UNIVERSIDAD SIGLO 21

INTELIGENCIA ARTIFICIAL

Licenciatura en Informática

Trabajo Práctico 3

Prof. Titular Disciplinar: María Paula González

Prof. Titular Experto: Pablo Alejandro Virgolini

Alumno: Pablo Alejandro Hamann

Legajo: VINF010782

Año: 2025

Tabla de contenido

1. Objetivo

El objetivo para esta actividad es:

1. Analizar, desarrollar e implementar una solución basada en modelos neuronales artificiales, específicamente una red de Hopfield, que permita identificar y corregir automáticamente pequeños desplazamientos en imágenes representativas del posicionamiento de componentes en una línea de montaje de block de motores robotizada.
2. Evaluar y justificar la efectividad del método aplicado mediante el desarrollo de un prototipo, destacando características, ventajas y limitaciones frente al problema concreto.

2. Situación problemática

La situación problemática se centra en la línea de montaje de blocks (de motores), donde se dan pequeños desplazamientos sobre la cinta transportadora que ocasionan errores en el posicionamiento de las piezas, interrumpiendo el proceso, y potencialmente llegando a detener toda la línea de producción. Estos problemas surgen porque los robots ensambladores carecen de autonomía para ajustar sus movimientos ante estas variaciones. La empresa busca implementar inteligencia artificial para otorgar a los robots un nivel suficiente de autonomía, permitiendo que puedan identificar y corregir automáticamente estos desplazamientos menores, ahora haciendo uso de **modelos neuronales artificiales como las redes de Hopfield**, que les otorguen a los robots, la capacidad autónoma suficiente para reconocer visualmente desplazamientos y corregir automáticamente sus trayectorias.

3. Desarrollo de las consignas

3.1. Análisis y resumen breve de modelos neuronales artificiales

Los modelos neuronales artificiales, se inspiran en el funcionamiento del cerebro humano: tienen la **capacidad de aprender y de adaptarse a partir de estímulos externos**. Su estructura básica se conforma por neuronas artificiales que procesan entradas, generan salidas, y ajustan sus conexiones (pesos sinápticos) según métodos específicos de aprendizaje.

Existen tres formas principales de aprendizaje:

1. **Aprendizaje Supervisado:** se proporcionan ejemplos con entradas y salidas esperadas. La red neuronal ajusta sus pesos sinápticos hasta lograr reproducir adecuadamente la función esperada.
   Ejemplos: reconocimiento de voz, de imágenes, y de patrones numéricos.
2. **Aprendizaje No Supervisado (autoorganizado):** solo se presentan entradas y la red descubre por sí misma patrones, regularidades o agrupamientos.
   Redes destacadas: mapas autoorganizados de Kohonen, redes de Hopfield.
   Aplicaciones: agrupamiento (*clustering*), identificación y recuperación de imágenes, reconocimiento de patrones y de voz.
3. **Aprendizaje por Refuerzo:** se basa en recompensas (refuerzos), donde el sistema aprende a partir de la evaluación continua de sus acciones para maximizar recompensas.
   Aplicaciones significativas en juegos y control de robots.

De estos modelos destaca especialmente el de las redes neuronales multicapa para el aprendizaje supervisado (como perceptrones multicapa con algoritmo *backpropagation*), altamente difundido por la capacidad para resolver problemas complejos, y las redes de Hopfield para el aprendizaje no supervisado, efectivas en recuperación y reconocimiento de imágenes deterioradas.

Estos modelos tienen aplicaciones críticas en el sector industrial como la automatización avanzada en procesos industriales, control autónomo en robótica, procesamiento y reconocimiento visual y sistemas de decisión autónomos… nuestro caso de estudio, sin ir más lejos, es un ejemplo concreto.

3.2. Análisis sobre la identificación de imágenes del problema abordado

Orientando el análisis hacia la identificación de imágenes similares al problema planteado, tenemos que los siguientes modelos resultan especialmente apropiados para nuestro problema:

1. **Redes neuronales multicapa (Aprendizaje Supervisado)**
    - **Ventajas:** Alta precisión en la clasificación y reconocimiento de imágenes, capacidad para aprender características complejas.
    - **Limitaciones:** Requiere grandes conjuntos de datos etiquetados para entrenamiento, alto costo computacional.
2. **Redes de Hopfield (Aprendizaje No Supervisado)**
    - **Ventajas:** Capacidad de recuperación y corrección de imágenes deterioradas o con ruido, adecuado para memorias asociativas.
    - **Limitaciones:** Capacidad limitada de almacenamiento (cantidad de imágenes almacenadas relacionada al número de neuronas), sensibilidad a la falta de ortogonalidad entre imágenes.

Aplicaciones:

- Las redes multicapa son útiles para tareas que requieren reconocimiento preciso, como la identificación exacta de componentes críticos en procesos industriales.
- Las redes de Hopfield resultan particularmente eficientes en la corrección y recuperación de imágenes parcialmente dañadas o mal posicionadas, ideales para el contexto industrial del problema abordado, donde la precisión en reconocimiento es crítica para el correcto funcionamiento de sistemas automatizados.

Estos modelos atacan problemas específicos en el reconocimiento y el procesamiento de imágenes, proporcionándonos soluciones efectivas según las características y necesidades del entorno operativo.

3.3. Desarrollo e implementación de un prototipo

A través del modelo de Hopfield en el contexto de un caso sencillo, elaboraremos un prototipo consiste en una clase Java simple que implementa la lógica básica de una red de Hopfield, utilizando la regla de aprendizaje de Hebb, y por matriz pseudoinversa, que nos permitirá:

- Almacenar patrones de imágenes representados como vectores.
- Recuperar imágenes dañadas o incompletas.

El prototipo opera sobre una imagen pequeña de 10 x 10 píxeles. Su finalidad es eliminar el ruido y otros elementos innecesarios para poder identificar con precisión el aro. Se lo hará juntamente con algún elemento de referencia de posición inalterable (marco inferior izquierdo).

Para el caso de la matriz pseudoinversa, en lugar de sumar productos de vectores (como en Hebb), la matriz de pesos se calculará mediante una fórmula matricial más general:

$$W + U * U^{\dagger}$$

Donde:

- U es la matriz de patrones columna (cada patrón es una columna).
- $U^{\dagger}$ es la pseudoinversa de U, definida como: $U^{\dagger} = (U^T * U)^{-1} * U^T$

Este método de matriz pseudoinversa, no requiere que los patrones sean ortogonales, y permite almacenar más patrones con mayor fidelidad.

Finalmente, para realizar todas esas operaciones matriciales, hacemos uso de una librería (externa al *Java OpenJDK*) llamada **JAMA**[1] (no está en los repositorios de Maven, aunque la

[1] JAMA : *A Java Matrix Package*. Web: https://math.nist.gov/javanumerics/jama/

incluí en el repositorio Git), que permite realizar operaciones de algebra líneas en alto nivel, de forma transparente. Una vez clonado el proyecto hay que instalarla manualmente:

```
REM Definir DIRBASE. Por ejemplo:
SET "DIRBASE=%USERPROFILE%\Downloads"
REM y luego indicarle a Maven:
mvn install:install-file ^
 -Dfile="%DIRBASE%\ia\Modelo-Hopfield\src\Jama-1.0.3.jar" ^
 -DgroupId=gov.nist.math ^
 -DartifactId=jama ^
 -Dversion=1.0.3 ^
 -Dpackaging=jar
```

Código Fuente

El código fuente se encuentra (junto a otros recursos de la materia), en el repositorio GitHub para la materia, en: https://github.com/linkstat/ia. El archivo de la clase Java, que contiene todo el código fuente (incluidos comentarios y *Javadoc*) se encuentra en el siguiente enlace: https://github.com/linkstat/ia/blob/master/Modelo-Hopfield/src/ar/edu/uesiglo21/ModeloHopfield.java
Por cuestiones de espacio en el documento, se exponen solamente fragmentos relevantes de código, como la estructura general de la clase, algunos métodos principales, y el método *main*. El proyecto completo, está en el repositorio en GitHub.

Estructura de la clase

```
package ar.edu.uesiglo21;

import java.util.Arrays;
import java.util.Scanner;
import Jama.Matrix;

public class ModeloHopfield {
    private int[][] pesos;
    private int tamano;

    public ModeloHopfield(int tamano) {
        this.tamano = tamano;
        pesos = new int[tamano][tamano];
    }

    public void entrenarHebb(int[][] patrones) {...}
```

```
    public void entrenarPseudoinversa(int[][] patrones) {...}

    public int[] rellamarSincronico(int[] patron, int iteraciones) {...}

    public int[] rellamarAsincronico(int[] patron, int iteraciones, int ancho) {...}

    public static void printPatronEnCuadricula(int[] patron, int ancho) {...}

    public static void printPatron(int[] patron, int ancho) {...}

    public static void verificarSimilitudPatrones(int[] patron1, int[] patron2,
double umbral) {...}

    public static void main(String[] args) {...}

}
```

Método de entrenamiento por regla de aprendizaje de Hebb

```
public void entrenarHebb(int[][] patrones) {
    System.out.println("\n🔹🔹🔹  Entrenando la red neuronal con Hebb...  🔹🔹🔹");
    for (int idx = 0; idx < patrones.length; idx++) {
        int[] p = patrones[idx];
        System.out.println("Entrenando con el patrón " + (idx + 1));
        for (int i = 0; i < tamano; i++) {
            for (int j = 0; j < tamano; j++) {
                if (i != j) {
                    pesos[i][j] += p[i] * p[j];
                }
            }
        }
    }
    System.out.println("✅ ¡Finalizado! Red neuronal entrenada con Hebb.");
}
```

Método de entrenamiento por regla de aprendizaje de matriz pseudoinversa

```
public void entrenarPseudoinversa(int[][] patrones) {
    System.out.println("\n🔹🔹🔹  Entrenando la red neuronal con
pseudoinversa...  🔹🔹🔹");
    int q = patrones.length;
    int n = patrones[0].length;
```

```
    if (q < 2) {
        System.out.println("⚠ Se recomienda al menos dos patrones para usar
pseudoinversa. Usando regla de Hebb en su lugar.");
        entrenarHebb(patrones);
        return;
    }

    double[][] uData = new double[n][q];
    for (int p = 0; p < q; p++) {
        for (int i = 0; i < n; i++) {
            uData[i][p] = patrones[p][i];
        }
    }

    Matrix U = new Matrix(uData);           // U : [n x q]
    Matrix UT = U.transpose();              // U^T : [q x n]
    Matrix UTU = UT.times(U);               // [q x q]
    Matrix UTUinv = null;
    try {
        UTUinv = UTU.inverse();             // [q x q]
    } catch (RuntimeException e) {
        System.out.println("× No se pudo invertir U^T * U. Esto pasa cuando no hay
independencia lineal entre los patrones");
        return;
    }
    Matrix Udag = UTUinv.times(UT);         // [q x n]
    Matrix W = U.times(Udag);               // [n x n]
    for (int i = 0; i < W.getRowDimension(); i++) {
        W.set(i, i, 0);
    }

    pesos = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            double w = W.get(i, j);
            pesos[i][j] = (w > 0.01) ? 1 : (w < -0.01) ? -1 : 0;
        }
    }
    System.out.println("√ ¡Finalizado! Red neuronal entrenada con pseudoinversa.");
}
```

Método de ejecución sincrónico (no muestra el paso a paso)

```
public int[] rellamarSincronico(int[] patron, int iteraciones) {
    int[] resultado = Arrays.copyOf(patron, tamano);
    System.out.println("\n♦♦  Iniciando recuperación del patrón...  ♦♦");
    for (int iteracion = 0; iteracion < iteraciones; iteracion++) {
        System.out.println("\nIteración " + (iteracion + 1) + ":");
        int[] nuevoResultado = new int[tamano];
        for (int i = 0; i < tamano; i++) {
            int suma = 0;
            for (int j = 0; j < tamano; j++) {
                suma += pesos[i][j] * resultado[j];
            }
            nuevoResultado[i] = suma >= 0 ? 1 : -1;
        }

        if (Arrays.equals(nuevoResultado, resultado)) {
            System.out.println("Patrón estable alcanzado en iteración " +
(iteracion + 1) +"♦♦♦♦");
            printPatron(nuevoResultado, (int)Math.sqrt(tamano));
            break;
        }

        resultado = nuevoResultado;
        printPatron(resultado, (int)Math.sqrt(tamano));
    }
    System.out.println("\n♦♦  ¡Reconstrucción completada!  ♦♦");
    return resultado;
}
```

Método de ejecución asincrónico (muestra el paso a paso, neurona por neurona)

```
public int[] rellamarAsincronico(int[] patron, int iteraciones, int ancho) {
    int[] resultado = Arrays.copyOf(patron, tamano);

    System.out.println("\n-- Iniciando recuperación (actualización secuencial) --");
    printPatron(resultado, ancho);

    for (int iteracion = 0; iteracion < iteraciones; iteracion++) {
        System.out.println("\nIteración " + (iteracion + 1) + " (neurona por
neurona):");

        boolean cambios = false;
```

```
        for (int i = 0; i < tamano; i++) {
            int sum = 0;
            for (int j = 0; j < tamano; j++) {
                sum += pesos[i][j] * resultado[j];
            }

            int valorAnterior = resultado[i];
            resultado[i] = sum >= 0 ? 1 : -1;

            // Mostrar si la neurona cambia
            if (valorAnterior != resultado[i]) {
                cambios = true;
                System.out.println("\n⚙ Cambio en la neurona " + i + ":");
                System.out.println("Antes:");
                resultado[i] = valorAnterior;
                printPatron(resultado, ancho);
                resultado[i] = sum >= 0 ? 1 : -1;
                System.out.println("Después:");
                printPatron(resultado, ancho);
            }
        }

        if (!cambios) {
            System.out.println("\n√ Patrón estable alcanzado en iteración " +
(iteracion + 1));
            break;
        }
    }

    System.out.println("--- Recuperación completada ---");
    return resultado;
}
```

Método de impresión por pantalla del patrón

```
public static void printPatron(int[] patron, int ancho) {
    for (int i = 0; i < patron.length; i++) {
        System.out.print(patron[i] == 1 ? "" : ".");
        if ((i + 1) % ancho == 0) System.out.println();
    }
}
```

Método de verificación de similitudes entre patrones

```
public static void verificarSimilitudPatrones(int[] patron1, int[] patron2, double
umbral) {
    if (patron1 == null || patron2 == null || patron1.length != patron2.length) {
        System.out.println("No se pueden comparar patrones: alguno es nulo o de
diferente longitud.");
        return;
    }
    int suma = 0;
    for (int i = 0; i < patron1.length; i++) {
        suma += patron1[i] * patron2[i];
    }
    double similitud = (double) suma / patron1.length;

    System.out.printf("→ Similitud entre patrones: %.2f\n", similitud);

    if (Math.abs(similitud) > umbral) {
        System.out.println("⚠ ADVERTENCIA: Los patrones agregados son demasiado
similares.");
        System.out.println("  Un valor alto (positivo o negativo) significa patrones
poco independientes y potencialmente problemáticos para la red de Hopfield.");
        System.out.println("  Esto puede provocar confusión en la red de
Hopfield.");
    } else {
        System.out.println("✓ Los patrones son suficientemente diferentes. Va a
funcionar...");
    }
}
```

Método main

```
public static void main(String[] args) {
    int ancho = 10;
    int tamano = ancho * ancho;
    // Patrón ideal (acorde al ejemplo del caso de estudio)
    int[] patronLimpio = {
            -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
            -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
            -1,-1,-1, 1, 1,-1,-1,-1,-1,-1,
            -1,-1, 1, 1, 1, 1,-1,-1,-1,-1,
            -1, 1, 1,-1,-1, 1, 1,-1,-1,-1,
            -1, 1, 1,-1,-1, 1, 1,-1,-1,-1,
            -1,-1, 1, 1, 1, 1,-1,-1,-1,-1,
```

```
            -1,-1,-1, 1, 1,-1,-1,-1,-1,-1,
             1,-1,-1,-1,-1,-1,-1,-1,-1,-1,  // ref: elemento fijo en pos. 80
             1, 1,-1,-1,-1,-1,-1,-1,-1,-1   // ref: elementos fijos en pos. 90 y 91
    };
    // Elemento fijo de referencia (esquina inferior izquierda)
    patronLimpio[80] = 1;
    patronLimpio[90] = 1;
    patronLimpio[91] = 1;

    // Segundo patrón
    int[] patronLimpio2 = {
             1, 1, 1,-1, 1, 1, 1,-1, 1, 1,
            -1,-1, 1, 1, 1,-1, 1, 1, 1,-1,
             1, 1, 1,-1, 1, 1, 1,-1,-1, 1,
             1, 1,-1,-1,-1,-1, 1, 1, 1, 1,
             1,-1,-1, 1, 1,-1,-1, 1,-1, 1,
             1,-1,-1, 1, 1,-1,-1, 1, 1, 1,
             1, 1,-1,-1,-1,-1, 1, 1,-1, 1,
             1,-1,-1,-1,-1,-1, 1,-1,-1, 1,
             1, 1, 1, 1,-1,-1, 1, 1, 1, 1,  // ref: elemento fijo en pos. 80
             1, 1, 1, 1, 1, 1, 1, 1,-1, 1   // ref: elementos fijos en pos. 90 y 91
    };
    patronLimpio2[80] = 1;
    patronLimpio2[90] = 1;
    patronLimpio2[91] = 1;

    // Patrón inicial dañado (acorde a ejemplo del caso de estudio)
    int[] patronSucio = new int[patronLimpio.length];
    System.arraycopy(patronLimpio, 0, patronSucio, 3, patronLimpio.length - 3);
    // Mantenemos las posiciones fijas (80, 90, 91)
    patronSucio[80] = patronLimpio[80];
    patronSucio[90] = patronLimpio[90];
    patronSucio[91] = patronLimpio[91];
    // Definimos posiciones con ruido (acorde a ejemplo de las consignas del TP3)
    int[] posicionesConSuciedad = {
            0, 7, 11, 14, 31, 61, 82, 86, 89, 93};

    // Aplicamos ruido a las posiciones específicas
    for (int pos : posicionesConSuciedad) {
        patronSucio[pos] = (patronSucio[pos] == 1) ? -1 : 1;
    }

    Scanner sc = new Scanner(System.in);
    boolean usarDosPatrones = false;
    boolean salir = false;
```

```
    while (!salir) {
        System.out.println("\nPrototipo de implementación del modelo neuronal de
Hopfield\n");
        System.out.println("================");
        System.out.println("MENÚ DE OPCIONES:");
        System.out.println("================");
        System.out.println("1. Entrenar con Hebb y mostrar patrón recuperado.");
        System.out.println("2. Entrenar con Hebb y mostrar patrón paso a paso
(asíncrono).");
        System.out.println("3. Entrenar con Pseudoinversa y mostrar patrón
recuperado.");
        System.out.println("4. Entrenar con Pseudoinversa y mostrar patrón paso a
paso (asíncrono).");
        System.out.printf("5. Mostrar %s.\n", !usarDosPatrones ? "patrón limpio" :
"patrones limpios");
        System.out.println("6. Mostrar patrón dañado (hardcodeado).");
        System.out.printf("7. %s un segundo patrón de memoria.\n", usarDosPatrones ?
"Quitar" : "Agregar");
        System.out.println("\n0. Salir.");
        System.out.println("================");
        System.out.print("¿Su opción? [0-6]: ");
        int opcion = sc.hasNextInt() ? sc.nextInt() : -1;
        sc.nextLine();

        int[][] patrones = usarDosPatrones ? new int[][]{patronLimpio,
patronLimpio2} : new int[][]{patronLimpio};
        ModeloHopfield modeloHebb = new ModeloHopfield(tamano);
        ModeloHopfield modeloPseudo = new ModeloHopfield(tamano);

        switch (opcion) {
            case 1:
                modeloHebb.entrenarHebb(patrones);
                int[] recHebb = modeloHebb.rellamarSincronico(patronSucio, 10);
                System.out.println("→ Patrón recuperado (Hebb):");
                ModeloHopfield.printPatron(recHebb, ancho);
                break;
            case 2:
                modeloHebb.entrenarHebb(patrones);
                int[] recHebbPaso = modeloHebb.rellamarAsincronico(patronSucio, 10,
ancho);
                System.out.println("→ Patrón recuperado paso a paso (Hebb):");
                ModeloHopfield.printPatron(recHebbPaso, ancho);
                break;
            case 3:
                modeloPseudo.entrenarPseudoinversa(patrones);
```

```
                int[] recPseudo = modeloPseudo.rellamarSincronico(patronSucio, 10);
                System.out.println("→ Patrón recuperado (Pseudoinversa):");
                ModeloHopfield.printPatron(recPseudo, ancho);
                break;
            case 4:
                modeloPseudo.entrenarPseudoinversa(patrones);
                int[] recPseudoPaso = modeloPseudo.rellamarAsincronico(patronSucio,
10, ancho);
                System.out.println("→ Patrón recuperado paso a paso
(Pseudoinversa):");
                ModeloHopfield.printPatron(recPseudoPaso, ancho);
                break;
            case 5:
                System.out.println("\n→ Patrón limpio:");
                ModeloHopfield.printPatron(patronLimpio, ancho);
                if (usarDosPatrones) {
                    System.out.println("\n→ Segundo patrón limpio:");
                    ModeloHopfield.printPatron(patronLimpio2, ancho);
                }
                break;
            case 6:
                System.out.println("→ Patrón dañado (entrada):");
                ModeloHopfield.printPatron(patronSucio, ancho);
                break;
            case 7:
                usarDosPatrones = !usarDosPatrones;
                System.out.printf("%s el segundo patrón de memoria. Ahora se %s %d
%s.\n",
                        usarDosPatrones ? "Agregado" : "Quitado",
                        usarDosPatrones ? "usan" : "usa",
                        usarDosPatrones ? 2 : 1,
                        usarDosPatrones ? "patrones" : "patrón");
                if (usarDosPatrones) {
                    verificarSimilitudPatrones(patronLimpio, patronLimpio2, 0.6);
                }
                break;
            case 0:
                System.out.println("\n\nDe aquí hasta reconocer rostros no paramos
;-)!!!\nSaludos profes!!!");
                salir = true;
                break;
            default:
                System.out.println("Opción inválida. De introducir un entero  entre
0 y 7.");
        }
```

```
    }
    sc.close();
}
```

Descripción de características, ventajas y limitaciones

El modelo de red neuronal de Hopfield es un tipo de red recurrente utilizada como memoria asociativa, capaz de almacenar y recuperar patrones previamente aprendidos. Sus características clave en el contexto de identificación de imágenes industriales (como la detección y corrección de desplazamientos en una línea de montaje) son:

Características:

- **Capacidad de memoria asociativa:** puede recordar y reconstruir patrones completos a partir de versiones incompletas o con ruido, ideal para recuperar imágenes dañadas o desplazadas.
- **Simplicidad de implementación:** el modelo es sencillo tanto en la estructura de datos como en las reglas de actualización, lo que permite prototipos rápidos y claros.
- **Aprendizaje por regla de Hebb y Matriz Pseudoinversa:**
- **Regla de Hebb:** adecuada para almacenar un número reducido de patrones diferentes y robusta ante ruido moderado.
- **Matriz Pseudoinversa:** permite una mejor separación de patrones si son suficientemente ortogonales entre sí, pero es más sensible a la calidad y variedad de los patrones de entrenamiento.

Ventajas:

- **Robustez ante ruido:** el modelo es capaz de corregir imágenes con defectos menores y de “recordar” el patrón más parecido.
- **Recuperación rápida:** el proceso de recuperación es eficiente y en imágenes pequeñas converge en pocas iteraciones.

- **Aplicación directa a problemas de reconocimiento y corrección en líneas de montaje**, donde se necesita detectar y ajustar pequeños desplazamientos… ideal para pequeños entornos industriales donde se requiere reconocimiento rápido y corrección automática de errores menores (lo que nos interesa en este TP).
- **Implementación sencilla y bajo costo computacional** para casos simples.

Limitaciones:

- **Capacidad limitada:** solo puede almacenar correctamente hasta un cierto número de patrones (aproximadamente 0.15 × número de neuronas para Hebb).
- **Sensibilidad a la ortogonalidad:** si los patrones de entrenamiento no son lo suficientemente diferentes (ortogonales), se pueden producir errores de recuperación (patrones espurios).
- **Resultados sensibles a la cantidad y la diversidad de patrones:** en el caso de la pseudoinversa, patrones poco variados o muy similares generan pesos demasiado pequeños y pobre desempeño (lo que sucede en el ejemplo dado).
- **Dificultad para escalar:** al aumentar el tamaño de la imagen, crecen la complejidad y los requisitos de memoria, y la red se vuelve menos eficiente.

Presentación de un ejemplo

Imagen simple de 10x10 píxeles para demostrar la recuperación de una figura geométrica (por ejemplo, un cuadrado) distorsionada mediante el método de Hebb, permitiendo observar claramente el proceso de identificación y corrección.

Patrón dañado:

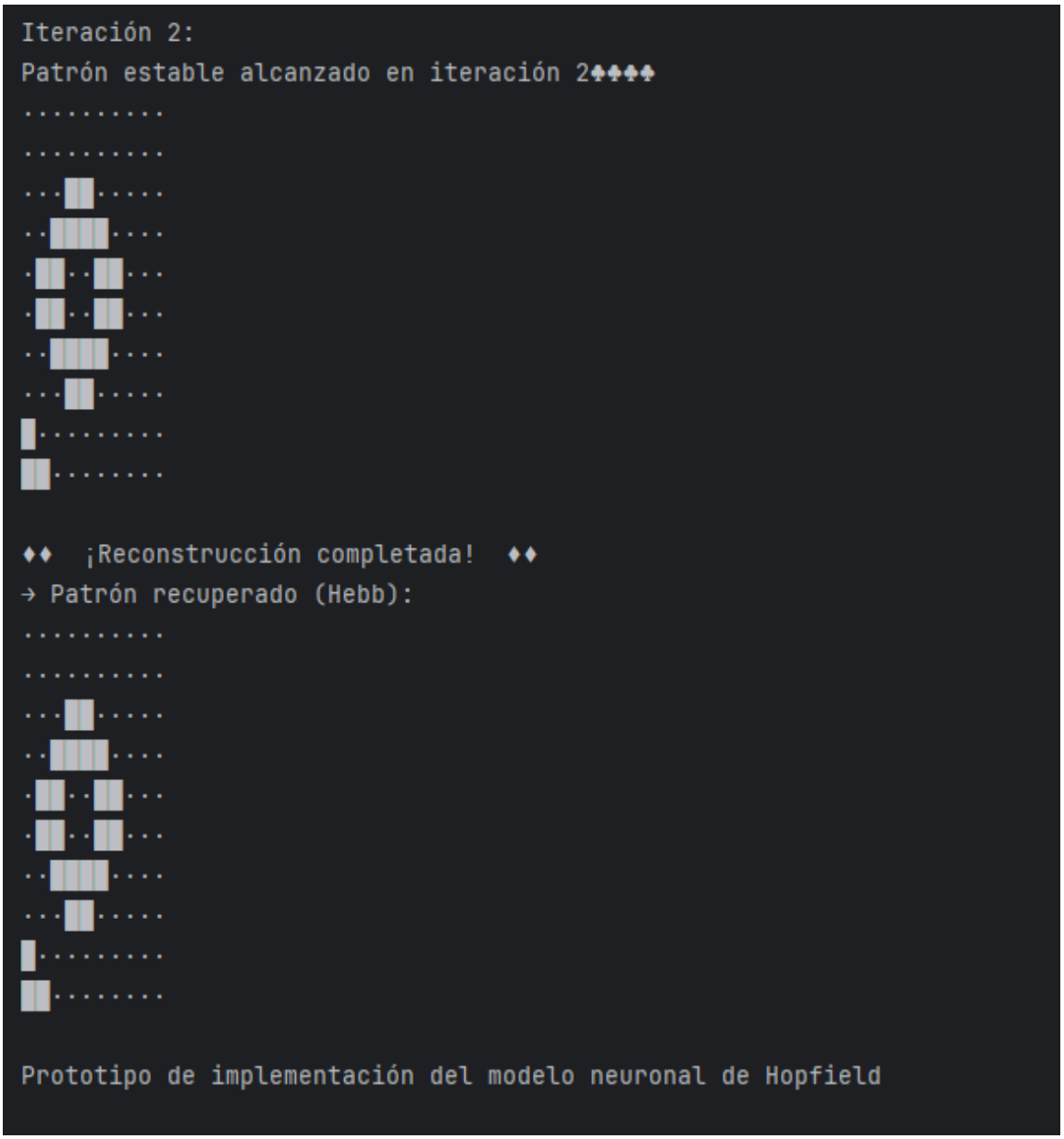
```
Prototipo de implementación del modelo neuronal de Hopfield

=================
MENÚ DE OPCIONES:
=================
1. Entrenar con Hebb y mostrar patrón recuperado.
2. Entrenar con Hebb y mostrar patrón paso a paso (asíncrono).
3. Entrenar con Pseudoinversa y mostrar patrón recuperado.
4. Entrenar con Pseudoinversa y mostrar patrón paso a paso (asíncrono).
5. Mostrar patrón limpio.
6. Mostrar patrón dañado (hardcodeado).
7. Agregar un segundo patrón de memoria.

0. Salir.
=================
¿Su opción? [0-6]: 5

→ Patrón limpio:
..........
..........
...██.....
..████....
.██..██...
.██..██...
..████....
...██.....
█.........
██........
```

Patrón limpio:

```
Prototipo de implementación del modelo neuronal de Hopfield

=================
MENÚ DE OPCIONES:
=================
1. Entrenar con Hebb y mostrar patrón recuperado.
2. Entrenar con Hebb y mostrar patrón paso a paso (asíncrono).
3. Entrenar con Pseudoinversa y mostrar patrón recuperado.
4. Entrenar con Pseudoinversa y mostrar patrón paso a paso (asíncrono).
5. Mostrar patrón limpio.
6. Mostrar patrón dañado (hardcodeado).
7. Agregar un segundo patrón de memoria.

0. Salir.
=================
¿Su opción? [0-6]: 6
→ Patrón dañado (entrada):
#·····#··
·#··#·····
·····##··
·#···####·
····##··##
····##··##
·#···####·
·····##··
#·##··#··#
##··#·····

Prototipo de implementación del modelo neuronal de Hopfield
```

Patrón recuperado con entrenamiento Hebb:

```
Prototipo de implementación del modelo neuronal de Hopfield

=================
MENÚ DE OPCIONES:
=================
1. Entrenar con Hebb y mostrar patrón recuperado.
2. Entrenar con Hebb y mostrar patrón paso a paso (asíncrono).
3. Entrenar con Pseudoinversa y mostrar patrón recuperado.
4. Entrenar con Pseudoinversa y mostrar patrón paso a paso (asíncrono).
5. Mostrar patrón limpio.
6. Mostrar patrón dañado (hardcodeado).
7. Agregar un segundo patrón de memoria.

0. Salir.
=================
¿Su opción? [0-6]: 1

☻☻☻  Entrenando la red neuronal con Hebb...  ☻☻☻
Entrenando con el patrón 1
√ ¡Finalizado! Red neuronal entrenada con Hebb.

♦♦  Iniciando recuperación del patrón...  ♦♦

Iteración 1:
..........
..........
...██.....
..████....
.██..██...
.██..██...
..████....
...██.....
█.........
██........
```



Resumen de las dificultades encontradas

Durante la implementación y pruebas del prototipo, me encontré con las siguientes dificultades:

- **Problemas con el lenguaje de programación utilizado:** *Java* no es el lenguaje más popular para este tipo de desarrollos (como lo son *Python* por ejemplo). No llegué a desarrollar una interfaz gráfica, y usé consola de texto que no es lo más atractivo para representar patrones (aun así, eché mano a todo el subconjunto Unicode para dar con los símbolos más convenientes). Para la implementación de la

pseudoinversa, tuve que hacer uso de librerías externas, lo que agrega complejidad a la ejecución del código por terceros.

- **Problemas con la pseudoinversa:** al usar la pseudoinversa con patrones poco ortogonales o escasa variedad, los pesos resultaron muy pequeños, generando patrones de salida saturados (todo unos básicamente). Para esto tuve que modificar el método de redondeo (redondeando solo con umbral en vez de Math.round); esto ayudó a mitigar, aunque para lograr una recuperación del patrón limpio, tuve que desordenar más mi segundo patrón, para estuvieran bien diferenciados. Adicionalmente, se desarrolló un método que muestra una advertencia cuando los patrones son "poco ortogonales" o "muy similares" (la ortogonalidad se determina mediante producto escalar, y el umbral se define manualmente en el código).
- **Ruido y desplazamientos:** si el patrón dañado difiere demasiado del patrón almacenado (por ejemplo, desplazamientos grandes), la red no es capaz de recuperar la imagen original.
- **Ajuste de umbrales de activación:** tuve que modificar varias veces el umbral para transformar los valores de la matriz de pesos en 1, -1 o 0, porque los valores eran muy pequeñitos y la activación por signo no funcionaba bien.
- **Visualización y depuración:** fue necesario imprimir la matriz de pesos antes y después de la binarización para entender qué pasaba (cuál era la falla); eso me permitió verificar lo descripto en el inciso anterior y poder “corregir” el prototipo.

3.4. Análisis de coincidencias y diferencias entre el problema de fondo y las posibilidades que ofrecen el método aplicado y su prototipo

El problema planteado, corresponde a una línea de montaje industrial de block de motores, donde se requiere **precisión absoluta en la identificación y posicionamiento de piezas** por parte del robot, para realizar operaciones críticas. Las redes neuronales de tipo Hopfield y el prototipo desarrollado ofrecen una posible solución por el lado del reconocimiento visual y la corrección automática de imágenes distorsionadas, incompletas, sucias, etc.

Coincidencias

Capacidad de corrección: Ambas situaciones (la real y la del prototipo) involucran la necesidad de corregir o recuperar imágenes parcialmente dañadas o distorsionadas. Así como el bloque puede estar mal posicionado en la línea, el prototipo debe recuperar una imagen deteriorada. El modelo de Hopfield puede reconstruir estas imágenes, coincidiendo con la necesidad planteada en el caso dado, de corrección automática de desviaciones menores.

Robustez ante ruido o desplazamientos menores: Tanto en el prototipo como en el problema real, se requiere tolerancia a alteraciones leves (ruido, suciedad, desplazamientos pequeños). La red de Hopfield demuestra ser eficaz para recuperar patrones ligeramente alterados, una capacidad esencial para reducir errores en la línea de montaje.

Adaptabilidad y aprendizaje: La posibilidad de entrenar patrones específicos en la red neuronal permite adaptarse a variaciones comunes y situaciones recurrentes en el entorno industrial, lo que puede traducirse en una mayor autonomía y menor dependencia de intervención humana.

Diferencias

Complejidad del problema real vs. simplicidad del prototipo: el prototipo opera sobre matrices simples y de baja resolución, mientras que en la industria real se manejan imágenes de alta resolución y con detalles técnicos mucho más complejos.

Requerimientos de precisión industrial: la precisión requerida en una línea de montaje industrial (por ejemplo, tolerancias de menos de un milímetro) es considerablemente mayor que la que puede ofrecer una red de Hopfield básica. Para lograr los niveles de exactitud necesarios, sería preciso incorporar modelos más avanzados o redes multicapa que permitan un procesamiento de imágenes más sofisticado.

Escalabilidad y rendimiento: el modelo de Hopfield tiene una capacidad de almacenamiento limitada (directamente proporcional al número de neuronas). Esto dificulta su aplicación directa en entornos industriales donde hay una mayor variedad de patrones y situaciones posibles.

Justificación

La aplicación práctica del modelo de Hopfield en su forma básica, tal como se ha implementado en el prototipo desarrollado, sirve principalmente como prueba conceptual y educativa del potencial que tienen los modelos neuronales artificiales para resolver problemas reales relacionados con reconocimiento y corrección de imágenes.

Sin embargo, para llevar esta solución a un entorno real industrial (como la línea de montaje de nuestro caso), sería indispensable:

- Escalar el modelo y aumentar la complejidad de las redes empleadas.
- Incorporar técnicas adicionales para el procesamiento de las imágenes.
- Implementar modelos neuronales más avanzados, que sean capaces de complementarse con otros métodos de inteligencia artificial que proporcionen la precisión milimétrica necesaria para nuestra línea de montaje industrial.

El prototipo actual no es aplicable (así como está desarrollado) a un problema industrial real como el caso planteado. En cambio, sí podemos tomarlo como un valioso primer paso para explorar soluciones basadas en IA y establecer una hoja de ruta que nos permitan alcanzar implementaciones más complejas y eficientes.

4. Repositorio en Github

Accesos directos:

- **Repositorio de la materia:** https://github.com/linkstat/ia
- **Este documento (PDF):** https://github.com/linkstat/ia/raw/refs/heads/master/docs/HAMANN-PABLO-ALEJANDRO-TP3.pdf
- **Clase Java (RAW):** https://github.com/linkstat/ia/blob/master/Modelo-Hopfield/src/ar/edu/uesiglo21/ModeloHopfield.java
- **Proyecto Java (IntelliJ):** https://github.com/linkstat/ia/tree/master/Modelo-Hopfield
- **Javadoc:** https://github.com/linkstat/ia/tree/master/docs/javadoc