Homework 1

Anthon Odengard
19950114-3953

September 2023

1 Boolean functions

dimensions	samples	separable functions
2	16	14
3	256	104
4	10000	276
5	10000	0

Table 1: Number of linearly separable functions of a n-dimensional Boolean function under the constraint samples $\leq 10^4$

For dimensions ≤ 3 the number of possible functions i less then 10^4 since number of unique Boolean functions k is given by: $k = 2^{2^n}$ where n denotes the number of dimensions. Therefore the functions were sampled $\mathbf{min}(k, 10^4)$ times. From our results its evident that the number of linearly separable functions does not grow in proportion to the number of unique Boolean functions. I have not come across any general formula that describes the number of linearly separable functions given Boolean function of n-dimensions. The results given by the computer program are not surprising given the rapid divergence between a set of Boolean functions and the number of linearly separable functions of the same set from the get go. The fact that we restrict the number of sampled functions also makes it less probable that we sample a linearly separable one as we increase n. This is due to the discrepancy that arises in the partition of linearly separable functions of a set of unique n-dimensional Boolean functions.

```python
import numpy as np
import itertools
from perceptron import Perceptron as perceptron
import matplotlib.pyplot as plt
#sample a boolean function


def generate_binary_inputspace(dimensions):
    binary_inputspace = np.array(list(itertools.product([0, 1], repeat=dimensions)))
    return binary_inputspace


def boolean_function_sampler(number_of_samples, number_of_dimensions):

    input_space = generate_binary_inputspace(number_of_dimensions)
    number_of_samples = min(number_of_samples, np.power(2, pow(2,number_of_dimensions)))
    number_of_entries = input_space.shape[0]
    sampled_functions = np.zeros((number_of_samples, number_of_entries, number_of_dimensions+1))
    sampled_outputs = np.zeros((number_of_samples, number_of_entries,1))
    input_space = generate_binary_inputspace(number_of_dimensions)
    j = 0
    last_element = 0

    while last_element == 0:

        random_output = np.random.randint(2, size=(number_of_entries, 1)) * 2 - 1
        random_function = np.append(input_space, random_output, axis=1)

        if not any((random_output == sample).all() for sample in sampled_outputs):
            sampled_functions[j] = random_function
            sampled_outputs[j] = random_output
            j = j + 1

        last_element = sampled_functions.flatten()[-1]
        if j%10 == 0:
            print(j)

    return sampled_functions


def initiate_weights(number_of_weights):

    distribution_mean = 0
    distribution_variance = 1/number_of_weights
    weight_matrix = np.random.normal(distribution_mean, distribution_variance, number_of_weights)

    return weight_matrix


# Testing
eta = 0.05
threshold = 0
nr_of_samples = np.power(10,4)

weights_2D = initiate_weights(2)
weights_3D = initiate_weights(3)
weights_4D = initiate_weights(4)
weights_5D = initiate_weights(5)
threshold = 0

perceptron_2D = perceptron(weights_2D, threshold)
perceptron_3D = perceptron(weights_3D, threshold)
perceptron_4D = perceptron(weights_4D, threshold)
perceptron_5D = perceptron(weights_5D, threshold)

boolean_funtion_space_2D = boolean_function_sampler(nr_of_samples, 2)
boolean_funtion_space_3D = boolean_function_sampler(nr_of_samples, 3)
boolean_funtion_space_4D = boolean_function_sampler(nr_of_samples, 4)
boolean_funtion_space_5D = boolean_function_sampler(nr_of_samples, 5)

nr_linearly_seperable_2D = perceptron_2D.evaluate_function_space(boolean_funtion_space_2D, weights_2D, threshold, eta)
nr_linearly_seperable_3D = perceptron_3D.evaluate_function_space(boolean_funtion_space_3D, weights_3D, threshold, eta)
nr_linearly_seperable_4D = perceptron_4D.evaluate_function_space(boolean_funtion_space_4D, weights_4D, threshold, eta)
nr_linearly_seperable_5D = perceptron_5D.evaluate_function_space(boolean_funtion_space_5D, weights_5D, threshold, eta)

print(nr_linearly_seperable_2D)
print(nr_linearly_seperable_3D)
```

```
print(nr_linearly_seperable_4D)
print(nr_linearly_seperable_5D)
```

```python
import numpy as np


class Perceptron:

    def __init__(self, weights, threshold):
        self.weights = weights
        self.threshold = threshold

    def evaluate_perceptron(self, inputs):
        size_input_space = len(inputs)-1
        b = self.weights * inputs[0:size_input_space]
        b = np.sum(b) - self.threshold
        if b == 0:
            signum_b = 1
        else:
            signum_b = np.sign(b)
        return signum_b

    def evaluate_perceptron_list(self, inputs):
        outputs = np.zeros(len(inputs))
        i = 0
        for input in inputs:
            signum_b = self.evaluate_perceptron(input)
            outputs[i] = signum_b
            i = i + 1
        return outputs

    def update_weights(self, pattern, output, eta):
        size_input_space = len(pattern)-1
        inputs = pattern[0:size_input_space]
        error = pattern[-1] - output
        dw = eta * error * inputs
        self.weights = self.weights + dw

    def evaluate_function_space(self, function_space, weights, threshold, eta):
        print('started evaluation')
        linearly_separable_functions = 0

        for function in function_space:
            self.weights = weights
            self.threshold = threshold
            self.train_network(function, eta, eta, 20)
            outputs = self.evaluate_perceptron_list(function)
            if (function[:, -1] == outputs).all():
                linearly_separable_functions = linearly_separable_functions + 1
            else:
                pass

        return linearly_separable_functions

    def update_thresholds(self, pattern, output, eta):
        error = (pattern[-1] - output)
        d_theta = -eta * error
        self.threshold = self.threshold + d_theta

    def get_thresholds(self):
        return self.threshold

    def get_weights(self):
        return self.weights

    def train_network(self, pattern, eta, epochs):
        for i in range(0, epochs):
            for loc_input in pattern:
                output = self.evaluate_perceptron(loc_input)
                self.update_weights(loc_input, output, eta)
                self.update_thresholds(loc_input, output, eta)
```

```python
import matplotlib.pyplot as plt
import numpy as np
from patterns import *
# Patterns
NUMBER_OF_NEURONS = 160
training_patterns = [x1, x2, x3, x4, x5]
question_patterns = [q1, q2, q3]

def hebb_weight_matrix(patterns):
    patterns = format_patterns(patterns)
    weight_matrix = np.matmul(patterns.transpose(), patterns) * 1/NUMBER_OF_NEURONS
    np.fill_diagonal(weight_matrix, 0)
    return weight_matrix


def format_patterns(patterns):
    pattern_index = 0
    for pattern in patterns:
        patterns[pattern_index] = np.array(pattern).flatten()
        pattern_index = pattern_index+1
    patterns = np.array(patterns)
    return patterns


def calculate_next_state(weight_matrix, state_t):
    state_t = np.array(state_t).flatten()
    state_t = state_t.transpose()
    b_t = np.matmul(weight_matrix, state_t)
    b_t = np.where(b_t == 0, 1, b_t)
    state_t_plus_1 = np.sign(b_t)
    return state_t_plus_1


def plot_state(state):
    state = state.reshape((16,10))
    plt.imshow(state, cmap='binary')
    plt.show()


def type_writer_update(weight_matrix, state):
    number_of_rows = len(state)
    new_state = np.array(state).reshape((16, 10))
    #plot_state(new_state)
    for i in range(number_of_rows):
        temporary_state = calculate_next_state(weight_matrix, new_state)
        temporary_state = temporary_state.reshape((16, 10))
        new_state[i] = temporary_state[i]
    #plot_state(new_state)
    return new_state


# Weight matrix
weight_matrix = hebb_weight_matrix(training_patterns)
# Update states
results = []

for i, pattern in enumerate(question_patterns):
    results.append(type_writer_update(weight_matrix, pattern))

# Print resulting patterns
for result in results:
    print(result.tolist())

# Plot states
for result in results:
    plot_state(result)
```

```
x1=[[ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
    [ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],
     [ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],
    [ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],
     [ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],
    [ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],
     [ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],
    [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]];

x2=[[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
    [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
     [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
    [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
     [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
    [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
     [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
    [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1]];

x3=[[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],
    [ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],
    [ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],
    [ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],
    [ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],
    [ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],
    [ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],
    [ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1]];

x4=[[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, 1, -1],
    [ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
     [ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
    [ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],
     [ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
    [ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
     [ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
    [ -1, -1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1]];

x5=[[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],
    [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],
     [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],
    [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, 1, 1, 1, 1, 1, 1, -1],
     [ -1, 1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],
    [ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],
     [ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],
    [ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1]];

q1 = [[-1, -1, -1, -1, -1, -1, -1, -1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, -1, -1],
      [-1, -1, -1, -1, -1, 1, 1, 1, -1, -1], [-1, -1, -1, -1, -1, 1, 1, 1, -1, -1],
      [-1, -1, -1, -1, -1, 1, 1, 1, -1, -1], [-1, -1, -1, -1, -1, 1, 1, 1, -1, -1],
      [-1, -1, -1, -1, -1, 1, 1, 1, -1, -1], [1, 1, 1, 1, 1, 1, 1, 1, -1, -1],
      [1, 1, 1, 1, 1, 1, 1, 1, -1, -1], [1, 1, 1, -1, -1, -1, -1, -1, -1, -1],
      [1, 1, 1, -1, -1, -1, -1, -1, -1, -1], [1, 1, 1, -1, -1, -1, -1, -1, -1, -1],
      [1, 1, 1, -1, -1, -1, -1, -1, -1, -1], [1, 1, 1, -1, -1, -1, -1, -1, -1, -1],
      [1, 1, 1, 1, 1, 1, 1, 1, -1, -1], [1, 1, 1, 1, 1, 1, 1, 1, -1, -1]];

q2 = [[-1, -1, -1, 1, 1, 1, 1, -1, -1, -1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
      [-1, -1, -1, 1, 1, 1, 1, 1, -1, -1], [-1, -1, -1, 1, -1, 1, 1, 1, -1, -1],
      [-1, -1, -1, 1, -1, 1, 1, 1, -1, -1], [-1, -1, -1, 1, -1, 1, 1, 1, -1, -1],
      [-1, -1, -1, 1, -1, 1, 1, 1, -1, -1], [-1, 1, 1, 1, 1, 1, 1, 1, -1, -1],
      [-1, 1, 1, 1, 1, 1, 1, 1, -1, -1], [-1, 1, 1, 1, -1, -1, 1, -1, -1, -1],
      [-1, 1, 1, 1, -1, -1, 1, -1, -1, -1], [-1, 1, 1, 1, -1, -1, 1, -1, -1, -1],
      [-1, 1, 1, 1, -1, -1, 1, -1, -1, -1], [-1, -1, 1, 1, 1, 1, 1, -1, -1, -1],
      [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1]];

q3 = [[1, -1, 1, 1, 1, -1, 1, 1, -1, -1], [1, -1, 1, 1, 1, -1, 1, 1, 1, -1],
      [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1],
      [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1],
      [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [1, -1, 1, 1, 1, -1, 1, 1, -1, -1],
      [1, -1, 1, 1, 1, -1, 1, 1, -1, -1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1],
      [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1],
      [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1],
      [1, -1, 1, 1, 1, -1, 1,
```