Exercise 8

1. Maximum MEC of Neural Networks

a) 4 * 3 + min(4*3, 3) + min(4, 3) = 18

b) 3 + 4 + 4 = 11

c) Binary classification

maximum amount of rows for (a) - 18 rows

maximum amount of rows for (b) - 11 rows

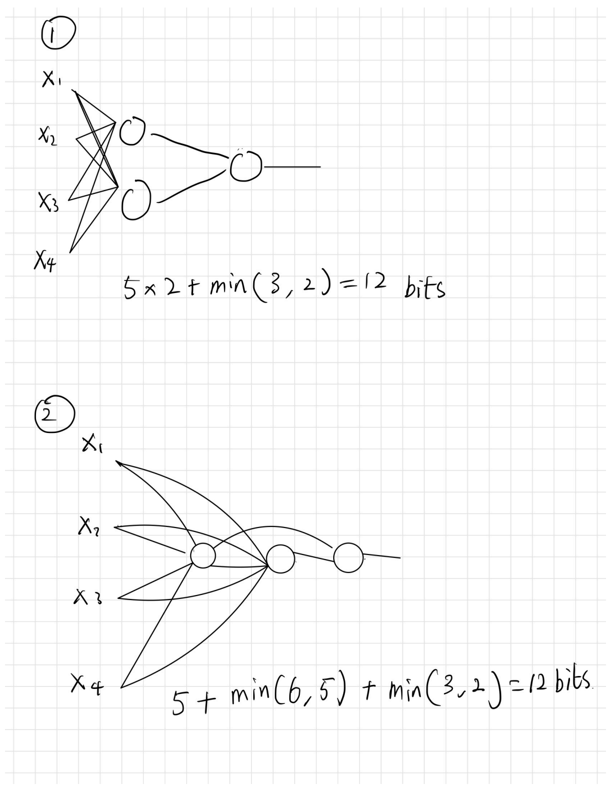
d) 4 classes

maximum amount of rows for (a) - 9 rows

maximum amount of rows for (b) - 5 rows

2. Design NNs

Draw two different neural network architectures that can guarantee to memorize the training data of a 12-instance binary classification problem of 4-dimensional inputs (assuming perfect training)


1
X_1
X_2
X_3
X_4
$5\times 2+\min(3,2)=12$ bits
2
X_1
X_2
X_3
X_4
$5+\min(6,5)+\min(3,2)=12$ bits

4. Upper Bounds:

a) Do Exercise 40.8 in MacKay's book (MacKay 2003). It is cited here as follows:

Estimate in bits the total sensory experience that you have had in your life – visual information, auditory information, etc. Estimate how much information you have memorized. Estimate the information content of the works of Shakespeare. Compare these with the capacity of your brain assuming you have neurons each making 1000 synaptic connections and that the (information) capacity result for one neuron (two bits per connection) applies. Is your brain full yet?

Note that MacKay is right to suggest using information capacity for this estimate as image and acoustic data are relatively high dimensional and he also suggests 1000 connections per neuron.

The total sensory experience that I have had in my life

https://www.britannica.com/science/information-theory/Physiology

According to the article above, each second:

- the eye receives 10^7 bits of information
- the skin receives 10^6 bits of information
- the ear receives 10^5 bits of information
- the nose receives 10^5 bits of information

- the tongue receives 10^3 bits of information

In [ ]:
```
eyes_bits_per_second = 10**7
skin_bits_per_second = 10**6
ears_bits_per_second = 10**5
nose_bits_per_second = 10**5
tongue_bits_per_second = 10**3
```

In [ ]:
```
# I am 23 years old, so I have lived for:
seconds = 23 * 365 * 24 * 60 * 60
total_eyes_bits = eyes_bits_per_second * seconds
total_skin_bits = skin_bits_per_second * seconds
total_ears_bits = ears_bits_per_second * seconds
total_nose_bits = nose_bits_per_second * seconds
total_tongue_bits = tongue_bits_per_second * seconds

print(f"I am {seconds:e} seconds old") # seconds
print(f"I have seen {total_eyes_bits:e} bits")
print(f"I have felt {total_skin_bits:e} bits")
print(f"I have heard {total_ears_bits:e} bits")
print(f"I have smelled {total_nose_bits:e} bits")
print(f"I have tasted {total_tongue_bits:e} bits")

sum_bits = total_eyes_bits + total_skin_bits + total_ears_bits + total_nose_bits +
print(f"Total bits: {sum_bits:e}")
```

```
I am 7.253280e+08 seconds old
I have seen 7.253280e+15 bits
I have felt 7.253280e+14 bits
I have heard 7.253280e+13 bits
I have smelled 7.253280e+13 bits
I have tasted 7.253280e+11 bits
Total bits: 8.124399e+15
```

Assume I can memorize 10% of the information we receive. Then, I have memorized:

In [ ]:
```
memorized_bits = sum_bits * 0.1
print(f"Memorized bits: {memorized_bits:e}")
```

```
Memorized bits: 8.124399e+14
```

There are 884,421 total words in Shakespeare's 43 works

In [ ]:
```
num_words = 884421
avg_word_length = 5
num_letters = num_words * avg_word_length

total_bits_ascii = num_letters * 8
print(f"Total bits in Shakespeare's work (ASCII encoding): {total_bits_ascii:e}")
```

```
Total bits in Shakespeare's work (ASCII encoding): 3.537684e+07
```

In [ ]:
```
## Brain capacity
num_neurons = 10**11
num_synapses = num_neurons * 1000
num_layers = 10
```

```
brain_capacity = 2 * num_synapses * num_layers

print(f"Brain capacity: {brain_capacity:e} bits")
```

```
Brain capacity: 2.000000e+15 bits
```

In [ ]:
```
# Upon researching, the brain capacity is 2.5 petabytes according to
# https://www.medanta.org/patient-education-blog/what-is-the-memory-capacity-of-a-h
brain_capacity_2 = 2.5 * 10**15 * 8
print(f"Alternative brain capacity: {brain_capacity_2:e} bits")
```

```
Alternative brain capacity: 2.000000e+16 bits
```

In [ ]:
```
proportion_used = memorized_bits / brain_capacity
print(f"Proportion of brain used: {proportion_used:.2f}")
shakespeare_proportion = total_bits_ascii / brain_capacity
print(f"Proportion of brain capacity to memorize Shakespeare's works: {shakespeare_
```

```
Proportion of brain used: 0.41
Proportion of brain capacity to memorize Shakespeare's works: 0.00
```

In [ ]:
```
proportion_used = memorized_bits / brain_capacity_2
print(f"Proportion of brain used (alt_source): {proportion_used:.2f}")
shakespeare_proportion = total_bits_ascii / brain_capacity_2
print(f"Proportion of brain capacity to memorize Shakespeare's works (alt_source):
```

```
Proportion of brain used (alt_source): 0.04
Proportion of brain capacity to memorize Shakespeare's works (alt_source): 0.00
```

b) Expand Algorithm 8 to work with more than one binary classification.

In [ ]:
```
#  Calculate the upper-bound memory-equivalent capacity expected for a three-layer
import numpy as np
import math
import pandas as pd

def calculate_mec(X, y, task="classification"): # task should be either "classifica
    thresholds = 0
    num_rows, num_features = X.shape
    table = np.zeros((num_rows, 2))
    unique_labels = np.unique(y)
    for row in range(num_rows):
        table[row, 0] = np.sum(X[row])
        table[row, 1] = y[row]
    sorted_table = sorted(table, key=lambda x: (x[0], x[1]))
    sorted_table = np.array(sorted_table)
    label = 0
    # print(sorted_table)
    for row in range(num_rows):
        if not sorted_table[row,1] == label:
            thresholds += 1
            label = sorted_table[row,1]
    minthreshs = math.log(thresholds + 1, len(unique_labels))
    if task == "classification":
        mec = minthreshs * (num_features + 1) + (minthreshs + 1)
    elif task == "regression":
        mec = minthreshs * (num_features + 1)
    return mec
```

In [ ]:
```
# Binary classification
X = np.random.randint(0, 2, (100, 10))
y = np.random.randint(0, 2, (100, 1))
mec = calculate_mec(X, y, "classification")
mec
```

Out[ ]:
```
50.04955409500408
```

In [ ]:
```
# Multi-class classification
X = np.random.randint(0, 2, (100, 10))
y = np.random.randint(0, 10, (100, 1))
mec = calculate_mec(X, y, "classification")
mec
```

Out[ ]:
```
21.490842113175233
```

c) Expand Algorithm 8 to work with regression.

In [ ]:
```
# Multi-class regression
# X = np.random.randint(0, 2, (100, 10))
# y = np.random.randint(0, 10, (100, 1))
mec = calculate_mec(X, y, task="regression")
mec
```

Out[ ]:
```
18.7832719370773
```