Lab1 report

Threshold Logical Unit

Eduardo Delgado Coloma Bier (s3065979)
Mikel Orbea Sotil (s3075001)

February 17, 2016

1 Linear Algebra

1. **Coordinate System**

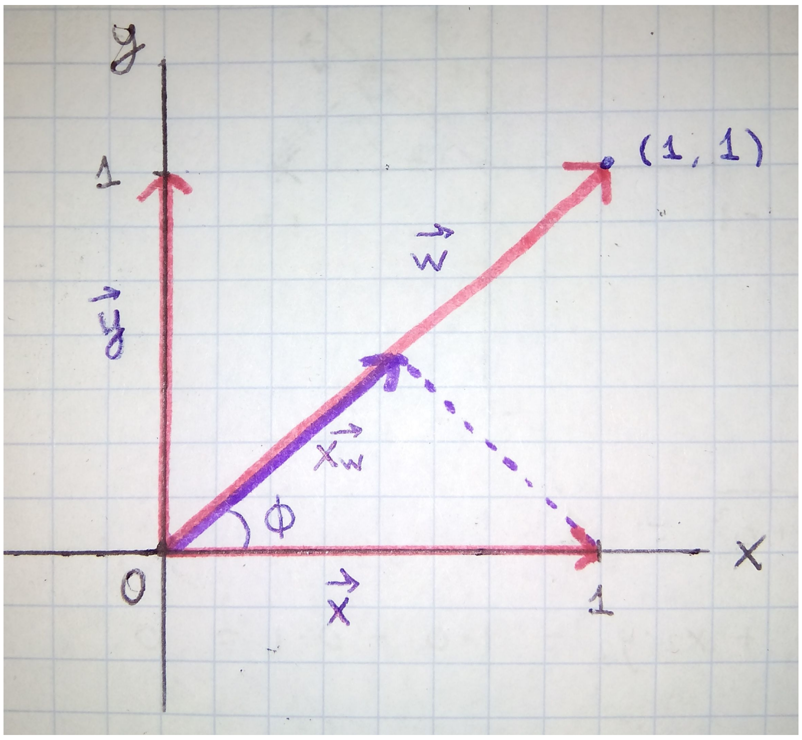

Figure 1: Coordinate system resulting from the exercises

3. **Length of x = ||x||**
   Given x = $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$:

$$\begin{aligned} ||x|| &= \sqrt{1^2 + 0^2} \\ ||x|| &= 1 \end{aligned} \tag{1}$$

4. **Inner product of x^T y = x * y**
   Given x = $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and y = $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$:

$$\begin{aligned} \vec{x} * \vec{y} &= x_1 * y_1 + x_2 * y_2 \\ \vec{x} * \vec{y} &= 1 * 0 + 0 * 1 \\ \vec{x} * \vec{y} &= 0 \end{aligned} \tag{2}$$

5. **Inner product of x^T y = x * y from geometry**
   We can check our answer for the previous question by verifying that the vectors x and y are perpendicular in the picture. Whenever two vectors are perpendicular, their inner product is 0.

8. **Demonstration** $\cos\Phi = \frac{x_w}{||x||}$

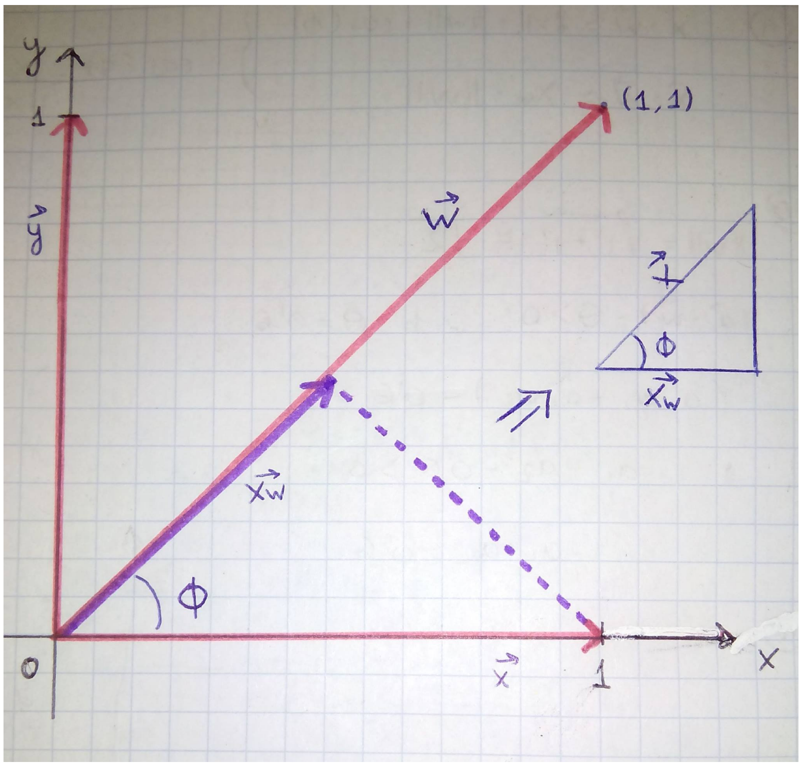

Figure 2: Triangle formed by $\vec{x}$ and $\vec{x_w}$

As it can be seen in the coordinate system, $\vec{x_w}$ and $\vec{x}$ form an angled triangle given the perpendicular line between $\vec{w}$ and $\vec{x}$.
Giving the fact that:

$$\cos\Phi = \frac{adjacent}{hypotenuse} \tag{3}$$

We conclude that

$$\begin{gathered} adjacent = x_w \\ hypotenuse = ||x|| \end{gathered} \tag{4}$$

So

$$\cos\Phi = \frac{x_w}{||x||} \tag{5}$$

9. **Demonstration** $\vec{x} * \vec{w} = x_w * ||w||$
   Given:

$$\cos(\Phi) = \frac{x_w}{||x||} \tag{6}$$

$$\begin{gathered}\vec{x} * \vec{w} = ||x|| * ||w|| * \cos(\Phi) \Leftrightarrow \vec{x} * \vec{w} = ||x|| * ||w|| * \frac{x_w}{||x||} \\ \vec{x} * \vec{w} = ||x|| * ||w|| * \frac{x_w}{||x||} \Leftrightarrow \vec{x} * \vec{w} = \frac{||x||}{||x||} * ||w|| * x_w \\ \vec{x} * \vec{w} = \frac{||x||}{||x||} * ||w|| * x_w \Leftrightarrow \vec{x} * \vec{w} = x_w * ||w||\end{gathered} \tag{7}$$

10. **Length of w** $=$ **||w||**
    Given w $= \begin{pmatrix} 1 \\ 1 \end{pmatrix}$:

$$\begin{gathered}||w|| = \sqrt{1^2 + 1^2} \\ ||w|| = \sqrt{2}\end{gathered} \tag{8}$$

   Given $\theta = 0.6$ and w $= \begin{pmatrix} 1 \\ 1 \end{pmatrix}$:

$$\begin{gathered}(\vec{a} * \vec{w}) - \theta > 0 \\ (a_1 * w_1 + a_2 * w_2) - 0.6 > 0 \\ a_1 + a_2 - 0.6 > 0 \\ a_1 + a_2 > 0.6\end{gathered} \tag{9}$$

   Therefore, **a** satisfies $\vec{a} \cdot \vec{w} - \theta > 0$ when $a_1 + a_2 > 0.6$

11. **Plane split in two by w and the threshold** θ

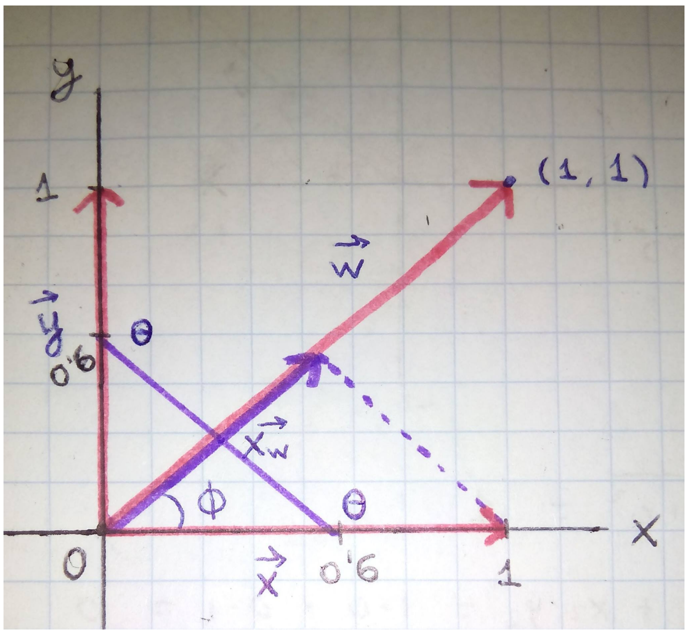

Figure 3: Coordinate system resulting from the last exercise

2 Theory questions

1. **What is an action potential and how is it generated in a biologic neuron?**

   An action potential, also known as nerve impulses or spikes, is the event when the electrial membrane potential of a neuron rises and falls. It is used by neurons to communicate with the others.

   When the membrane potential of a biological neuron is increased to a specific threshold, the ion channels of the neuron begin to open so an action potential is allowed to travel down the axon.

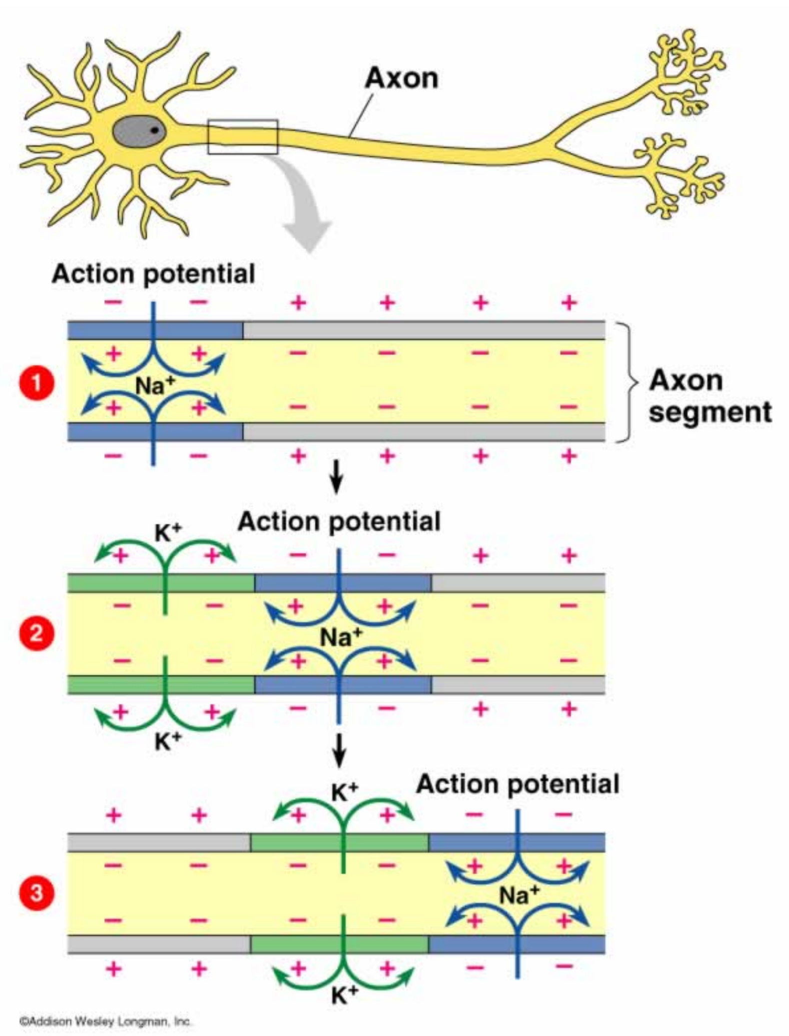

Figure 4: Transmission procedure of action potential

2. **What is hyperpolarization?**

   Hyperpolarization is produced when the membrane potential of a neuron becomes more negative. This makes it more difficult for the threshold to be reached because the stimulus required to move the membrane potential is higher.

   In a neuron, hyperpolarization appears after the generation of an action potential. While the neuron it is hyperpolarized, it can not generate the next action potential so the neuron needs time to recover.

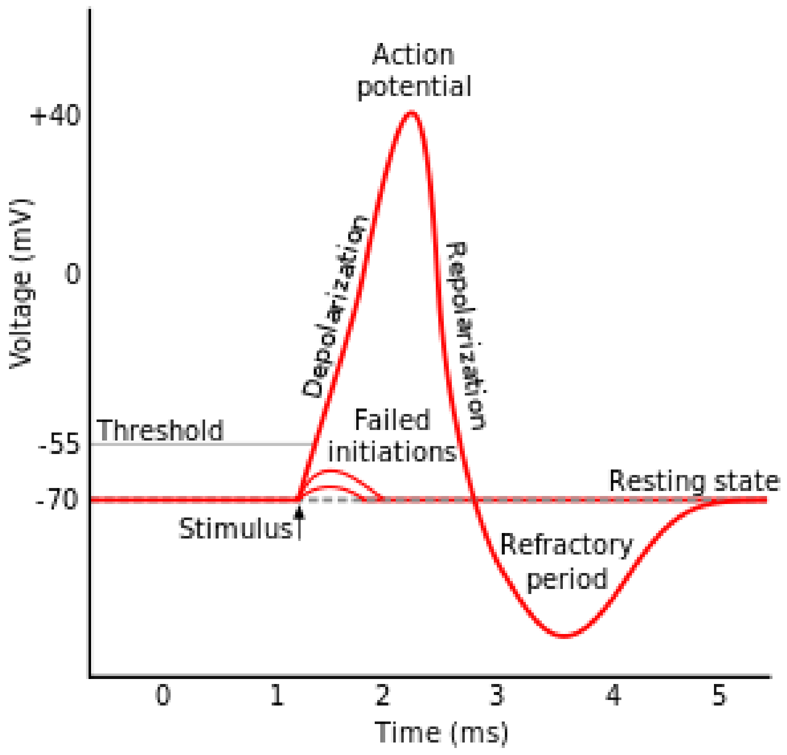

Figure 5: Action potential diagram

As it can be seen in the diagram, after the depolarization (threshold is reached and an action potential is sent), the neuron enters in refractory period where it is hyperpolarized. Once the neuron is recovered, it is able to send the next action potential.

3. **What is a PSP and how is a PSP represented in an artificial neuron?**

PSP (Postsynaptic potentials) are changes in the membrane potential of a neuron. Their function is to initiate or inhibit action potentials.

If the membrane is gaining positive charge (depolarization) it is an excitatory postsynaptic potential (EPSP). Otherwise, if the membrane is gaining negative charge (hyperpolarization) it is an inhibitory postsynaptic potential (IPSP).

In an artificial neuron, PSPs are represented as weighted inputs.

4. **Which feature do the step function and the action potential in a biologic neuron have in common?**

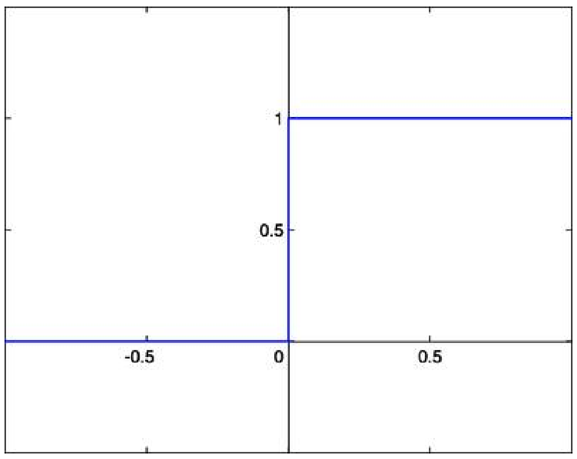

Figure 6: Step function

The step function can be interpreted as an action potential cycle. In the step function when the maximum is reached, it stops increasing and then drops. This behavior mimics the one of the action potential in a biologic neuron.

3 A TLU on paper

1. **Simple artificial neuron**

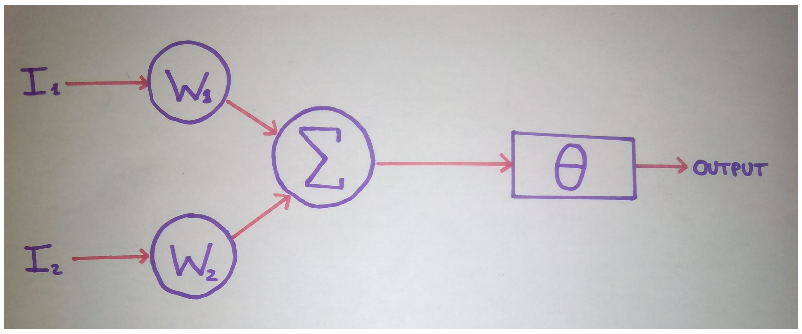

Figure 7: Simple artificial neuron diagram

$$I_1 = Input\ 1$$
$$w_1 = Weight\ 1$$

$$I_2 = Input\ 2$$
$$w_2 = Weight\ 2$$

$$\sum = \sum_{i=1}^{2} w_i * i_i \tag{10}$$

$$\theta = threshold$$

$$OUTPUT = \begin{cases} 0, & \text{if } \sum < \theta \\ 1, & \text{if } \sum \geq \theta \end{cases}$$

2. **Comparing with a biologic neuron**

   The weights correspond to how strong a connection is between this neuron and the previous neurons, while the sum function is it done by the cell body. The threshold is the potential needed to send a nerve impulse through the axon.

3. **Class Neuron**

```

class Neuron () {

  float threshold;
  weights[];

  /**
  * Teach the neuron how to compute correct values for a function
  */
  void learn(float inputs[], float outputs[], float learning_rate, int n_epochs)

  /**
  * Compute the result of the function with the given inputs.
  * If you want the correct ouput, you have to run learn(...) one time before
    calling this method.
  */
  float compute(float inputs[])
}
```
Listing 1: Neuron Class

4 A TLU in Matlab

1. **AND-rule**

```matlab
% TLU implementation
% Eduardo Delgado Coloma Bier
% Mikel Orbea Sotil

% Parameters
learn_rate = 0.1;     % the learning rate
n_epochs = 100;       % 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 = [1 1 1 0];

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

% Preallocate vectors for efficiency. They are used to log your data
% The 'h' is for history
h_error = zeros(n_epochs,1);
h_weights = zeros(n_epochs,2);
h_threshold = zeros(n_epochs,1);

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

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

    h_weights(epoch,:) = weights;
    h_threshold(epoch) = threshold;

    for pattern = 1:n_examples

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

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

        % Compute output
        if threshold_value >= 0
            output = 1;
        else
            output = 0;
        end

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

        % Compute delta rule
        delta_weights = learn_rate * error * examples(pattern,:);
        delta_threshold = - learn_rate * error;

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

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

    h_error(epoch) = sum(epoch_error);
end
% Plot functions
figure(1);
plot(h_error)
title('\textbf{TLU-error over epochs}', 'interpreter', 'latex', 'fontsize', 12);
xlabel('\# of epochs', 'interpreter', 'latex', 'fontsize', 12)
ylabel('Summed Squared Error', 'interpreter', 'latex', 'fontsize', 12)

figure(2);
plot(1:n_epochs,h_weights(:,1),'r-','DisplayName','weight 1')
hold on
plot(1:n_epochs,h_weights(:,2),'b-','DisplayName','weight 2')
plot(1:n_epochs,h_threshold,'k-','DisplayName','threshold')
xlabel('\# of epochs', 'interpreter', 'latex', 'fontsize', 12)
title('\textbf{Weight vector and threshold vs epochs}', 'interpreter', 'latex', 'fontsize', 12);
h = legend('location','NorthEast');
set(h, 'interpreter', 'latex', 'fontsize', 12);
hold off
```

Listing 2: AND.m

2. XOR-rule

```
% TLU implementation
% Eduardo Delgado Coloma Bier
% Mikel Orbea Sotil

% Parameters
learn_rate = 0.1;    % the learning rate
n_epochs = 100;      % 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 1 1 0];

% Initialize the weights and the threshold
```

```
weights = [rand rand];
threshold = rand;

% Preallocate vectors for efficiency. They are used to log your data
% The 'h' is for history
h_error = zeros(n_epochs,1);
h_weights = zeros(n_epochs,2);
h_threshold = zeros(n_epochs,1);

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

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

    h_weights(epoch,:) = weights;
    h_threshold(epoch) = threshold;

    for pattern = 1:n_examples

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

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

        % Compute output
        if threshold_value >= 0
            output = 1;
        else
            output = 0;
        end

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

        % Compute delta rule
        delta_weights = learn_rate * error * examples(pattern,:);
        delta_threshold = - learn_rate * error;

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

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

    h_error(epoch) = sum(epoch_error);
end
% Plot functions
```

```
figure(1);
plot(h_error)
title('\textbf{TLU-error over epochs}', 'interpreter', 'latex', 'fontsize', 12);
xlabel('\# of epochs', 'interpreter', 'latex', 'fontsize', 12)
ylabel('Summed Squared Error', 'interpreter', 'latex', 'fontsize', 12)

figure(2);
plot(1:n_epochs,h_weights(:,1),'r-','DisplayName','weight 1')
hold on
plot(1:n_epochs,h_weights(:,2),'b-','DisplayName','weight 2')
plot(1:n_epochs,h_threshold,'k-','DisplayName','threshold')
xlabel('\# of epochs', 'interpreter', 'latex', 'fontsize', 12)
title('\textbf{Weight vector and threshold vs epochs}', 'interpreter', 'latex', '
    fontsize', 12);
h = legend('location','NorthEast');
set(h, 'interpreter', 'latex', 'fontsize', 12);
hold off
```

Listing 3: XOR.m

3. Stop learning

```
% TLU implementation
% Eduardo Delgado Coloma Bier
% Mikel Orbea Sotil

% Parameters
learn_rate = 0.1;    % the learning rate
n_epochs = 100;      % 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
% The 'h' is for history
h_error = zeros(n_epochs,1);
h_weights = zeros(n_epochs,2);
h_threshold = zeros(n_epochs,1);

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

epoch = 1;
stop = 1;
while (epoch <= n_epochs && stop == 1)
    stop = 1;
    epoch_error = zeros(n_examples,1);
```

```matlab

    h_weights(epoch,:) = weights;
    h_threshold(epoch) = threshold;

    for pattern = 1:n_examples

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

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

        % Compute output
        if threshold_value >= 0
            output = 1;
        else
            output = 0;
        end

        % Compute error
        error = (goal(pattern) - output);
        if (error ~= 0)
            stop = 0;
        end


        % Compute delta rule
        delta_weights = learn_rate * error * examples(pattern,:);
        delta_threshold = - learn_rate * error;

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

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

    h_error(epoch) = sum(epoch_error);
    epoch = epoch + 1;
end

n_epochs = epoch;

% Plot functions
figure(1);
plot(h_error(1:n_epochs))
title('\textbf{TLU-error over epochs}', 'interpreter', 'latex', 'fontsize', 12);
xlabel('\# of epochs', 'interpreter', 'latex', 'fontsize', 12)
ylabel('Summed Squared Error', 'interpreter', 'latex', 'fontsize', 12)

figure(2);
```

```
plot(1:n_epochs,h_weights(1:n_epochs,1),'r-','DisplayName','weight 1')
hold on
plot(1:n_epochs,h_weights(1:n_epochs,2),'b-','DisplayName','weight 2')
plot(1:n_epochs,h_threshold(1:n_epochs),'k-','DisplayName','threshold')
xlabel('\# of epochs', 'interpreter', 'latex', 'fontsize', 12)
title('\textbf{Weight vector and threshold vs epochs}', 'interpreter', 'latex', '
    fontsize', 12);
h = legend('location','NorthEast');
set(h, 'interpreter', 'latex', 'fontsize', 12);
hold off
```

Listing 4: STOP.m

5 Experimenting with a TLU

a) **The error does not decrease each epoch. Why?**
The error does not decrease each epoch because sometimes the weights and threshold work for a certain input, although they do not for all inputs. When that happens, the TLU does not adjust the weights and the threshold, not decreasing the weight.

b) **Why are we interested in the summed squared error $\sum(t(p) - y(p))^2$ instead of simply summing N the errors $\sum(t(p) - y(p))$?**
If we were to simply sum the errors, in the case where t < y (t = 0, y = 1), we'd actually be lowering the total error. This would mean that t = 0 y = 1 would be less wrong then the case t = 1, y = 0, which is clearly not what we want, as they are equally wrong. By squaring the difference we eliminate that problem, as we would only be summing positive numbers and the error for those cases would be the same.

c) **Why is the number of epochs required to reach an error of 0 not always the same?**

The number of epochs to reach an error of 0 is not always the same because the weight vector and the threshold are both randomly picked. This means that we can be either very close to the solution or very far from it, which may take a greater number of epochs.

d) **Increase the learning rate from 0.1 to 0.6. What do you observe? Is a higher learning rate better? Explain you answer.**
When changing the learning rate from 0.1 to 0.6 it's possible to see that the number of epochs required to reach an error of 0 is usually greater than before. This happens because when we adjust the weight vector with a larger learning rate, we take larger "steps", which can be too much for a certain situation. That way, we end up having to compensate for these bigger steps and take longer to reach the solution.

e) **The input is 0 or 1. Change this to 0.1 or 0.9. Is the TLU still capable of learning the AND- function? What happens if the input is set to 0.2 or 0.8?**
In both cases, the TLU is still able to learn the AND rule.

f) **Which important feature of artificial neural networks did we encounter in (e)?**
The previous answer indicates that an artificial neural network are capable of learning regardless of the value of the input. This means that as long as the input and output make sense within a certain logic, the neural network will be able to learn the desired function.

g) **Change the vector goal such that the TLU learns a NAND-function (NOT-AND). Does this work too? What happens to the weight values? Explain why the threshold has become negative by drawing a geometrical sketch.**
The TLU is able to learn the NAND function. Both the weight values and the threshold become negative because the weight vector always points to the region where y = 1. The separating line for the NAND and AND function is the same, but the regions are swaped. This means that since the threshold and weights were positive for the AND function and since the weight vector needs to point to the opposite side now, they have to become negative.

6 XOR-rule

Since the error never reaches zero, the XOR rule cannot be learned by the TLU. The reason for that is the fact that there doesn't exist a separating line for the XOR inputs. As so, the TLU is unable to separate the inputs and can't learn the XOR function.