ICOM2021 – Examen Final

27 de enero de 2022

Notas:

1. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados.**

Problema 1: El Juego de la Predicción (no holística)

En este juego se solicita al usuario que ingrese una secuencia de caracteres compuesta por estos dos { x, o } de la forma más aleatoria posible. El algoritmo intentará predecir cuál será el próximo caracter de la secuencia. Si adivina recibe 1 punto, y si falla el usuario recibe 1 punto. Gana quien primero llegue a juntar 50 puntos.

Se solicita implementar el siguiente algoritmo predictor: tomando los 4 últimos caracteres ingresados por el usuario (supongamos por ejemplo que fueron “xxox”) se computa la frecuencia (fx) con que a ese patrón lo siguió una x (“xxoxx”) y la frecuencia (fo) con que lo siguió una o (“xxoxo”) en toda la secuencia hasta ahora. Si fx > fo se genera la predicción x y si es menor, o. En caso de ser iguales (o no poder calcularlas), se elige al azar. Si la predicción es correcta, el algoritmo se anota 1 punto y si no, lo hace el usuario. El programa debe continuar solicitando caracteres al usuario (y agregándolos a la secuencia) hasta que uno de los jugadores llegue a 50 puntos. Y al alcanzar el resultado, comunicarlo al usuario y terminar (si gana el algoritmo, vale verduguear un poco al usuario…)

Hint: este ejercicio es mucho más fácil si utiliza std::string en lugar de strings nativos

Problema 2: Semánticas FIFO y LIFO

En un sistema que implementa una política FIFO (First-In-First_Out) para la organización y manipulación de datos, el procesamiento se realiza sobre el elementó más “antiguo” disponible en el sistema (otra forma de verlo es que se respeta como orden de salida el mismo orden de la entrada).

Por otro lado, si un sistema implementa una política LIFO (Last-In-First_Out) para la organización y manipulación de datos, el procesamiento se realiza sobre el elementó más “nuevo” disponible en el sistema.

Se desea implementar clases que implementen esas políticas, pero que no estén atadas a un contenedor particular de información.

Para ello se definen las interfases:

```
class FIFO_IF {
  public:
    // agrega un nuevo elemento, excepción cuando el contenedor está lleno
    virtual void put(int e) = 0;

    // retorna el elemento más viejo, excepción cuando el contenedor está vacío
    virtual int  get() = 0;

    // chequea si el contenedor está vacío
    virtual bool isEmpty() = 0;

    // vacía el contenedor
    virtual void clear() = 0;


    struct ERROR_FIFO_FULL {};
    struct ERROR_FIFO_EMPTY {};
};
```

```
class LIFO_IF {
  public:
    // agrega un nuevo elemento, excepción cuando el contenedor está lleno
    virtual void push(int e) = 0;

    // retorna el elemento más viejo, excepción cuando el contenedor está vacío
    virtual int  pop() = 0;

    // chequea si el contenedor está vacío
    virtual bool isEmpty() = 0;

    // vacía el contenedor
    virtual void clear() = 0;

    struct ERROR_LIFO_FULL {};
    struct ERROR_LIFO_EMPTY {};
};
```

Implemente las siguientes clases concretas:

- **VectorAsFIFO**: Implementa un contenedor basado en un **std::vector** cumpliendo la interfase **FIFO_IF**
- **VectorAsLIFO**: Implementa un contenedor basado en un **std::vector** cumpliendo la interfase **LIFO_IF**
- **ListAsFIFO**: Implementa un contenedor basado en un **std::list** cumpliendo la interfase **FIFO_IF**
- **ListAsLIFO**: Implementa un contenedor basado en un **std::list** cumpliendo la interfase **LIFO_IF**

Utilice el esqueleto dado en xifo_test para probar las clases creadas.

Problema 3: Please remember

Las redes neuronales de Hopfield se usan como sistemas de memoria asociativa con unidades binarias. En otras palabras, es una red de neuronas conectadas entre si que pueden tomar solo dos valores 1 y -1 con la capacidad de retener patrones en memoria. Están diseñadas para converger a un mínimo local, pero la convergencia a uno de los patrones almacenados no está garantizada. En caso de almacenar patrones 2D el tamaño de la red neuronal es de NumNeuronas = XRes $*$ YRes.

La red neuronal en este caso consta de neuronas representadas con números enteros que pueden valer 1 o -1. Una vez presentado una condición inicial al sistema este evoluciona en pasos discreto cambiando los valores de sus neuronas en cada paso (según el valor de las demás neuronas y los pesos asociados a sus conexiones). Paso a paso el sistema evoluciona a un mínimo local (como muestra la Fig 1.), y con suerte a un patrón aprendido en memoria.

- Si bien las imágenes son patrones en 2D estos pueden ser representados en cadenas unidimensionales. Los patrones los podemos escribir como:

$$\varepsilon_i^\mu (i = 0, \dots, N-1; \mu = 0, \dots, p-1).$$

Con N: número de pixeles y p : número de patrones

- Cada una de las neuronas de nuestra red está conectada a todas las demás. En total hay NumPixeles^2 cantidad de conexiones en nuestra red. Con el fin de crear una red de memoria asociativa las conexiones entre neuronas deben valer:

$$w_{ij} = \frac{1}{N} \sum_{\mu=0}^{p-1} \varepsilon_i^\mu \, \varepsilon_j^\mu \;\; (w_{ij} = 0 \; cuando \; i = j)$$

- Cada neurona (S_i) de esta red es un sistema dinámico gobernado por la siguiente ecuación:

$$S_i = sgn\left(\sum_j^N w_{ij}\ S_j\right)$$

La red actualiza todas sus neuronas con esta ecuación por cada paso de tiempo desde una condición inicial (patrón de prueba). La actualización de las neuronas se debe hacer en orden y sin memoria (la última neurona se actualiza con los valores de todas las neuronas ya actualizadas). Debemos dejar de actualizar en cuanto haya llegado a una solución estable, es decir, la mayoría de neuronas mantuvieron su valor pese a que fueron actualizados. La otra forma de detener la evolución del sistema, es con un máximo número de iteraciones (implementar las dos formas). Una vez dejado de actualizar el sistema, si el estado coincide con algún patrón aprendido, decimos que la memoria funciona correctamente.

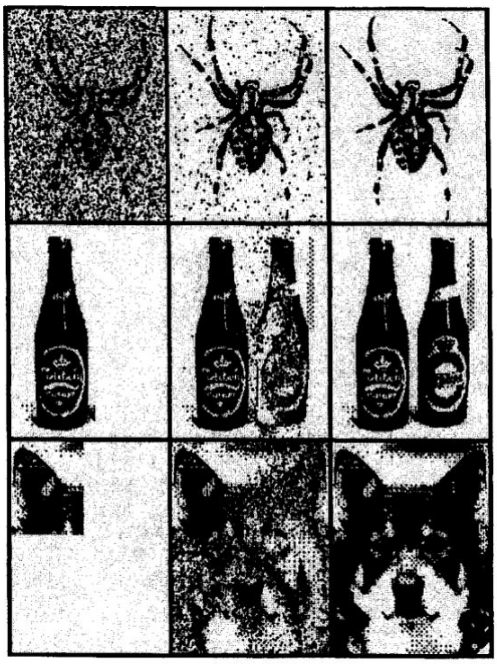

Figura 1: Evolución temporal (de izquierda a derecha) de una red binaria de memoria asociativa partiendo de un patrón incompleto/dañado, convergiendo un patrón aprendido en memoria.

En el archivo **`please_remember.cpp`** se encuentra un esqueleto del código a desarrollar, con comentario TODO en las funciones que debe implementar. Los patrones los debe de leer del archivo formas.dat, para mejor la interpretación de la imagen, estos valores fueron almacenados con 0 y 1 (las ecuaciones mostradas son para neuronas con valores -1 y 1). Una vez ingresados los patrones, debe calcular la matriz de conexiones, y luego hacer la lógica de evolución del sistema según la ecuación de actualización de cada neurona. El **`main`** utiliza los archivos recuerda.dat y olvida.dat para probar el código.