Homework1: Boolean functions

A perceptron with a single output neuron can be used to determine whether an *n*-dimensional Boolean function is linearly separable. In this task, you implement a computer program that determines whether a Boolean function is linearly separable or not, using a perceptron with activation function $g(b) = \mathrm{sgn}(b)$, where $b = \sum_{j=1}^{n} w_j x_j - \theta$ [See Eq. (5.9) in the course book for the case *n*=2]. Using this program, sample an *n*-dimensional Boolean function randomly and determine whether it is linearly separable. Do this 10^4 times for *n*=2,3,4 and 5 dimensions, and save the number of linearly separable Boolean functions found. Be sure to not count the same Boolean function twice. This can, for example, be done by adding every sampled Boolean function to a list and excluding the function if it comes up a second time.

The learning rules for the weights w_j and threshold θ are

$$\delta w_j^{(\mu)} = \eta(t^{(\mu)} - O^{(\mu)})x_j^{(\mu)}$$

and

$$\delta\theta^{(\mu)} = -\eta(t^{(\mu)} - O^{(\mu)})$$

See also Eq. (5.18) in the course book. For each Boolean function, train the perceptron for 20 training epochs (one epoch amounts to updating the parameters once for every input-output pair) using a learning rate η=0.05. Initialize the weights from a normal distribution with mean zero and variance 1/*n*, and initialize the thresholds to zero.

Task description

for trail = 1:10^4

- Sample Boolean

- Train percepton

- Check accuracy

- if 100% correct, counter+=1

- else, pass

shape of weights

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

number of Boolean functions

$$2^{2^N}$$

number of outputs

2^N

```
clc
clear
n = 2; % N binary inputs 3,4,5
nTrials = 10e4;     % number of trails
nEpoches = 20;      % number of epoches
eta = 0.05;         % learning rate
counter = 0;        % number of linearly separable Boolean found
```

Setup boolean inputs and used bool list

```
boolean_inputs_2 = ff2n(2)';
boolean_inputs_3 = ff2n(3)';
boolean_inputs_4 = ff2n(4)';
boolean_inputs_5 = ff2n(5)';
```

```
n = 2;
boolean_inputs = boolean_inputs_2;
separable_counter_2D = learning_iter(n, nTrials, nEpoches, eta, boolean_inputs);
disp(separable_counter_2D)
```

```
    14
```

```
n = 3;
boolean_inputs = boolean_inputs_3;
separable_counter_3D = learning_iter(n, nTrials, nEpoches, eta, boolean_inputs);
disp(separable_counter_3D)
```

```
   104
```

```
n = 4;
boolean_inputs = boolean_inputs_4;
separable_counter_4D = learning_iter(n, nTrials, nEpoches, eta, boolean_inputs);
disp(separable_counter_4D)
```

```
        1304
```

```
n = 5;
boolean_inputs = boolean_inputs_5;
separable_counter_5D = learning_iter(n, nTrials, nEpoches, eta, boolean_inputs);
disp(separable_counter_5D)
```

```
     2
```

```
function separable_counter = learning_iter(n, nTrials, nEpoches, eta, boolean_inputs)
```

```
    used_bool=[];
    separable_counter = 0;
    for trail = 1:nTrials
        boolean_outputs = 2 * randi([0, 1], 2^n, 1) - 1;
        if isempty(used_bool) || ~ismember(boolean_outputs',used_bool','rows')
            w = normrnd(0, 1/n, [n,1]);       % initialize weight with normal distribution
            theta = 0;                      % initialize theta with 0
            for epoch = 1:nEpoches
                total_error = 0;
                for mu = 1:2^n              % 1 epoch is updating the parameters once for every
                    Out = sign(dot(w,boolean_inputs(:,mu))-theta);
                    if Out == 0
                        Out = 1;
                    end
                    err = boolean_outputs(mu) - Out;
                    % delta_w, delta_theta calculation
                    delta_w = cal_delta_w(n, eta, err, boolean_inputs(:,mu));
                    delta_theta = -eta * err;
                    % update w and theta
                    w = w + delta_w;
                    theta = theta + delta_theta;

                    total_error = total_error + abs(err);
                end
                if total_error == 0
                    separable_counter = separable_counter + 1;   % find one linear sepration fu
                    break
                end
            end
            used_bool(:,end+1) = boolean_outputs;
        end
    end
end

function delta_w = cal_delta_w(n, eta, err, input)
    delta_w = zeros([n,1]);
    for j=1:n
        delta_w(j) = eta * err * input(j);
    end
end
```