INTERNATIONAL COMPUTER SCIENCE COMPETITION

PROBLEM A : Neural Network Components

Explanation:

According to neural network architeture it has three main sections of layers **the input layer, hidden layers and output layer**. In the input layer is where the **data** is injected in the input nodes e.g (images,text,numerical data,etc), then this data is applied on different **weights values** and passed to different hidden layers with different applied **biases** and **activation fuctions** e.g(sigmoid, ReLU, ELU, etc), final the **output** is formed and passed to the output layer where it is then used for certain model **prediction,classification,** etc.

Answer:

$W_{12}^{(1)}$: a weight, connecting the first node for the first layer and second node of second layer

$\mathcal{E}$: summation notation, for the all applied weights and biases on a single node

$\mathcal{F}$: activation function

O : input data(input node)

O : bias

O : output prediction (output node)

PROBLEM B : Cake Calculator

Explanation:

If we are working to find the number of cakes that can be made and the left overs then knowing the amount either of sugar or flour that leads to the least production of cakes could really provide an out standing help, so that is why in the cone we make all possible cakes with the insufficient ingredient then calculate the left overs for the unused ingredients after the production of such cakes, then the output for the cakes made, left over flour and sugar.

Answer code (Python):

```python
import sys

def cake_calculator(flour: int, sugar: int) -> list:
```

```python
    # WRITE YOUR CODE HERE
    left_flour = 0
    left_sugar = 0
    cakes = 0
    if flour>0 and sugar>0 :
        n_cake_flour = flour//100 # max cakes with the flour
        n_cake_sugar = sugar//50  # max cakes with the sugar
        # the number of cakes is the minimum of the two
        cakes = min(n_cake_flour,n_cake_sugar)
        # we calculate the left over engridients by calculating the total
        # minus the used
        # ingredients for the cakes
        left_flour = flour - 100 * cakes
        left_sugar = sugar - 50 *cakes
    return [cakes,left_flour,left_sugar]
```

```python
# --- Main execution block. DO NOT MODIFY  ---
if __name__ == "__main__":
    try:
        # 1. Read input from stdin
        flour_str = input().strip()
        sugar_str = input().strip()

        # 2. Convert inputs to appropriate types
        flour = int(flour_str)
        sugar = int(sugar_str)

        # 3. Call the cake calculator function
        result = cake_calculator(flour, sugar)

        # 4. Print the result to stdout in the required format
        print(f"{result[0]} {result[1]} {result[2]}")

    except ValueError as e:
        # Handle errors during input conversion or validation
        print(f"Input Error or Validation Failed: {e}", file=sys.stderr)
        sys.exit(1)
    except EOFError:
        # Handle cases where not enough input lines were provided
        print("Error: Not enough input lines provided.", file=sys.stderr)
        sys.exit(1)
    except Exception as e:
        # Catch any other unexpected errors
        print(f"An unexpected error occurred: {e}", file=sys.stderr)
        sys.exit(1)
```

PROBLEM C: The School Messaging App

Question 1:

Explanation:

Standard text encoding using the same number of bit like for example
A -> 0001
B -> 0010
C -> 0101

Compared to the use of different length codes according to the probability difference for example

A -> 01
B -> 010
C -> 101

. Consumes a lot of space since all the characters are assign the same length meaning memory regardless of their probability in the text, another.
. The first approach could lead to the requirement of large bandwidth which then leads to the low amount of data transmitted if there are such bandwidth transmission limits
. High transmitting time due to some kind of big data with fixed bandwidth

Answer 1 : - Using different codes length **saves time** of sending
- It **requires a small bandwidth** for much data transmission hence one can transmit more data with bandwidth limits
- It requires **small transmission storage** hence preventing storage wastage and hence more messages can be transmitted within data limits

Question 2:

$$H = -\sum_{i=1}^{n} p_i * \log_2(p_i)$$

Character	Probability	Character	Probability
A	0.20	G	0.05
B	0.15	H	0.05
C	0.12	I	0.04
D	0.10	J	0.03
E	0.08	K	0.02
F	0.06	L	0.10

Calculation:

H = - (
0.2*log2(0.2)
+0.15*$\log_2$(0.15)
+0.12*$\log_2$(0.12)
+0.1*$\log_2$(0.1)
+0.08*$\log_2$(0.08)
+0.06*$\log_2$(0.06)
+0.05*$\log_2$(0.05)
+0.05*$\log_2$(0.05)
+0.04*$\log_2$(0.04)
+0.03*$\log_2$(0.03)
+0.02*$\log_2$(0.02)
+0.1*$\log_2$(0.1)
) = 3.32

Explanation:

This answer means that characters can be represented using an average of number of bits H = 3.37, but since we can't have number of bits in decimal we consider assigning the number of bits cross to the H = 3.37 say it might be 3 or4 according to the probability for the ones with low have more bits than those with high probabilities

Answer 2:

the value H=3.37 represent the **average number of bit to be used** for representing characters in optimal encoding.

Question 3:

Explanation:

To calculate the average code length of my Fano code **we calculate the number of bits used per character multiplied by its own probability**. Then we compare the result with the theoretical entropy mathematical using the efficiency formula

Answer 3:

L = (
0.2*3
+0.15*3
+0.12*3
+0.10*4
+0.08*4
+0.06*4
+0.05*3
+0.05*4
+0.04*4
+0.03*4
+0.02*4
+0.10*4
) = 3.48

H = 3.32
L = 3.48

Efficiency = (H/L) * 100% = (3.32/3.48) * 100% = 95.4%

PROBLEM D: Word Search Puzzle

Explanation:

Here we are tasked at creating a word puzzle of **10*10** dimension containing the provided word in certain different orientations, so the first thing is to check whether this word **is able to fit** in the puzzle of **10*10**, then fit it according to a specific direction might be (**up,down,right,left,diagonal directions**) finally we return that such word puzzle.

Answer code(python):

```
import sys
import random
import string
```

```
def create_crossword(words: list) -> list:
    """
    Generate a 10x10 word search puzzle containing the given words.

    Args:
        words: A list of words to include in the puzzle.

    Returns:
        A 2D array (list of lists) representing the word search puzzle.
    """
    # WRITE YOUR CODE HERE
```

```
def create_empty_grid():
    return [[" " for _ in range(10)] for _ in range(10)]
```

```
# Function checkc if a word can be placed at the specified position in
the grid
# with the specified direction (dr, dc).
def can_place(grid, word, row, col, dr, dc):
    n = len(grid)
    for i in range(len(word)):
        r = row + dr * i
        c = col + dc * i
        if not (0 <= r < n and 0 <= c < n):
            return False
        if grid[r][c] != " " and grid[r][c] != word[i]:
            return False
    return True
```

```
# Function places a word in the grid at the specified position and
direction.
def place_word(grid, word, row, col, dr, dc):
    for i in range(len(word)):
        r = row + dr * i
        c = col + dc * i
        grid[r][c] = word[i]
```

```
# Function fills empty spaces in the grid with random letters.
def fill_empty(grid):
    n = len(grid)
    letters = string.ascii_lowercase
    for i in range(n):
        for j in range(n):
            if grid[i][j] == " ":
                grid[i][j] = random.choice(letters)
```

```
# Function to create a crossword puzzle with the given words.
def create_crossword(words):
    directions = [
```

```
        (0, 1),    # right direction
        (1, 0),    # down
        (1, 1),    # down-rigth
        (0, -1),   # left
        (-1, 0),   # up
        (-1, -1),  # up-left
        (1, -1),   # down-left
        (-1, 1),   # up-right
        ]
    grid = create_empty_grid()
    for word in words:
        placed = False
        for _ in range(100):  # try 100 times to place
            dr, dc = random.choice(directions)
            row = random.randint(0, 9)
            col = random.randint(0, 9)
            if can_place(grid, word, row, col, dr, dc):
                place_word(grid, word, row, col, dr, dc)
                placed = True
                break

    fill_empty(grid)
    return grid
```

```
# --- Main execution block. DO NOT MODIFY.  ---
if __name__ == "__main__":
    try:
        # Read words from first line (comma-separated)
        words_input = input().strip()
        words = [word.strip() for word in words_input.split(',')]

        # Generate the word search puzzle
        puzzle = create_crossword(words)

        # Print the result as a 2D grid
        for row in puzzle:
            print('   '.join(row))

    except ValueError as e:
        print(f"Input Error: {e}", file=sys.stderr)
        sys.exit(1)
    except EOFError:
        print("Error: Not enough input lines provided.",
file=sys.stderr)
        sys.exit(1)
    except Exception as e:
        print(f"An unexpected error occurred: {e}", file=sys.stderr)
        sys.exit(1)
```

PROBLEM E : Functional Completeness of NAND

Explanation:

To prove that NAND is functional completeness we are going to simply represent all the other logic gates using only the NAND gate, if we do so we will be done with the proof.

Answer:

. Representing NOT in terms of NAND (first proof)

-> we say that A is a binary value (e.g 1 0r 0)

Simply as it can be solved
NOT(A) = (A) NAND (A)
Which automatically puts the NOT into only NAND representation

. Representing AND in terms of NAND (second proof)

From the results above we can also prove this by starting at
(A) AND (A) = NOT((A) NAND (A))

From first proof :

(A) AND (A)= ((A) NAND (A)) NAND ((A) NAND (A))

. Representing OR in terms of NAND (third proof)

Form de morgan's laws:
NOT((A) OR (A)) = (NOT(A)) NAND (NOT (A))

=> (A) OR (A) = NOT ((NOT(A)) NAND (NOT (A)))
From first proof
(A) OR (A) = (((A) NAND (A)) NAND ((A) NAND (A))) NAND (((A) NAND (A)) NAND ((A) NAND (A)))

Hence proved!!

Note : In the last question consider the colored logical operation because it will help you determine the result of expanding according to the previous known expansion.

End of the solution file…