Midterm Assignment

Exercise 6.1

a.

```
Binary Classification:
d=2: n_full=4, Avg. req. points for memorization n_avg=3.52, n_full/n_avg=1.1356898517673888
d=4: n_full=16, Avg. req. points for memorization n_avg=10.81, n_full/n_avg=1.4799405646359585
d=8: n_full=256, Avg. req. points for memorization n_avg=152.67, n_full/n_avg=1.676767676767677
d=16: n_full=65536, Avg. req. points for memorization n_avg=32899.60, n_full/n_avg=1.992
```

b.

```
Multi-Class Classification (3 classes):
d=2: n_full=4, Avg. req. points for memorization n_avg=2.62, n_full/n_avg=1.5252679938744256
d=4: n_full=16, Avg. req. points for memorization n_avg=9.14, n_full/n_avg=1.750439367311072
d=8: n_full=256, Avg. req. points for memorization n_avg=137.51, n_full/n_avg=1.861682242990654
d=16: n_full=65536, Avg. req. points for memorization n_avg=33623.39, n_full/n_avg=1.949119373776908
```

Exercise 6.2

a.

```
Train on the Iris Dataset
Number of clauses for strategy 1 (Generate one clause per row): 100
1.0
Number of clauses for strategy 2 (Consolidate on single feature): 2
1.0
Number of clauses for strategy 3 (Decision tree pruning): 3
1.0
```

b.

```
Train on the Breast Cancer Dataset
Number of clauses for strategy 1 (Generate one clause per row): 569
0.9473684210526315
Number of clauses for strategy 2 (Consolidate on single feature): 2
0.9385964912280702
Number of clauses for strategy 3 (Decision tree pruning): 43
0.956140350877193
```

c.

```
Train on the Artificial Dataset
Number of clauses for strategy 1 (Generate one clause per row): 1000
0.695
Number of clauses for strategy 2 (Consolidate on single feature): 2
0.67
Number of clauses for strategy 3 (Decision tree pruning): 231
0.645
```

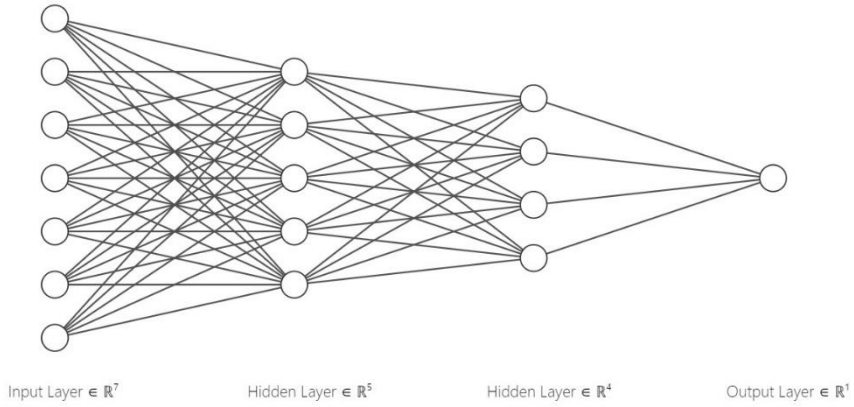
Exercise 6.3

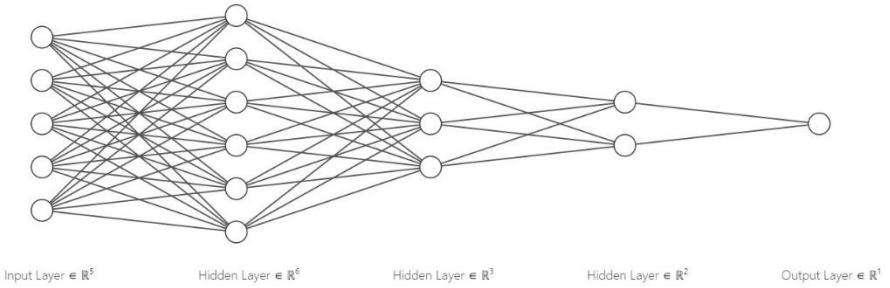
```
Original string length: 10000
Compressed data length: 7527
Compression ratio: 1.328550513484788
The expected compression ratio for a random string generated in this manner is likely to be close to 1 or slightly above 1. This is because random d
ata, such as a randomly generated string, does not have much redundancy or patterns that can be effectively compressed by lossless compression algor
ithms. In fact, random data can sometimes even result in larger compressed sizes due to the overhead introduced by the compression algorithm.
```

Exercise 8.1

a. MEC = 4*3 + min(4*3, 3) + min(4*3, 3) + min(4, 3) = 21 bits
b. MEC = 3 + 4 + 4 = 11 bits
c. For network a, the maximum amount of rows is 21; for network b, it is 11.
d. Both zero, since both networks can only perform binary classification (only one output neuron).

Exercise 8.2





Exercise 8.4

a. Visual information: The human eye can perceive a vast range of colors and shapes. Assume an average of 10 million colors and 100,000 shapes. This gives us approximately 27 bits (2^27) to represent any visual input.
Auditory information: The human ear can distinguish various frequencies and volumes. Let's assume an average of 100,000 distinguishable sounds, resulting in approximately 17 bits (2^17) for auditory input.
Tactile, olfactory, and gustatory experiences contribute additional bits.
Total sensory experience (rough estimate): 44 bits.

Average human memory capacity: 2.5 petabytes (2.5*10^15 bytes). Assuming each byte represents 8 bits, the total memorized information is approximately 20 exabits (2*10^19 bits).

William Shakespeare’s works, including plays, sonnets, and poems, contain rich language, themes, and characters. The complete works of Shakespeare consist of approximately 884,421 words. Assuming an average word length of 5 characters per word, we have approximately 4.4 million characters. Each character can be represented by 8 bits (ASCII encoding), resulting in an information content of approximately 35.2 megabits (35.2*10^6 bits).

The human brain contains approximately 100 billion neurons, each forming around 1,000 synaptic connections. If we assume 2 bits per synaptic connection, the total capacity of the brain is approximately 200 terabits (200*10^12 bits).

Sensory experience: 44 bits
Memorized information: 20 exabits
Shakespeare's works: 35.2 megabits
Brain capacity: 200 terabits

Comparing these values, it's clear that our brains are far from being “full.” The brain’s immense capacity allows us to continue learning, experiencing, and creating throughout our lives.

b.

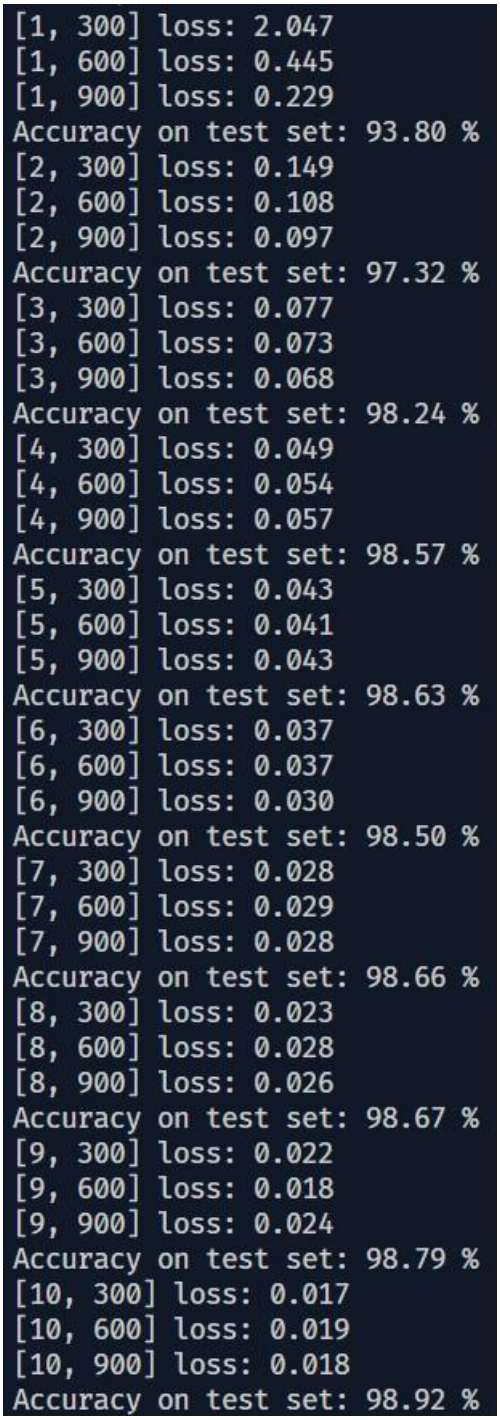
```
def memorize_multi_class(data, labels, num_classes):
    thresholds = 0
    table = []
    sortedtable = []
    class_id = 0

    # Create a table with (Σx[i]*d[i], label[i]) for each data point
    for i in range(len(data)):
        table.append((sum(data[i]), labels[i]))

    # Sort the table based on the first column (Σx[i]*d[i])
    sortedtable = sorted(table, key=lambda x: x[0])

    # Determine the thresholds and update class_id
    for i in range(len(sortedtable)):
        if sortedtable[i][1] != class_id:
            class_id = sortedtable[i][1]
            thresholds += 1

    # Calculate the minimum number of thresholds
    minthreshs = log2(thresholds + 1)

    # Calculate mec
    mec = (minthreshs * (len(data[0]) + num_classes)) + (minthreshs +
num_classes)

    return mec
```

c.

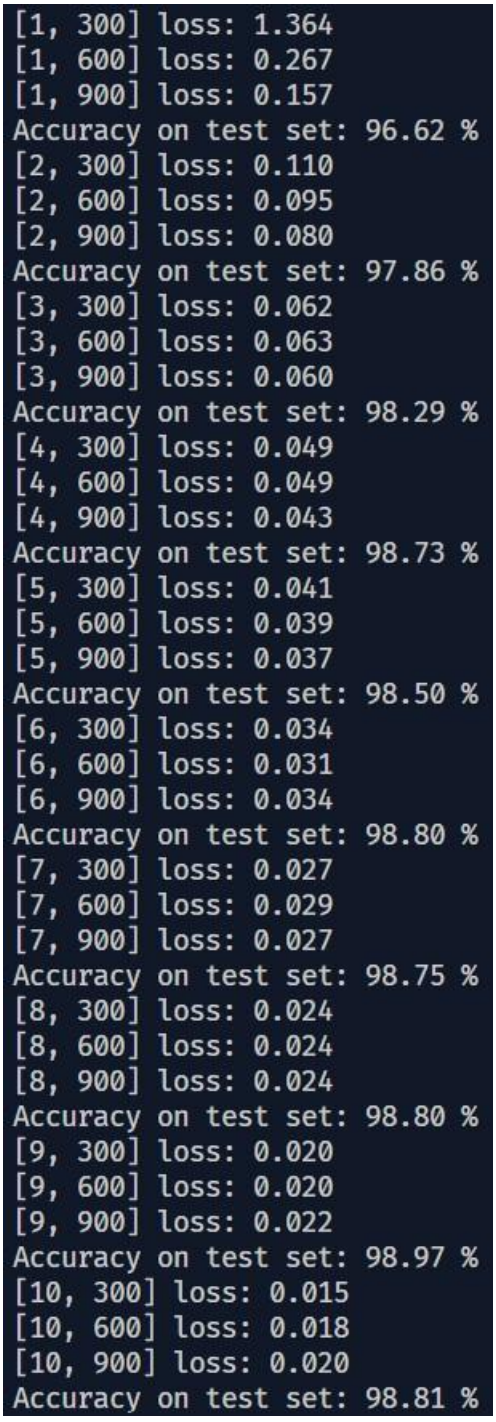
```
def memorize_regression(data, labels):
    thresholds = 0
    table = []
    sortedtable = []
    class_id = None

    # Create a table with (Σx[i]*d[i], label[i]) for each data point
    for i in range(len(data)):
        table.append((sum(data[i]), labels[i]))

    # Sort the table based on the first column (Σx[i]*d[i])
    sortedtable = sorted(table, key=lambda x: x[0])

    # Determine the thresholds and update class_id
    for i in range(len(sortedtable)):
        if sortedtable[i][1] != class_id:
            class_id = sortedtable[i][1]
            thresholds += 1

    # Calculate the minimum number of thresholds
    minthreshs = log2(thresholds + 1)

    # Calculate mec
    mec = (minthreshs * len(data[0])) + minthreshs

    return mec
```

Exercise 9.1

Before reducing:



After reducing:



We can see that after reducing hyperparameter search space, the accuracy on the test set improved.