Neural Networks and Deep Learning (2026)

Assignment (1):

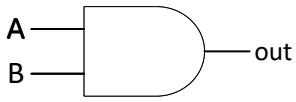
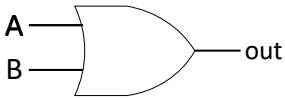
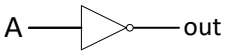
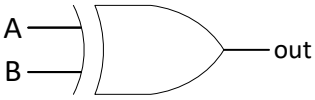
Neural Network Implementation of Logical Circuits

Shiraz University

1 Introduction

Logic gates are the fundamental building blocks of digital systems. They take one or more binary inputs (0 or 1) and produce a single binary output based on a logical operation. Common gates are represented in Table 1.

Table 1. Common logical gates

Logic Gate	Schematic	Boolean Expression	Truth table
AND		$A \wedge B$	A B out 0 0 0 0 1 0 1 0 0 1 1 1
OR		$A \vee B$	A B out 0 0 0 0 1 1 1 0 1 1 1 1
NOT		$\bar{A}$	A out 0 1 1 0
XOR		$A \oplus B$	A B out 0 0 0 0 1 1 1 0 1 1 1 0

Artificial neural networks (ANNs), particularly simple perceptron, can approximate or perfectly replicate these logical behaviors when configured properly. A function (or logic gate) is linearly separable if there exists a straight line (or hyperplane) that can divide all inputs producing 0s from those producing 1s. Two famous linearly separable logic gates are AND, OR logic gates. The XOR gate is nonlinearly separable because no single linear boundary can correctly separate its outputs in two-dimensional space.

While perceptron model logic functions as static input-output mappings, recurrent neural networks such as the Hopfield network provide a different perspective by modeling computation as a dynamical system with memory. Instead of learning decision boundaries, a Hopfield network stores patterns as stable states (attractors) in an energy landscape. Given an initial (possibly noisy) input, the network iteratively updates its neurons to minimize an energy function and converges to the closest stored pattern.

2 Problem Statement

2.1 NN implementation of multi-input Logic gates

We aim to simulate the behavior of logical circuits using simple linear neural networks (single-layer perceptron). Each network will be trained to reproduce the truth table of a given logical function. Each system is defined by a complex Boolean expression:

1. $out = (A \wedge \bar{B} \wedge \bar{C}) \vee (\bar{A} \wedge B \wedge \bar{C}) \vee (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge B \wedge C)$
2. $out = (A \wedge B \wedge C) \vee (A \wedge B \wedge D) \vee (A \wedge C \wedge D) \vee (B \wedge C \wedge D)$

you are going to do the following tasks:

1. List all possible input combinations using bipolar and binary representation (8 rows for 3 inputs, 16 rows for 4 inputs).
2. **Implement the Delta learning rule with appropriate activation function (binary/bipolar sigmoid)** from scratch to solve this classification problem. Report the accuracy for each of the logical circuits and each representation. Compare the results of each representation in terms of convergence speed.
3. Report which of the two logical systems can be perfectly learned by a single-layer network and which cannot and why?

2.2 NN implementation of SR latch

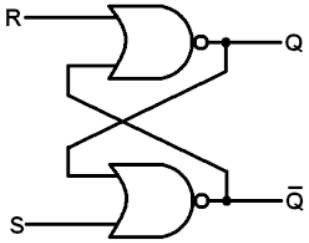
The SR (Set-Reset) latch is one of the simplest and most fundamental bistable memory circuits in digital electronics, capable of storing a single bit of information by maintaining its output state until directed to change by its inputs. It is widely used in sequential circuits, registers, and memory elements, making it a key building block for more complex logic systems.

In this assignment, you are tasked with modeling an SR latch using a Hopfield network. Consider the SR latch truth table with inputs S (set) and R (reset), and outputs Q and $\bar{Q}$, using bipolar encoding $(-1, +1)$, and assume that the valid stored patterns of the system are limited to $[Q, \bar{Q}] = [+1, -1]$ and $[-1, +1]$.

Construct a Hopfield network with an appropriate weight matrix to enforce these states as stable attractors. Using the Hopfield update rule with bias, define a bias vector as a function of S and R such that the network reproduces the SR latch behavior for set, reset, hold, and invalid conditions. Evaluate Hopfield network behavior starting from **corrupted versions of the stored patterns** and from invalid states. **Introduce noise by flipping one or more bits in the initial input vector**. For each test case, run the network until it reaches a stable state or a maximum number of iterations is reached. Record whether the

network converges, the final output state, the number of iterations required, and whether the final state matches one of the valid SR latch states. Repeat this test for several input combinations and report the percentage of successful convergence.

S	R	$Q(t+1)$
-1	-1	$Q(t)$
1	-1	1 (set)
-1	1	-1 (reset)
1	1	invalid



Notes:

- Allowed programming languages: Python, MATLAB
- Any sign of cheating would result in a zero grade for this assignment.
- You should upload your submissions at:
  https://quera.org/course/add_to_course/course/27529/

  All of the files should be in a ZIP file named in this format:
  "FirstNameFamilyName-SudentNumber.zip"
  Ex: "AmirZamani-4053040.zip"
- Your reports should be in a PDF file including: key points of your implementation, explanation of your chosen approach, reports of your final results and answers of assignment questions (if given).
- If you do not have **Anaconda** or **Python** installed, you can download Anaconda from the following link: [**DOWNLOAD LINK**]
- Use the following **domestic PyPi mirror** to install the required libraries:
  https://package-mirror.liara.ir/repository/pypi/simple
  For more information about this mirror, visit: **https://liara.ir/mirrors/pypi/**
- **Example:** "`pip install minisom --index-url https://package-mirror.liara.ir/repository/pypi/simple`"