Solution ANN

i) Basic level

3. By reading the matrix INPUT and the vector CORRECT_OUTPUT together, we can know that

(0,0)->0; (1,0)->1; (0,1)->1; (1,1)->1. We can see that it is an OR statement.

4.The format of the input is a matrix with each row being one possible input combination (example) and the CORRECT_OUTPUT is where we decide it's expected output.
5. TOTAL_ERROR stores the sum of errors (defined as the difference between the correct answer and the actual answer) running through all four examples. The success of the algorithm is reached when TOTAL_ERROR equals to zero, which means the weights are adjusted correctly, so the input (0,0) can be correctly marked as 0 by the step function (separated from the other three examples).

6. The Count variable marks the total numbers of generation until the algorithm finds the solution.

7. The total run times may differ, depending on the initial weights we draw from the uniform distribution. As we have a fixed weight-adjusting step (eta=0.8), when the random weights are drawn further from the final boundaries, it takes more time to converge.

8. Make a for loop outside covering the random sampling of weights. Build an empty vector with length of ten and store the count of each generation. Use the mean() function to calculate the average generation time:

```
generation_count<-rep(0,10)
for(generation in 1:10){

eta <- 0.8  # factor controlling how much weights should be adjusted
done <- FALSE     # Flag to signal when a solution has been found
Count <- 0        # counter for the number of batches (a batch = try all four examples)
W <- runif(3,-3,3) # initialize random uniform weights between -3 and 3
print(W)
SaveOutput <- c() # empty vector to save output


while(!done && Count < 25) {      # the ! is a NOT operator. Task will be solved when done = TRUE
```

9. Change the CORRECT_OUTPUT <- c(0,0,0,1)

11. The difference might not be so huge, but AND could take a little more time, since the w1 and w2 are of similar values comparing to the OR case, but the bias weight, which is expressed as the intercept in the figure, is larger for the AND case, so it might take more steps to converge.

ii) Advanced level

12. Perceptron is an algorithm for Supervised Learning of single layer binary linear classifiers. Optimal weight coefficients are automatically learned. Weights are multiplied with the input features and decision is made if the neuron is fired or not. Activation function applies a step rule to check if the output of the weighting function is greater than zero. If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.

13. NAND: (0,0)->1; (1,0)->1; (0,1)->1; (1,1)->0

14. As XOR problem is not linearly separable, there will always be at least one error, namely one correct answer will not be reached. This makes the minimal average error equals to ¼=0.25. To make the algorithm stays around the minimal error, we would need to identify the condition where TOTAL_ERROR can only be converging towards 1, and once it reaches the lowest possible error, any new iteration that will give a larger error will not be taken.

*For testing out the tasks related to MATLAB, you need to download the Deep Learning Toolbox as well: https://se.mathworks.com/products/deep-learning.html