TASK A

Exercise 1: Determine the minimum number of epochs necessary to train the model in order to obtain only valid predictions.

After 4 epochs. So we need to train minimum of 4 epochs. Afterwards, our model was able to correctly predict the out come of an AND gate.

Exercise 2: Modify the program in order to perform the prediction for an OR gate.

```
[ ] # And
    t_and = np.array([0, 0, 0, 1])

    #Or
    t_or = np.array([0, 1, 1, 1])

    def main(t):

        #Application 1 - Train a single neuron perceptron in order to predict the output of an AND gate.
        #The network should receive as input two values (0 or 1) and should predict the target output

        #Input data
        P = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

        #Labels
        #t = np.array([0, 0, 0, 1])

        #TODO - Application 1 - Step 2 - Initialize the weights with zero  (weights)
        weights = np.array([0,0])

        #TODO - Application 1 - Step 2 - Initialize the bias with zero  (bias)
        bias = 0

        #TODO - Application 1 - Step 3 - Set the number of training steps  (epochs)
        epochs = 30

        #TODO - Application 1 - Step 4 - Perform the neuron training for multiple epochs
        for ep in range(epochs):
            for i in range(len(t)):

                #TODO - Application 1 - Step 4 - Call the forwardPropagation method
                a = forwardPropagation(P[i], weights, bias)

                #TODO - Application 5 - Compute the prediction error (error)
                error = t[i] - a

                #TODO - Application 6 - Update the weights
                weights[0] = weights[0] + error * P[i][0]
                weights[1] = weights[1] + error * P[i][1]
                #TODO - Update the bias
                bias = bias + error
```

```
                #TO_DELETE
                #continue
        #TODO - Application 1 - Print weights and bias
                #print("weights: ", weights, "bias", bias)
        #TODO - Application 1 - Step 7 - Display the results
                print("epochs: ", ep, "Input: ", P[i], "Predict: ", a, "groundTruth: ", t[i])
        return
```

```
[ ] if __name__ == "__main__":
        main(t_or)
```

```
epochs:  0 Input:  [0 0] Predict:  0 groundTruth:  0
epochs:  0 Input:  [0 1] Predict:  0 groundTruth:  1
epochs:  0 Input:  [1 0] Predict:  1 groundTruth:  1
epochs:  0 Input:  [1 1] Predict:  1 groundTruth:  1
epochs:  1 Input:  [0 0] Predict:  1 groundTruth:  0
epochs:  1 Input:  [0 1] Predict:  1 groundTruth:  1
epochs:  1 Input:  [1 0] Predict:  0 groundTruth:  1
epochs:  1 Input:  [1 1] Predict:  1 groundTruth:  1
epochs:  2 Input:  [0 0] Predict:  1 groundTruth:  0
epochs:  2 Input:  [0 1] Predict:  1 groundTruth:  1
epochs:  2 Input:  [1 0] Predict:  1 groundTruth:  1
epochs:  2 Input:  [1 1] Predict:  1 groundTruth:  1
epochs:  3 Input:  [0 0] Predict:  0 groundTruth:  0
epochs:  3 Input:  [0 1] Predict:  1 groundTruth:  1
epochs:  3 Input:  [1 0] Predict:  1 groundTruth:  1
epochs:  3 Input:  [1 1] Predict:  1 groundTruth:  1
epochs:  4 Input:  [0 0] Predict:  0 groundTruth:  0
epochs:  4 Input:  [0 1] Predict:  1 groundTruth:  1
epochs:  4 Input:  [1 0] Predict:  1 groundTruth:  1
epochs:  4 Input:  [1 1] Predict:  1 groundTruth:  1
epochs:  5 Input:  [0 0] Predict:  0 groundTruth:  0
epochs:  5 Input:  [0 1] Predict:  1 groundTruth:  1
epochs:  5 Input:  [1 0] Predict:  1 groundTruth:  1
```

Exercise 3: Change the neuron activation function from step unit to sigmoid. Are there any differences when performing the prediction?

We have implemented the sigmoid function as follow;

```
def sigmoid(n):
    s=1/(1+np.exp(-n))
    #ds=s*(1-s)
    return int(round(s))
```

However, we did not observe any significant difference in the outcome.

TASK B

Exercise 4: Determine the minimum number of epochs necessary to train the model in order to obtain a prediction error inferior to 0.01. Consider the following error function:

$$C = \frac{1}{2N}\sum_{i=1}^{N}(out - pred)^2$$

where N is the total number of samples applied as input, **out** represents the desired output label, while **pred** is the predicted output.

According to my implementation, the minimum error is reached at the epoch of : 6742 and the error is : 0.009

The error function is defined as follow;

```
def errorFun(list_):
  sum = 0
  for idx in list_:
    e = (idx[0] - idx[1])**2   # label - predict
    sum = sum + e

  C = sum/2*len(list_)
  return round(C, 3)
```

The following snippet help us to append the out and pred and hence to obtain the error.

```
# list to store error values
    list_ = []

# Display the results
    for i in range(len(labels)):
        outL1 = forwardPropagationLayer(points[i], weightsLayer1, biasLayer1)
        outL2 = forwardPropagationLayer(outL1, weightsLayer2, biasLayer2)

        list_.append([labels[i][0], outL2[0][0]])
    #print("Input = {} - Predict = {} - Label = {}".format(points[i], outL2, labels[i]))

    C = errorFun(list_)
    #print(C)
    # convert to string before write
    f.write("%s %s\n" % (str(ep), str(C)))

    if C < 0.01:
      print("The minimum error is reached at the epoch of : ", ep, " and the error is :", C )
```

Exercise 5: Modify the activation function for the neurons form sigmoid to hyperbolic tangent. The hyperbolic tangent is given by the following equation:

$$tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} = \frac{2}{1 + e^{-2u}} - 1$$

In the context of backpropagation the derivative of the $tanh(u)$ is:

$$\frac{dtanh(u)}{du} = (1 + u) * (1 - u)$$

What can be observed about the predicted values? Justify the results.

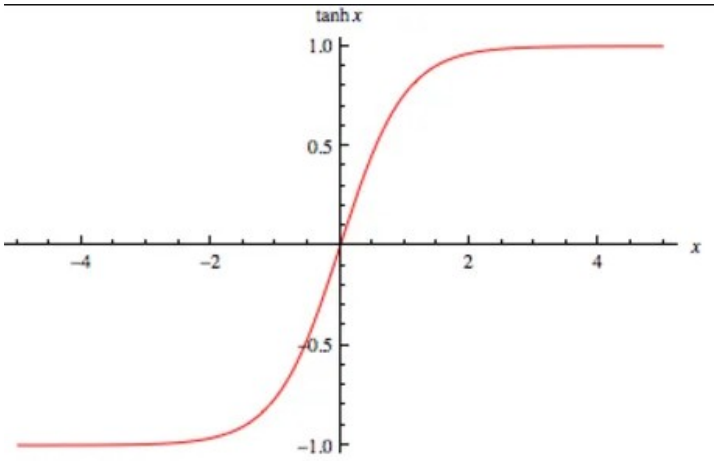
The activation functions and its derivative will be like below;

```
# activation function
def hyperbolicTanget(n):
  t = (np.exp(n)-np.exp(-n))/(np.exp(n)+np.exp(-n))

  return t

# derivative of hyperbolic tanget
def hyperbolicTangetDerivative(n):
  return (1+n)*(1-n)
```

The first point is when we introduce the hyperbolic activation function, the negative output has been observed. That's very normal because the shape of the function will allow to have negative value in the range (co-domain) as follow;



If we observe the **output_hyperBolic.txt** generated by the program reveals that the error of 0.01 (or inferior) is reached at the epoch of 410. Which is far better than the Sigmoid because in Sigmoid the error of 0.01 (or inferior) is reached at the epoch of **6742.** The following figure show the output of Jupyter Notebook cell;

```
if __name__ == "__main__":
    main()
epochs = 410 - Input = [0 1] - Predict = [1] - Label = [1]
epochs = 410 - Input = [1 0] - Predict = [1] - Label = [1]
epochs = 410 - Input = [1 1] - Predict = [0] - Label = [0]
The minimum error is reached at the epoch of :  410  and the error is : 0.009
epochs = 411 - Input = [0 0] - Predict = [0] - Label = [0]
epochs = 411 - Input = [0 1] - Predict = [1] - Label = [1]
epochs = 411 - Input = [1 0] - Predict = [1] - Label = [1]
epochs = 411 - Input = [1 1] - Predict = [0] - Label = [0]
```

Exercise 6: How many trainable-parameters are involved in the ANN architecture existent of a XOR gate.

We need 6 trainable parameters.

Is the ANN performances influenced by the random initialization of the different variables?

Yes it is. It must be initialized by small random numbers. Because its inherent nature of stochastic and as well as for faster convergent.

Exercise 7: By using Python and some dedicated machine learning libraries (*i.e.*, Tensorsflow with Keras API) write a novel script that re-implements the ANN architecture of the AND gate developed at Application 1 (consider as activation function sigmoid). Extend the code in order to predict the output for the XOR gate in Application 2. Both networks will use Adam as optimizer.