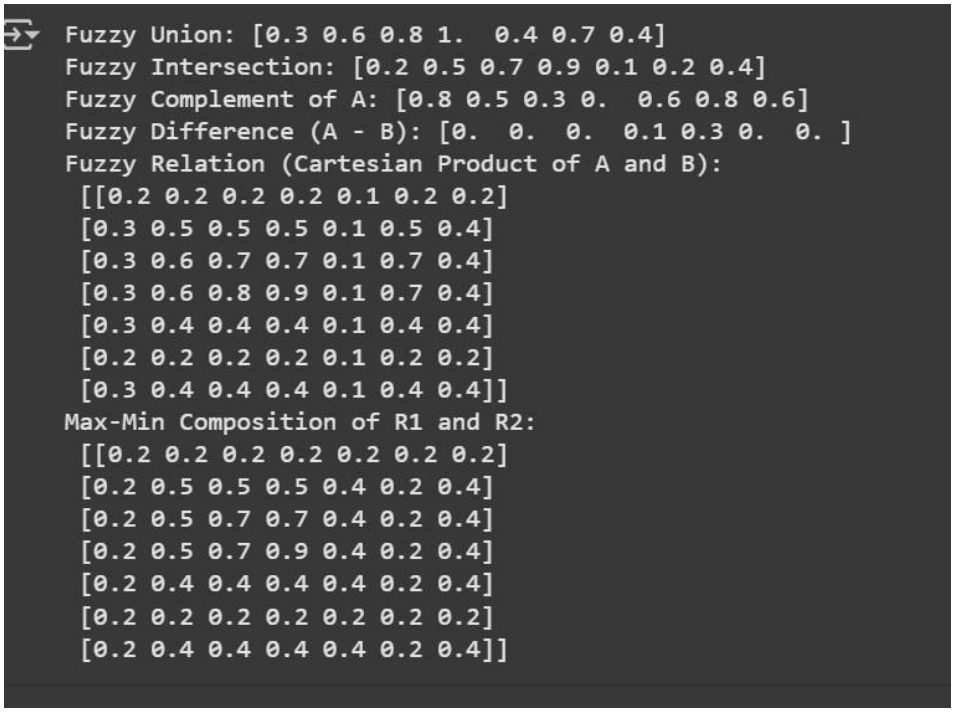
1. **Fuzzy Union (OR):**
   μA∪B(x)=max⁡(μA(x),μB(x))\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))μA∪B(x)=max(μA(x),μB(x))
   Represents the highest degree of membership from either set.
2. **Fuzzy Intersection (AND):**
   μA∩B(x)=min⁡(μA(x),μB(x))\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))μA∩B(x)=min(μA(x),μB(x))
   Represents the lowest degree of membership between the two sets.
3. **Fuzzy Complement (NOT):**
   μA‾(x)=1−μA(x)\mu_{\bar{A}}(x) = 1 - \mu_A(x)μA‾(x)=1−μA(x)
   Measures how much an element does not belong to a set.
4. **Fuzzy Difference:**
   μA−B(x)=max⁡(μA(x)−μB(x),0)\mu_{A - B}(x) = \max(\mu_A(x) - \mu_B(x), 0)μA−B(x)=max(μA(x)−μB(x),0)
   Shows how much of A is not in B.
5. **Cartesian Product:**
   The Cartesian product of fuzzy sets A and B results in a fuzzy relation, where each pair (a, b) is associated with a membership value typically defined as min⁡(μA(a),μB(b))\min(\mu_A(a), \mu_B(b))min(μA(a),μB(b)).
6. **Max-Min Composition:**
   Given two fuzzy relations R1 and R2, the max-min composition finds the degree to which one set relates to another through an intermediate set. It’s a crucial operation in fuzzy relational databases and fuzzy inference systems.

This experiment gives hands-on experience in applying fuzzy set operations and relational compositions, which are foundational in fuzzy logic controllers, decision support systems, and approximate reasoning models in artificial intelligence.

Code:

```
import numpy as np
def fuzzy_union(A, B):
    return np.maximum(A, B)
def fuzzy_intersection(A, B):
    return np.minimum(A, B)
def fuzzy_complement(A):
    return 1 - A
def fuzzy_difference(A, B):
    return np.maximum(A - B, 0)
def cartesian_product(A, B):
    return np.minimum.outer(A, B)
def max_min_composition(R1, R2):
    return np.maximum.reduce(np.minimum(R1[:, :, None], R2[None, :, :]), axis=1)
A = np.array([0.2, 0.5, 0.7, 1.0, 0.4, 0.2,0.4])
B = np.array([0.3, 0.6, 0.8, 0.9, 0.1, 0.7, 0.4])
union_result = fuzzy_union(A, B)
intersection_result = fuzzy_intersection(A, B)
complement_A = fuzzy_complement(A)
difference_result = fuzzy_difference(A, B)
R1 = cartesian_product(A, B)
R2 = cartesian_product(B, A)
composition_result = max_min_composition(R1, R2)
print("Fuzzy Union:", union_result)
print("Fuzzy Intersection:", intersection_result)
print("Fuzzy Complement of A:", complement_A)
print("Fuzzy Difference (A - B):", difference_result)
print("Fuzzy Relation (Cartesian Product of A and B):\n", R1)
print("Max-Min Composition of R1 and R2:\n", composition_result)
```

Output:



Learning Outcomes:

EXPERIMENT 8

Aim: To implement one-hot encoding of words and characters using Python and TensorFlow/Keras.

Theory:

One-hot encoding is a widely used technique in data preprocessing, especially for converting categorical or textual data into a numerical format that can be used in machine learning and deep learning algorithms. Since models such as neural networks and classification algorithms require numerical input, symbolic data like words or characters must be transformed in a way that preserves their identity without introducing unintended bias or relationships. One-hot encoding offers a simple and intuitive way to accomplish this.

In one-hot encoding, each unique item—be it a word, character, or other categorical variable—is represented as a vector of binary values. If there are **N** unique items in the dataset, each is encoded as a vector of length **N** with all positions set to 0 except for the one corresponding to the item's index, which is set to 1. For example, for a vocabulary of `["machine", "learning", "hello"]`, the word "learning" could be encoded as `[0, 1, 0]`. This encoding is *sparse* and *orthogonal*, ensuring that the encoded vectors are mutually exclusive and equally spaced.

There are two primary types of one-hot encoding explored in this experiment:

1. **Word-Level One-Hot Encoding:**
   This approach treats each distinct word as a unit. A tokenizer is used to assign an index to each word based on frequency or order of appearance. Then, the words are converted into one-hot vectors. This is particularly useful in document classification, language modeling, and basic NLP tasks, where understanding the presence of specific words matters more than their order or grammatical role.
2. **Character-Level One-Hot Encoding:**
   This method encodes individual characters rather than entire words. It is useful in low-level language processing tasks such as text generation, character-level sequence modeling, or spelling correction. Here, each character is assigned a unique index, and the entire string is transformed into a sequence of binary vectors.

In the experiment, the `Tokenizer` class from TensorFlow/Keras is used to map words to indices. These indices are then passed to the `to_categorical()` function, which generates the final one-hot encoded representations. For character encoding, a dictionary mapping of characters to integers is created manually, and the characters are similarly one-hot encoded using the same function.

Code:

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
words = ["hello", "world", "machine", "learning", "Sample", "Text","Output"]
tokenizer = Tokenizer()
tokenizer.fit_on_texts(words)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(words)
one_hot_words = to_categorical(sequences)
print("Word Index:", word_index)
print("One-hot Encoded Words:")
print(one_hot_words)
text = "hello world"
characters = list(set(text))
char_to_int = {c: i for i, c in enumerate(characters)}
int_to_char = {i: c for c, i in char_to_int.items()}
encoded_chars = [char_to_int[c] for c in text]
one_hot_chars = to_categorical(encoded_chars, num_classes=len(characters))
print("\nCharacter Index:", char_to_int)
print("One-hot Encoded Characters:")
print(one_hot_chars)
```

Output:

```
Word Index: {'hello': 1, 'world': 2, 'machine': 3, 'learning': 4, 'sample': 5, 'text': 6, 'output': 7}
One-hot Encoded Words:
[[0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]]

Character Index: {'l': 0, 'd': 1, 'r': 2, 'h': 3, 'e': 4, ' ': 5, 'w': 6, 'o': 7}
One-hot Encoded Characters:
[[0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0.]]
```

Learning Outcomes: