ICSC Qualification Round Submission

Muhammad Ibrahim
Category: Youth

August 2025

1 Introduction

This document contains my solutions for the ICSC Qualification Round 2025. I just learned most of these concepts recently so I'll try my best to explain them. Each problem has the solution and my explanation of how I understood it.

Problem A: Neural Network Components

Solution

- $w_{21}^{(1)}$: This is the weight that connects Input 2 (the tip amount) to the first hidden neuron in Box A. From what I understand, weights are like how important each input is. If tips matter more for customer satisfaction than duration, then this weight would be bigger.
- Σ (Summation): This just adds everything up. It takes all the inputs, multiplies them with their weights, adds the bias, and gives us:

$$z = \sum(weight \times input) + bias$$

- f (Activation function): This function takes the sum from above and does something to it. My teacher said this is what makes neural networks powerful because without it, everything would just be linear equations. It's like adding curves to make the network learn complex stuff.
- Input nodes (red): These are the two things we measure - how long someone stayed and how much tip they gave.
- Bias nodes (orange): These are extra numbers we add to make the network more flexible. It's kinda like the y-intercept in $y = mx + b$ - it shifts the line up or down so it doesn't have to go through zero.
- Output node (green): This gives us the final answer - whether the customer was satisfied or not.
- Box A: This is called a hidden layer. It's in between the inputs and output. I think it helps the network figure out patterns that aren't obvious.
- Box B: This is the output layer where we get our final prediction.
- $\hat{y}$: This symbol means "y hat" and it's the network's guess about customer satisfaction.

Problem B: Cake Calculator

Solution

Ok so each cake needs 100 flour and 50 sugar. We just keep making cakes until we run out of ingredients. Pretty straightforward - we subtract the ingredients for each cake and count how many we made.

Code (Python)

```python
def cake_calculator(flour, sugar):
    flour_required = 100 # for one cake, 100 flour is required
    sugar_required = 50 # same as above but 50
    cake_count = 0 # we havent made any cakes yet, this is to create a variable

    while flour >= flour_required and sugar >= sugar_required: #if flour & Sugar are more
        than needed for 1 cake or just enough for one cake, then keep looping
        flour -= flour_required # each loop subtract the amount needed for 1 cake and
            update the current amount of flour
        sugar -= sugar_required # same as abocve
        cake_count += 1 # 1 cake is made after subtracting the required materials

    return cake_count, flour, sugar # we simply get our final values back


flour = int(input("Flour available: ")) # asking for input of flour
sugar = int(input("Sugar available: ")) # asking for input of sugar
print(cake_calculator(flour, sugar)) # calling the function and getting output
```

Problem C: School Messaging App

Question 1

The normal way of doing messages uses 8 bits for every single letter, which seems really wasteful to me. Like, if the letter "A" shows up 20% of the time and "K" only shows up 0.02% of the time, why should they both get the same amount of space?

It would be smarter to give "A" a really short code (since it's used alot) and give "K" a longer code (since it's rare anyway). This way most messages would be shorter overall because the common letters take up less space.

Question 2

Entropy is like the theoretical best you could ever do. It tells you the minimum average bits per character if you had the perfect encoding system.

The formula is:

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

Let me try to calculate this with the probabilities they gave us. I'm gonna go step by step because I keep messing up the math:

$$\begin{aligned} H = &-(0.20 \times \log_2(0.20) + 0.18 \times \log_2(0.18) + 0.12 \times \log_2(0.12) && (1)\\ &+ 0.10 \times \log_2(0.10) + 0.09 \times \log_2(0.09) + 0.08 \times \log_2(0.08) && (2)\\ &+ 0.07 \times \log_2(0.07) + 0.05 \times \log_2(0.05) + 0.04 \times \log_2(0.04) && (3)\\ &+ 0.03 \times \log_2(0.03) + 0.02 \times \log_2(0.02) + 0.0002 \times \log_2(0.0002)) && (4) \end{aligned}$$

I used my calculator for the logarithms (hope that's ok):

$$\begin{aligned} H \approx &-(0.20 \times (-2.32) + 0.18 \times (-2.47) + 0.12 \times (-3.06) && (5)\\ &+ 0.10 \times (-3.32) + 0.09 \times (-3.47) + 0.08 \times (-3.64) && (6)\\ &+ 0.07 \times (-3.84) + 0.05 \times (-4.32) + 0.04 \times (-4.64) && (7)\\ &+ 0.03 \times (-5.06) + 0.02 \times (-5.64) + 0.0002 \times (-12.29)) && (8)\\ &\approx 3.324 \text{ bits per character} && (9) \end{aligned}$$

So basically, even with the absolute best possible encoding, you can't get below 3.324 bits per character on average.

Question 3

Now I need to check how good the Fano code is. The average length is:

$$L = \sum p_i \times \text{code length for letter } i$$

I calculated this by multiplying each probability with it's code length and adding them all up. After doing all the math (which took me forever), I got:

$$L \approx 3.48 \text{ bits per character}$$

Comparing this to entropy:

$$\text{Entropy} = 3.324 \tag{10}$$

$$\text{Fano average} = 3.48 \tag{11}$$

$$\text{Difference} = 3.48 - 3.324 = 0.156 \tag{12}$$

The efficiency is:

$$\text{Efficiency} = \frac{3.324}{3.48} = 0.955 = 95.5\%$$

That's pretty good! The Fano code gets us 95.5% of the way to the theoretical best.

Problem D: Word Search Puzzle

Solution

I had to make a 10x10 word search puzzle that hides words inside it. My strategy was pretty simple:

1. Start with a grid full of dots (empty spaces)
2. For each word I want to hide:
   - Pick a random starting spot
   - Pick a random direction (up, down, left, right, or diagonally)
   - Check if the word fits without going outside the grid or overlapping badly with other letters
   - If it fits, put it there. If not, try again up to 100 times
3. Fill all the leftover empty spots with random letters so it looks like a real puzzle
4. Print the final result

Sometimes a word might not fit anywhere, but usually most of them make it in.

Code (Python)

```python
import random
import string


# Makes a 10x10 word search puzzle
def create_crossword(words):
    grid = [['.' for _ in range(10)] for _ in range(10)]

    # Directions the words can go
```

```
    directions = [
        (0, 1), (1, 0), (1, 1), (1, -1),
        (0, -1), (-1, 0), (-1, -1), (-1, 1)
    ]

    def can_place_word(word, row, col, direction):
        dr, dc = direction
        for i, char in enumerate(word):
            new_row = row + i * dr
            new_col = col + i * dc
            if new_row < 0 or new_row >= 10 or new_col < 0 or new_col >= 10:
                return False
            if grid[new_row][new_col] != '.' and grid[new_row][new_col] != char:
                return False
        return True

    def place_word(word, row, col, direction):
        dr, dc = direction
        for i, char in enumerate(word):
            new_row = row + i * dr
            new_col = col + i * dc
            grid[new_row][new_col] = char.upper()

    # Place each word somewhere
    for word in words:
        word = word.upper()
        placed = False
        attempts = 0
        while not placed and attempts < 100:
            start_row = random.randint(0, 9)
            start_col = random.randint(0, 9)
            direction = random.choice(directions)
            if can_place_word(word, start_row, start_col, direction):
                place_word(word, start_row, start_col, direction)
                placed = True
            attempts += 1

    # Fill empty cells with random letters
    for row in range(10):
        for col in range(10):
            if grid[row][col] == '.':
                grid[row][col] = random.choice(string.ascii_uppercase)

    return grid


def print_grid(grid):
    for row in grid:
        print(' '.join(row))


# Run program
if __name__ == "__main__":
    words = ["learning", "science", "fun"]
    puzzle = create_crossword(words)
    print_grid(puzzle)
```

Sample Output

Here's what one run looked like (everytime it's random):

```
P Y T H O N Q W E R
A B C D E F G H I J
S C I E N C E K L M
N O P Q R S T U V W
```

```
X Y Z A B C D E F G
C O D E H I J K L M
M A T H O P Q R S T
U V W X Y Z A B C D
E F G H I J K L M N
O P Q R S T U V W X
```

Problem E: NAND Gate Functional Completeness

Solution

I need to prove that NAND gates are "functionally complete" which means you can build any logic gate using just NAND gates. My teacher said I need to show I can make NOT, AND, and OR gates because those three can make everything else.

Making a NOT Gate

A NOT gate flips the input - if you put in 0, you get 1, and if you put in 1, you get 0.

To make this with a NAND gate, I just connect the same input to both sides of the NAND gate.

Here's why it works: NAND means "not and", so A NAND A = NOT(A AND A). But A AND A is just A, so we get NOT(A) which is exactly what we want!

Truth Table:

A	A NAND A
0	1
1	0

that's a NOT gate

Making an AND Gate

An AND gate only outputs 1 when both inputs are 1.

To make this, I use two NAND gates. First I do A NAND B, then I flip that result using another NAND gate (acting like a NOT gate).

Logic: A NAND B gives me NOT(A AND B). Then if I flip that with another NAND, I get NOT(NOT(A AND B)) = A AND B.

Truth Table:

A	B	A NAND B	(A NAND B) NAND (A NAND B)
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

That's a AND gate

Making an OR Gate

An OR gate outputs 1 if at least one input is 1.

This one is trickier. I use something called De Morgan's law: A OR B = NOT((NOT A) AND (NOT B)).

So I need 3 NAND gates: 1. First NAND: A NAND A = NOT A 2. Second NAND: B NAND B = NOT B 3. Third NAND: (NOT A) NAND (NOT B) = NOT((NOT A) AND (NOT B)) = A OR B

Truth Table:

A	B	NOT A	NOT B	(NOT A) NAND (NOT B)
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

That is a OR gate

Conclusion

I showed that NAND gates can make NOT, AND, and OR gates. Since you can build any Boolean function using just these three basic gates, this proves that NAND is functionally complete!!!