Exercise 4 - (E7.3)

The required modules:

In [ ]:
```
import numpy as np

from utilities.activation_functions import hardlims
```

Let's define the problem variables. Similarly to what we've done in Exercise 3, we convert the squares into vectors by scanning them column by column, white squares are represented as -1, blue ones as 1. In this case we will also need to define the target variables t_1 and t_2:

In [ ]:
```
# First input
p1 = np.array([1, -1, -1, 1, 1, -1]).reshape(-1, 1)
t1 = -1

# Second input
p2 = np.array([1, 1, -1, 1, -1, 1]).reshape(-1, 1)
t2 = 1
```

The Hebb rule can be applied to find the weight matrix, considering the targets this be written as:

$$\mathbf{W} = t_1\mathbf{p}_1^T + t_2\mathbf{p}_2^T = \mathbf{T}\mathbf{P}^T \tag{1}$$

In [20]:
```
# Weight matrix
W = t1*p1.transpose() + t2*p2.transpose()
print("Hebb W matrix:\n", W)
```

```
Hebb W matrix:
 [[ 0  2  0  0 -2  2]]
```

We cen now define the prediction function:

In [21]:
```
# Layer output function
def predict(p: np.ndarray, W_fun: np.ndarray):
    return hardlims(np.dot(W_fun, p.reshape(-1, 1)))
```

To show the functioning of the so-found network through the use of the prediction function, we run the following simulations using the inputs:

- $p = p_1$
- $p = p_2$
- $p = p_1 + \eta$
- $p = p_2 + \eta$

where p is the network input, and η is a random noise.

In [22]:
```
# Predict the actual same inputs
y_pred_p1 = np.squeeze(predict(p1, W))
y_pred_p2 = np.squeeze(predict(p2, W))

# Predict with noisy input
p1_noisy = np.array([1, -1, 1, 1, 1, 1]).reshape(-1, 1)
p2_noisy = np.array([-1, 1, -1, 1, 1, 1]).reshape(-1, 1)

y_pred_p1_noisy = np.squeeze(predict(p1_noisy, W))
y_pred_p2_noisy = np.squeeze(predict(p2_noisy, W))

# Define map from number to string
def map_func(output: int) -> str:
    return "p1" if output == -1 else "p2"

print("Prediction for input p1:       ", map_func(y_pred_p1))
print("Prediction for input p2:       ", map_func(y_pred_p2))
print("Prediction for input noisy p1: ", map_func(y_pred_p1_noisy))
print("Prediction for input noisy p2: ", map_func(y_pred_p2_noisy))
```

```
Prediction for input p1:        p1
Prediction for input p2:        p2
Prediction for input noisy p1:  p1
Prediction for input noisy p2:  p2
```

The neural network is capable of producing accurate outputs for every input, including those with noise, as long as the lowest Hamming distance corresponds to its respective noiseless input.