# Assignment 2:Factors of a neural network

Dekel Viner(S1234567),Luis Knigge(S2560224)

September 3, 2019

## 1 Linear Algebra

## 2 Theory Questions

1. What is an action potential and how is it generated in a biologic neuron?
   an action potential is a short lasting event in which the membrane potential of a neuron rapidly spikes and falls.

   An initial depolarizing current open the fast sodium channels, this initiates an inward sodium current that via depolarization opens more sodium channels. The overall depolarization opens the slower potassium channels. These allow potassium ions to leave the sell causing a hyperpolarization. The sodiums channels also close after the initial opening and an overall depolarization brings the cell to equilibrium. And also enter a refractory period where the cell is hyperpolarized for some time in which the cell cannot be exited.

2. What is hyperpolarization?
   Hyperpolarization is the opposite of depolarization, it is a change in the membrane potential that makes it more negative.

3. What is a PSP and how is a PSP represented in an artificial neuron? PSP stands for the post-synaptic potential. In an artificial neuron the PSP is represented as the weighted input.

4. Which feature do the step function and the action potential in a biologic neuron have in common? For a step function there is a required threshold that need to be passed inorder for the artificial neuron to return an output of 1, and in a biologic neuron the action potential also needs to surpass a certain V inorder for the neuron to fire.

# 3 TLU on paper

# 4 Experimenting with a TLU

- a)
  In some cases an adjustment to the weights is not enough to change the output in which case the error remains the same.

- b)
  Without squaring the errors positive and negative errors would cancel each other out. Squaring the errors removes having negative errors.

- c)
  It depends on the initial values of the weights and the threshold. if they are far from a position where the error would be 0 the low learning speed will prevent a fast moving to that position.

- d)
  In some cases the network is learning really fast which seems to be better then the original model, but in most of the cases the learning rate makes the weights overshoot their good values in which case the error jumps up instead of falling down. For major adjustments it seems that a high leaning speed can be useful but for fine adjustments the learning rate should be decreased.

- e)
  In certain cases the input of 0.9 or 0.1 multiplied by a certain weight would cause that a weight line would be above a the threshold line and weighted input would be below the threshold. As you can see in the graph with inputs of .2 and .8 (stronger effect) the weight 1 is highe then the threshold which should not happen in the model with inputs of 1 and 0.

- f)
  The feature which we took out of the equation in e) is that the artificial neurons should have a binary in- and output. In this case that is not given anymore which results into unexpected behaviour.

- g)
  (see hand writen paper)

# 5 XOR-rule

The XOR-rule can't be learned by the model. Even though the weights converge to value the error function shows that the model cannot be right. The error always converges to 4 which means that all examples are wrongly categorised.

# 6 Code

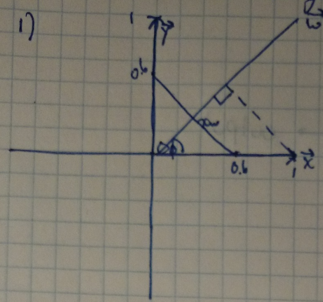
tluAND.m

```
% TLU implementation
% Put your names here

% Dekel Viner    S2612925
% Luis Knigge    S2560224
clear all;

% Parameters
learn_rate = 0.1;    % the learning rate
n_epochs = 30;       % the number of epochs we want to train

% Define the inputs
examples = [0,0;0,1;1,0;1,1];

% Define the corresponding target outputs
goal = [0;0;0;1];

% Initialize the weights and the threshold
weights = [rand rand];
threshold = rand;

% Preallocate vectors for efficiency. They are used to log your
    data
log_error = zeros(n_epochs,1);
log_weights = zeros(n_epochs,2);
log_threshold = zeros(n_epochs,1);

% Store number of examples and number of inputs per example
n_examples = size(examples,1);     % number of exapmles
n_inputs = size(examples,2);       % number of inputs

for epoch = 1:n_epochs
    epoch_error = zeros(n_examples,1);

    log_weights(epoch,:) = weights;
    log_threshold(epoch) = threshold;

    for pattern = 1:n_examples

        % Initialize weighted sum of inputs
        summed_input = examples(pattern,1) * weights(1) + examples
            (pattern,2) * weights(2);

        % Subtract threshold from weighted sum
        summed_input = summed_input - threshold;
```

```

        % Compute output
        if summed_input < 0
            output = 0;
        else
            output = 1;
        end

        % Compute error
        error = goal(pattern)-output;

        % Compute delta rule
        delta_weights = [learn_rate*error*examples(pattern, 1), learn_rate*error*examples(pattern, 2)];
        delta_threshold = learn_rate*error*(-1);

        % Update weights and threshold
        weights = weights + delta_weights;
        threshold = threshold + delta_threshold;

        % Store squared error
        epoch_error(pattern) = error.^2;
    end

    log_error(epoch) = sum(epoch_error);
end

% Plot functions
figure;
plot(log_error)
title('TLU-error over epochs');
xlabel('# of epochs')
ylabel('Summed Squared Error')

figure;
plot(1:n_epochs,log_weights(:,1),'r-','DisplayName','weight 1')
hold on
plot(1:n_epochs,log_weights(:,2),'b-','DisplayName','weight 2')
plot(1:n_epochs,log_threshold,'k-','DisplayName','threshold')
axis([1 n_epochs -1 1]);
legend('location','NorthEast')
```

## 3.1 Linear Algebra

1)




3) $||x|| = \sqrt{1^2+0^2} = 1$

4) $\vec{x} \cdot \vec{y} = 1 \cdot 0 + 0 \cdot 1 = 0$

5) $\vec{x} \cdot \vec{y} = ||x|| \cdot ||y|| \cdot \cos\theta$ as $\cos(90) = 0$
$\vec{x} \cdot \vec{y}$ will also equal 0.

8) The cosine rule is defined as $\cos\phi = \frac{adj}{hyp}$ for 90° angle triangles therefor with $||x||$ as the hypotenuse and $x_w$ as the adjecent $\cos\phi = \frac{x_w}{||x||}$.

9) $\vec{x} \cdot \vec{w} = ||x|| \, ||w|| \cos\phi$ and $\cos\phi = \frac{x_w}{||x||}$ so

$= \vec{x} \cdot \vec{w} = ||x|| \, ||w|| \, \frac{x_w}{||x||}$

$= \vec{x} \cdot \vec{w} = x_w \, ||w||$

10) $(\vec{a} \cdot \vec{w}) - \theta > 0$ so $\vec{a} \cdot \vec{w} > \theta$

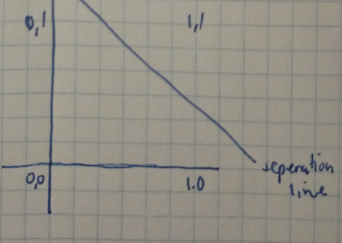
$a_1 w_1 + a_2 w_2 > \theta$ for $\vec{w} = \binom{1}{1}$

$a_1 + a_2 > \theta$ $\quad \theta = 0.6$

$a_1 + a_2 > 0.6$

5.9

changing the goal to NAND works aswell, However in this case bothe the threshold and the weights are negative. The reason for that being that when none of the nodes are activated or only one of them is activated they should still be above the threshold.

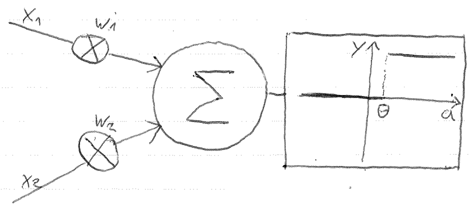



In this case everything above the seperation line needs to have an output of 0 and every thing below needs an output of 1. so the outputs have swapped accross the line, so which is why the weights and threshold became negative.

Luis Knigge S2560224
Dekel Viner S2612925

# Neural networks
## Assignment 2

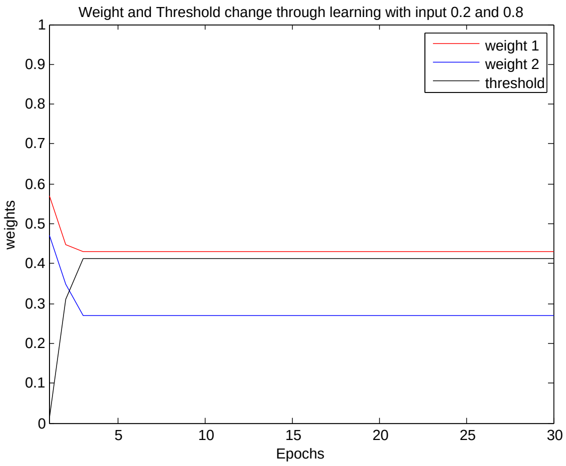
Q. 3.3
a)



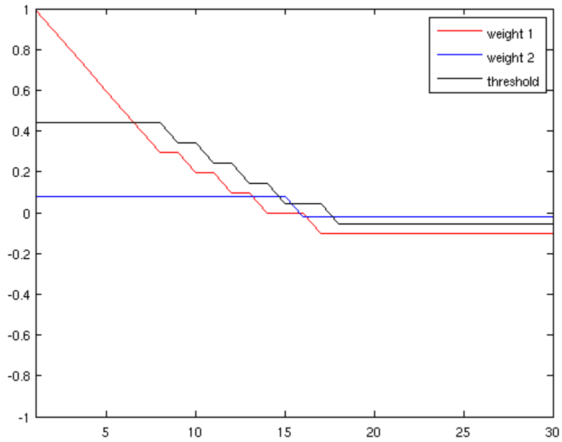

$\theta$ - threshold
$w_1, w_2$ - weights
a - activation

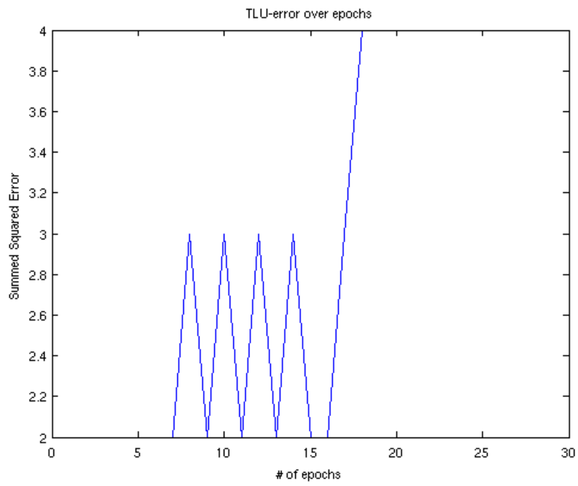
b) The activation is the combined voltage on the membrane of the neuron if it reaches a certain threshold $\theta$ which is "calculated (add) in the axon hillock the neuron fires itself.
The weights are ~~the~~ a measure of how strongly the pre-synaptic neuron is connected to the post-synaptic neuron.

c) The ~~function~~ class would have the variables input weights (vector), summed-input, threshold as well as a ~~a~~ ~~input v~~ binary input vector and a ~~out pu~~ binary output variable. Methods would be "summation" to weight the inputs and add them together and "compare" to calculate the appropriate output to the summed-input by comparing it to the threshold. The summed-input represents the activation.



Weight and Threshold change through learning with input 0.2 and 0.8
weight 1
weight 2
threshold
weights
Epochs
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
5
10
15
20
25
30


weight 1
weight 2
threshold


TLU-error over epochs
Summed Squared Error
# of epochs