import numpy as np

class HopfieldNetwork:

def \_\_init\_\_(self, n\_neurons):

self.n\_neurons = n\_neurons

self.weights = np.zeros((n\_neurons, n\_neurons))

def train(self, patterns):

for pattern in patterns:

self.weights += np.outer(pattern, pattern)

self.weights /= self.n\_neurons

np.fill\_diagonal(self.weights, 0)

def predict(self, pattern, max\_iterations=10):

for \_ in range(max\_iterations):

pattern = np.sign(np.dot(self.weights, pattern))

return pattern

def calculate\_energy(self, pattern):

return -0.5 \* np.dot(pattern, np.dot(self.weights, pattern))

if \_\_name\_\_ == '\_\_main\_\_':

patterns = np.array([

[-1, 1, -1, -1],

[-1, -1,- 1,- 1],

[-1, -1, 1, -1],

[-1, 1, -1, 1]

])

n\_neurons = patterns.shape[1]

network = HopfieldNetwork(n\_neurons)

network.train(patterns)

for pattern in patterns:

prediction = network.predict(pattern)

energy = network.calculate\_energy(prediction)

print('Input pattern:', pattern)

print('Predicted pattern:', prediction)

print('Energy:', energy)

print('---')