FACULTAD DE INGENIERIA Y CIENCIAS APLICADAS

MODELADO DE GESTOS TÁCTILES UTILIZANDO REDES ATRACTORAS ESPECIALIZADAS

AUTOR

CARLOS MARCELO DÁVILA ARMIJOS

AÑO

2019

FACULTAD DE INGENIERIA Y CIENCIAS APLICADAS

MODELADO DE GESTOS TÁCTILES UTILIZANDO REDES ATRACTORAS ESPECIALIZADAS

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniero en Sistemas de Computación e Informática

Profesor Guía

Ph.D. Mario Salvador González Rodríguez

Autor

Carlos Marcelo Dávila Armijos

AÑO

2019

DECLARACIÓN DEL PROFESOR GUÍA

"Declaro haber dirigido el trabajo, Modelado de gestos táctiles utilizando redes atractoras especializadas, a través de reuniones periódicas con el estudiante Carlos Marcelo Dávila Armijos en el semestre 201920 orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Mario Salvador González Rodríguez

Doctor en Informática

C.C. 0958376345

DECLARACIÓN DEL PROFESOR CORRECTOR

"Declaro haber revisado este trabajo, Modelado de gestos táctiles utilizando redes atractoras especializadas, del estudiante Carlos Marcelo Dávila Armijos, en el semestre 201920, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Bernarda Cecibel Sandoval Romo
Master en Ciencias de la Computación
C.C. 1709974453

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

_____

Carlos Marcelo Dávila Armijos

C.C. 1725055915

AGRADECIMIENTOS

Quiero agradecer a mi guía, corrector y a los docentes por orientarme a lo largo de este trabajo.

DEDICATORIA

Quiero dedicar este trabajo a mis padres, a mi familia y a mi novia por siempre darme su apoyo incondicional.

RESUMEN

El mundo de la interacción humano-computador cada vez se aleja más de controles físicos como botones para darle apertura a pantallas más grandes por lo que es muy importante mantener las funcionalidades de los botones, un camino son los gestos táctiles que dibuja el usuario en un lienzo, el dispositivo los interpreta y le da una respuesta.

En el siguiente documento se evalúa el uso de redes neuronales atractoras conjuntas de varios módulos de aprendizaje, basadas en el modelo de Hopfield, frente a un solo módulo con dilución, para el reconocimiento de gestos táctiles de un solo trazo. Para esto, se utiliza un *dataset* construido a partir de los 16 gestos utilizados en el trabajo "*Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes*" de Jacob Wobbrock, Andrew Wilson y Yang Li. El proceso empieza desde la extracción de los gestos representados con puntos en 2 dimensiones, estos son transformados en valores binarios, posteriormente la red hace un proceso de aprendizaje con un *dataset* y, por último, se le presenta un *dataset* con los mismos gestos, pero con variaciones menores. El resultado muestra que la red tiene un rendimiento pobre al trabajar con un solo módulo recuperando únicamente 4 gestos de los 16 enseñados, pero al modularizar la red y agregar dilución en la conexión de las neuronas se puede notar una mejora considerable primero, con 4 módulos la recuperación es de 11 de 16 patrones con asignación secuencial pero con asignación aleatoria mejora a 15 de 16 y finalmente con 8 módulos la red recupera todos los patrones con una correlación alta $m^{\mu} > 0.8$.

ABSTRACT

The world of human-computer interaction is constantly sailing apart from physical inputs like buttons in order to introduce bigger screens, but it's important to maintain buttons functionality. Therefore, tactile gestures are one solution in which the user draws a figure in a canvas/screen, the device translates it and generates an output.

The current work evaluates the usage of an ensemble attractor neural network made from multiple learning modules, based in Hopfield's model, against a simple module with dilution in their ability to recognize single stroke tactile gestures. The dataset used is based in the 16 gestures presented on "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes" by Jacob Wobbrock, Andrew Wilson y Yang Li. The process begins with the gestures' extraction, represented with 2D coordinates, then they are transformed to binary values for the neural network to learn. Lastly, the network is presented with the same dataset but with small tweaks. The results show the struggle that a single network goes through to learn the dataset, only retrieving 4 gestures out of the 16 learned, but by adding modularity and dilution to the neurons the results improve a lot, first with 4 modules the network retrieves 11 out of the 16 patterns with sequential assignation but with random assignation of the patterns, it improves by 15 out of 16 and finally with 8 modules the network retrieves every gesture with a high overlap $m^{\mu} > 0.8$.

ÍNDICE

1 INTRODUCCIÓN

El reconocimiento de gestos se ha vuelto ampliamente utilizado en varios dominios de interactividad para interpretar acciones o "*inputs*" de un usuario y realizar una tarea o acción deseada, por tal razón, la aparición de nuevas tecnologías es necesaria para cumplir las distintas necesidades de los usuarios relacionadas con interfaces naturales interactivas. (González, 2015)

Cada año las empresas de tecnología se enfocan en darle una interfaz/pantalla más grande al usuario y desaparecen los botones, esto da cabida a una interacción directa entre el usuario y una pantalla que puede recibir señales o gestos que amplían el rango de funcionalidades que el usuario puede realizar con únicamente sus dedos. Esto mejora la experiencia del usuario y se aleja de los métodos de entrada tradicionales. Una forma de que un dispositivo interprete estos gestos es utilizando redes neuronales por su capacidad de aprendizaje y reconocimiento.

Recientes trabajos han demostrado que el uso de redes neuronales atractoras conjuntas mejora la capacidad de almacenamiento de la red neuronal utilizando una estrategia de "divide y vencerás". Esto se consigue diluyendo la conectividad de la red y dividiéndola en módulos, los cuales tendrán cada uno su propio conjunto de patrones a aprender. Esto mejora su capacidad con respecto a una sola red con una conectividad más densa (González, 2017).

Las redes atractoras como máquinas de memoria para almacenar y reconocer patrones han sido aplicados como reconocimiento de gestos y modelamiento socioeconómico. Esto es posible gracias a las propiedades de la red como competición y remoción de ruido, haciendo a la red muy útil en estudios que necesiten robustez atractora (Doria, 2016).

El presente trabajo propone una red neuronal atractora conjunta basada en el modelo de Hopfield, realizada en el lenguaje de programación Python, para el

reconocimiento de gestos de dos dimensiones de un solo trazo. Los gestos fueron realizados en un *canvas,* donde fueron gráficados los gestos y levantadas las coordenadas, desarrollado en Javascript realizado por el docente de la Universidad de las Américas PhD. Jorge Pérez Medina disponible en el siguiente enlace (http://jperezmedina.eu/lightFTL/). Los gráficos para exponer los resultados fueron realizados con el lenguaje de programación R.

1.1 Justificación del Trabajo

La creciente demanda de dispositivos táctiles y pantallas más grandes han hecho desaparecer los botones en estos dispositivos, de manera que es necesario buscar alternativas para leer la entrada del usuario, interpretarla y dar una respuesta eficiente utilizando la menor cantidad de recursos, una de estas opciones es el uso de gestos en 2 dimensiones, ya sean single *touch* o *multitouch* recuperando la entrada del usuario con una pantalla o *smartphone* (Viet Le, 2018), o en 3 dimensiones utilizando cámaras para obtener información de los gestos (Nguyen Ngoc, 2018). Una de las formas de interpretar gestos es utilizando redes neuronales que entre sus aplicaciones están: predicción (Cripps, 1996), optimización (Sandeep, 2009) y aprendizaje (Ran, 2017). Por lo que, utilizando el aprendizaje y la clasificación que ofrece un modelo de redes neuronales atractoras basado en Hopfield agregando modularidad y dilución se pueden aprovechar las características de robustez de la red así como sus propiedades de complexión de patrones y limpieza de ruido en el reconocimiento sin utilizar más recursos que una red simple (Dominguez, 2012).

Se busca aplicar redes neuronales atractoras conjuntas para medir la viabilidad en el reconocimiento de gestos de 2 dimensiones, si bien los datos de entrada de un usuario no siempre son los mismos, se busca que la red pueda reconocerlos e interpretar la intensión del trazo.

1.2 Objetivo General del Trabajo

Reconocer gestos táctiles de un solo trazo utilizando redes neuronales atractoras añadiendo dilución y modularidad a la red para medir la efectividad de almacenamiento y recuperación.

1.3 Objetivos Específicos del Trabajo

- Codificar gestos realizados en 2 dimensiones para que puedan ser aprendidos por la red.
- Construir el modelo de Hopfield y agregarle modularidad y dilución para mejorar su capacidad y rendimiento.
- Evaluar el rendimiento de la red con 1 o varios módulos y tomar acción para mejorar los resultados.

2 MARCO TEORICO

2.1 Inteligencia Artificial

La inteligencia artificial se basa en construir un sistema que piense y actúe como un ser humano, apuntando a simular y extender la inteligencia humana. Entre los campos que abarca la inteligencia artificial están: *machine learning*, *computer vision*, procesamiento de lenguaje natural, reconocimiento y razonamiento, robótica y *game theory* y ética (Yanyao, 2018).

Computer Vision

Consiste en la recuperación de objetos y características de figuras en 3 dimensiones desde un conjunto de imágenes. Las imágenes pueden venir de cámaras o sensores. Una aplicación es el reconocimiento de lenguaje escrito.

Procesamiento de lenguaje natural

Se basa en la forma en la que una computadora procesa, analiza e interpreta

una gran cantidad de lenguaje natural (palabras, oraciones). Los retos de este campo son: reconocimiento de lenguaje hablado, interpretación del lenguaje natural y generación de respuestas en lenguaje natural.

Reconocimiento y razonamiento

Dedicado a representar información del mundo real de tal manera que una computadora pueda entender y utilizar esa información para realizar tareas complejas como diagnósticos médicos o conversaciones en lenguaje natural.

Machine Learning

Se centra en el uso de modelos matemáticos y estadísticos con el objetivo de que el computador realice una tarea específica de forma efectiva sin la necesidad de dar las instrucciones específicas basándose en patrones e inferencia. En esta área de la inteligencia artificial se utiliza un *dataset* de ejemplo que es tomado como referencia para tomar decisiones posteriormente.

2.2 Machine Learning

Arboles de decisiones

Es un modelo predictivo que va desde la observación de un ítem con sus generalidades hasta rasgos individuales que son representadas por las hojas de un árbol. Con la construcción de estos árboles se puede tomar decisiones basado en el conjunto de parámetros de entrada y la distribución de estos en el árbol. Se puede encontrar en aplicaciones como: buscar reglas para poses humanas atractivas (Oshita, 2017).

Máquinas de vectores de soporte

Es un conjunto de métodos de aprendizaje relacionados que utilizan clasificación y regresión de un *dataset* de aprendizaje, los clasifica en 2 categorías y un algoritmo se encarga de clasificar en cuál de las categorías pertenece. Una de las posibles aplicaciones es la clasificación y análisis de *malware* (Kruczkowski, 2014).

Redes bayesianas

Es un grafo interconectado de probabilidades en donde cada uno de los nodos representa una consecuencia y ésta se encuentra conectada con otras que puede ayudar a encontrar todos los componentes de un problema. Una aplicación de las redes bayesianas es la predicción de ataques informáticos (Okutan, 2017).

Algoritmos genéticos

Un método muy utilizado entre los 80s y 90s, es una técnica que se basa en un algoritmo de búsqueda y emula la selección natural para crear nuevos comportamientos que resuelvan de mejor manera un problema. Su uso puede ser observado en la construcción de horarios para enfermeras (You, 2010).

Redes neuronales

Es una colección de procesadores de señales simples interconectados que emulan el procesamiento biológico de los seres vivos. Una red neuronal es representada como un grafo direccionado, en el cual los nodos representan los elementos procesados, los vértices representan las conexiones y las flechas en los vértices representan el flujo normal de la información. Como se muestra en la figura 1, una imagen se convierte en un patrón binario y es almacenado en la red, cada bit almacenado en una neurona.

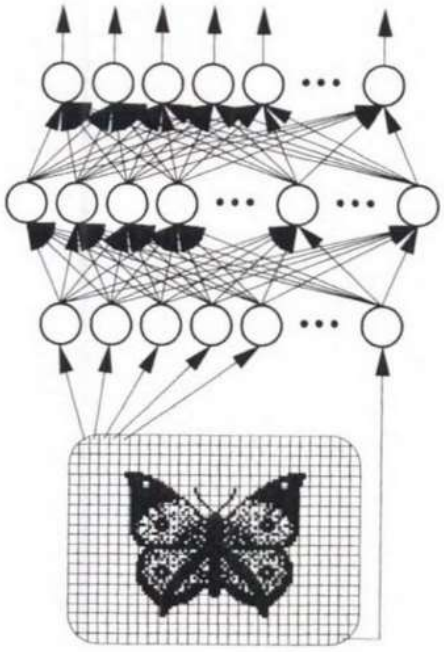


Figura 1. Procesamiento de una imagen utilizando una red neuronal.
Tomado de: (Skapura, 1996).

2.3 Redes neuronales atractoras

Una red neuronal busca simular de manera general el proceso de toma de decisiones de un sistema nervioso biológico. Al estimular la red con un patrón, estos se almacenarán como estados de memoria (atractor) y si un estímulo de entrada está dentro de la cuenca de atracción (contiene condiciones iniciales parecidas), la red tratará de reconstruir el patrón inicial que está almacenado como un estado de memoria (Rodríguez, 2011) como se muestra en la figura 2.

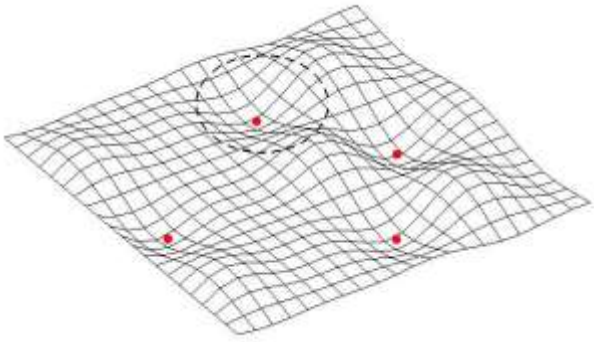


Figura 2. Cuencas de atracción.
Tomado de: Scholarpedia (Eliasmith, 2007)

2.4 Modelo de Hopfield

El estado de una neurona en una red de N unidades está definida, en cualquier tiempo t discreto, como un conjunto de tamaño N de variables binarias $\tau^t = \tau_i^t \in 0,1; i = 1, \dots, N$, donde 1 y 0 representan estados activos e inactivos respectivamente. El objetivo de la red será recuperar un grupo de patrones, que en este caso serán gestos 2D, $\{\eta^\mu,\ \mu = 1, \dots, P\}$, que han sido almacenadas en el proceso de aprendizaje. Cada patrón corresponde a un punto atractor fijo y el estado de recuperación cumple con $\tau^t = \eta^\mu$, para un tiempo t suficientemente largo. Los patrones están codificados como un conjunto binario de variables $\eta^\mu = \{\eta_i^\mu \in 0,1; i = 1, \dots, N\}$ con una probabilidad resultante

$$p(\eta_i^\mu = 1) = a, \quad p(\eta_i^\mu = 0) = 1 - a, \qquad \text{Ecuación 1}$$

donde $a \in (0,1)$ se refiere al promedio correspondiente del índice de actividad del *datasets* de patrones, como los tratados en redes codificadas sesgadas (Dominguez, Structured information in sparse-code metric neural networks, 2012).

Los acoplamientos entre las neuronas i y j son obtenidos mediante una matriz de adyacencia

$$J_{ij} \equiv C_{ij} W_{ij}, \quad \text{Ecuación 2}$$

donde la matriz de topología $C = \{C_{ij}\}$ describe la estructura de conectividad de la red neuronal y $W = \{W_{ij}\}$ es la matriz con los pesos de aprendizaje. La matriz de topología corresponde a una red aleatoria simétrica (Albert, 2002), con una red de grado K, que cada nodo está conectado con K nodos. La red también está caracterizada por un parámetro de dilución $\gamma = K/N$, que representa el índice de la red neuronal artificial.

La recuperación de un patrón de gesto se consigue a través de la dinámica de las neuronas

$$\tau_i^{t+1} = \Theta(h_i^t - \theta_i^t), \quad i = 1, \dots, N, \quad \text{Ecuación 3}$$

donde

$$h_i^t \equiv \frac{1}{K}\sum_j J_{ij}\frac{\tau_j^t - q_j^t}{\sqrt{Q_j^t}} \qquad \text{Ecuación 4}$$

indica el campo local en la neurona i y el tiempo t y θ_i, es el umbral de actividad., y también se utiliza la función de cambio de tiempo:

$$\Theta(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0. \end{cases} \qquad \text{Ecuación 5}$$

El promedio de actividad de un vecindario de una neurona i, $q_i^t = <\tau^t>_i$, y su correspondiente varianza $Q_i^t = Var(\tau^t)_i = <(\tau^t)^2>_i - <\tau^t>_i^2$ están presentes en las ecuaciones anteriores. El promedio del vecindario esta definido como $<f^t>_i \equiv \sum_j \quad C_{ij} f_j^t / K$.

Las variables normalizadas son usadas, la dependencia del tiempo y el momento es implícita:

$$\sigma \equiv \frac{\tau - q}{\sqrt{Q}}, q \equiv <\tau>,\ Q \equiv \mathrm{Var}(\tau) = q(1-q) \qquad \text{Ecuación 6}$$

$$\xi \equiv \frac{\eta - a}{\sqrt{A}},\ a \equiv <\eta>,\ A \equiv Var(\eta) = a(1-a) \qquad \text{Ecuación 7}$$

donde a y q son el patrón y la actividad neuronal, respectivamente. Los promedios realizados en este trabajo a través de diferentes grupos los cuales son indicados en cada caso. El modelo neuronal binario uniforma es recuperado cuando $a = 1/2$.(Dominguez, 2019)

En términos de estas variables normalizadas, la dinámica de la neurona puede ser escrita como

$$\sigma_i^{t+1} = g(h_i^t - \theta_i^t, q_i^t) \qquad \text{Ecuación 8}$$

$$h_i^t \equiv \frac{1}{K}\sum_j J_{ij}\sigma_j^t,\ i = 1, \ldots, N \qquad \text{Ecuación 9}$$

donde la función de ganancia está definida por

$$g(x,y) \equiv [\Theta(x) - y]/\sqrt{y(1-y)} \qquad \text{Ecuación 10}$$

La matriz de pesos W es actualizada de acuerdo a la regla de aprendizaje de Hebb,

$$W_{ij}^{\mu+1} = W_{ij}^{\mu} + \xi_i^{\mu}\xi_j^{\mu} \text{ (online)}, \quad W_{ij} = \sum_{\mu=1}^{P} \xi_i^{\mu}\,\xi_j^{\mu} \text{ (offline)}. \qquad \text{Ecuación 11}$$

Vale la pena señalar que la regla de aprendizaje *offline* no es iterativa, es decir, todos los patrones deben ser aprendidos antes de empezar el proceso de recuperación. Para contrastar, la regla de aprendizaje *online* es iterativa y el

proceso de recuperación se repite por cada uno de los nuevos patrones aprendidos por la red (González, 2015). Los pesos, en el aprendizaje *online* y *offline*, empiezan en $W_{ij}^{0} = 0$ y después de una cantidad P de pasos de aprendizaje, llegan a un valor $W_{ij} = \sum_{\mu}^{P} \xi_i^{\mu} \xi_j^{\mu}$. La red aprende $P = \alpha K$ patrones donde α es el índice de carga, el cual mide la capacidad de recuperación de la red. Un umbral es necesario para mantener la actividad neuronal cercana a la de los patrones aprendidos, $\theta_i^t(a) = \frac{1-2a}{2\sqrt{A}}$ es utilizado, donde a corresponde a la actividad promedio del conjunto de patrones por aprender, y $A = a(1-a)$ a la varianza de la actividad (Doria, 2016).

Con el objetivo de caracterizar la habilidad de recuperación de los módulos de la red se utiliza la superposición (*overlap*),

$$m_{\equiv} \frac{1}{N} \sum_{i}^{N} \xi_i \ \ \sigma_i', \qquad \text{Ecuación 12}$$

la cual es la correlación estadística entre el patrón aprendido ξ_i y el estado neural σ_i. Para $m = 1$, este tiene una recuperación perfecta del patrón mediante la red, para $m = 0$ no se ha recuperado nada, y para valores intermedios el patrón es recuperado con ruido. Por ello, el valor de m es una medida de calidad de recuperación del patrón procesado por la red (Dominguez, 2019).
El estado inicial de la red $\tau^{t=0}$ y la correspondiente transformación interna $\sigma_i^{t=0}$, es un estado ruidoso cargado desde un conjunto de patrones de prueba donde $m^{t=0} < 1$, por ejemplo, está permitido el ruido inicial en el principio de la fase de recuperación. Esto significa que el conjunto de prueba es el mismo que el almacenado en la red, pero con una variación ingresada por el usuario (González, Retrieval of noisy fingerprint patterns using metric attractor networks, 2014).

2.4.1 Redes atractoras conjuntas

Una red atractora conjunta o EANN, del inglés *Ensemble Artificial Neural Network*, es una colección de redes neuronales que realizan el mismo trabajo, cada una de las redes tiene sus propias características como pesos y

conexiones. Esta propiedad es muy útil al utilizar algoritmos de aprendizaje incremental, puesto que cada red aprende un conjunto de patrones diferente aumentado la capacidad general del modelo (Hansen, 1990). Como se muestra en la figura 3, los patrones se dividen en igual número para cada uno de los módulos balanceando la cantidad de patrones aprendidos por cada uno de los módulos.

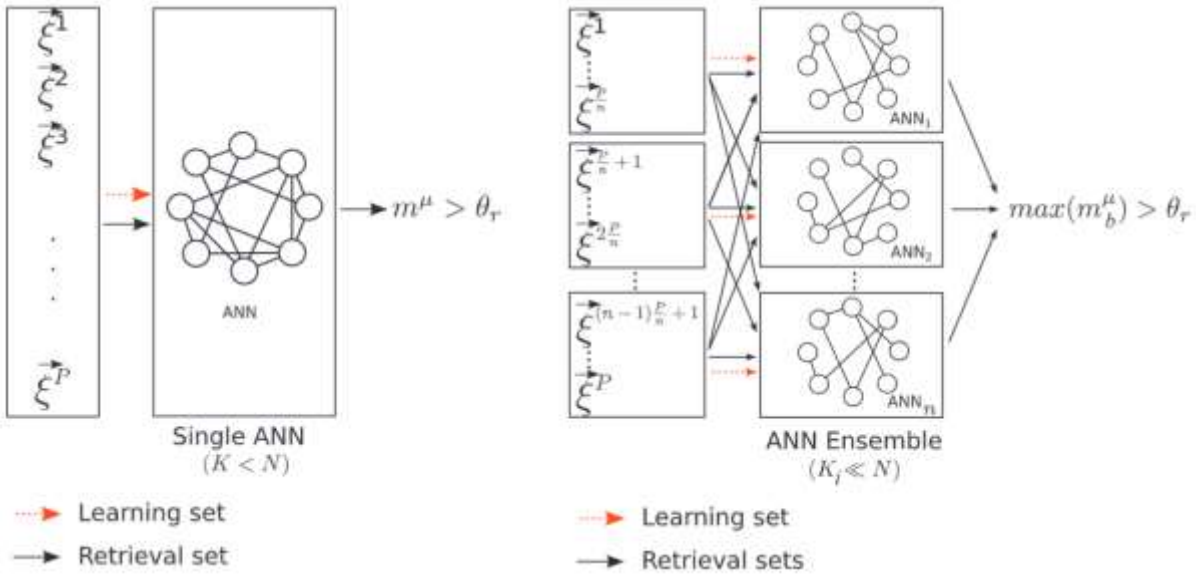


Figura 3. Red atractora de un solo módulo frente a red atractora conjunta
Tomado de: (González, 2017)

2.5 Reconocimiento de gestos

Un gesto es generalmente definido como una secuencia de puntos teniendo un inicio y un final, caracterizado por tener una escala, dirección, posición, rotación entre otros.

El reconocimiento de gestos es un campo de la inteligencia artificial que busca interpretar gestos que involucren alguna parte del cuerpo humano utilizando modelos matemáticos. La forma en la que las computadoras reciben un gesto varía entre pantallas, cámaras u otros periféricos. Estas entradas de los usuarios finales pueden contener ambigüedades que el modelo debe discernir y dar una respuesta esperada (Mitra, 2007).

El proceso de reconocimiento se resume en la comparación de la secuencia de puntos de un gesto candidato con la referencia a un gesto que es parte del *dataset* inicial que aprendió la red. Normalmente la comparación está basada en obtener la suma de las distancias de cada uno de los gestos comparados, y para esto se han propuesto diferentes distancias como: la distancia euclidiana usada por el algoritmo $1, DTW (*Dynamic Time Warping*) utilizada por programación dinámica en combinación con memoria jerárquica y la distancia de Levenshtein (Beuvens, 2012)

3 Desarrollo

El presente proyecto se encuentra dentro del campo de la inteligencia artificial, enfocado en *machine learning* utilizando redes neuronales atractoras.
Con el objetivo de representar el modelo propuesto es una aplicación de consola se utilizó el lenguaje de programación Python por su amplia gama de librerías de estadística y matemáticas.

3.1 Dataset

El *dataset* fue realizado en el *canvas* realizado por el Docente de la Universidad de las Américas PhD. Jorge Pérez Medina disponible en el siguiente enlace http://jperezmedina.eu/lightFTL/, en dicho *canvas* se grafican los gestos y se recupera un archivo txt donde se almacenan 1500 puntos en coordenadas cartesianas. Estos archivos deben ser codificados en binario para que la red neuronal pueda aprenderlos.

Para la codificación de los patrones se utilizó una librería de *scipy* para hacer un muestreo de los puntos que conformen el gesto en el caso de existir más de 501 coordenadas. En el mismo paso de la codificación se asigna el orden de aprendizaje de los gestos que en un principio fue de manera secuencial como se presenta en el *dataset* pero posteriormente se le dio un orden aleatorio generado por la librería *numpy*. El código de la codificación se encuentra en el anexo 2.

Los gestos en estado puro se pueden representar como una tupla de puntos continuos $\pi^{\mu} = \{(x_i' y_i\ ) \in R; i = 1, \dots, S\}$, donde $S = 1500$ es el número de puntos capturados para cada gesto. Para ser procesado por la red, se realiza un muestreo de tamaño $S = N/2 + 1$ a la señal del gesto π, donde el tamaño de la red es $N = 1000$. El valor de $N = 1000$ fue escogido por el tamaño de la red/patrones (nodos/sitios) como un valor mínimo para permitir módulos del conjunto diluidos.

La binarización ocurre como es descrita esquemáticamente en la figura 4. La codificación tiene como resultado una cadena binaria de tamaño $N = 1000$, donde la mitad de lo sitios corresponden al muestreo de x y la otra mitad a y intercalado en una tupla binarizada.

El panel izquierdo de la figura 4 muestra el cambio de dirección del trazo y su respectivo valor en binario. Si x, o y, aumenta, el resultado de la codificación es $\eta_i = 1$, caso contrario $\eta_i = 0$. Un umbral ϕ es utilizado para codificar los trazos que cambian en (x, y) pero no en ambos, es decir, trazos horizontales o verticales. Si el cambio ($\delta x\ o\ \delta y$) es menor que ϕ, $\eta_{yi} = yr, yr \sim Bern(p = 0.5)$ codificará trazos verticales, y $\eta_{xi} = xr, xr \sim Bern(p = 0.5)$ codificará horizontales. En este trabajo, el umbral es de 0.01, fue seleccionado de manera empírica para que funcione de manera correcta para el *dataset* de $1. Aquí $xr, xr \sim Bern(p)$ se refiere a la variable sigue una distribución de Bernoulli con probabilidad p. Para cambios $\delta x\ o\ \delta y$ que son mayores a ϕ, el trazo (xi, y_i) se considera que aumenta o decrece con respecto a al punto anterior (x_{i-1}, y_{i-1}) y son codificados de tal manera que: $x+, y+ \Rightarrow 1,\ x-, y- \Rightarrow 0$. A la derecha de la figura 4, se puede observar un ejemplo de un gesto y el resultado de su codificación. El punto negro indica el inicio del gesto. Nótese que el primer punto se toma como referencia para comprobar el cambio que sufre el trazo en el proceso de binarización. Se puede apreciar que cuando x aumenta y y la tupla resultante es 10. Cuando el trazo disminuye en x pero no incrementa ni disminuye en y el resultado es $0yr$, donde $yr \sim Bern(p = 0.5)$. Finalmente,

cuando el gesto disminuye en x, y la cadena resultante es 11 para cada punto.

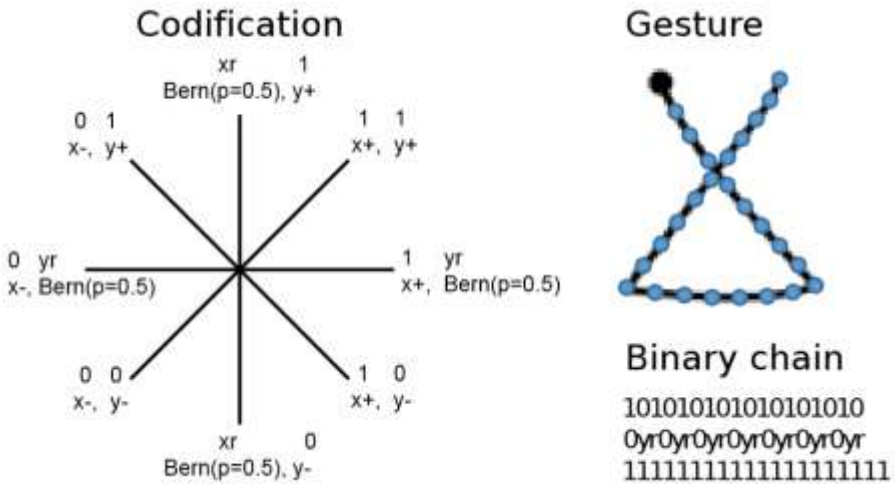


Figura 4. Izquierda: Codificación binaria. Derecha: Ejemplo gesto binarizado

El *dataset* de gestos está compuesto de $P = 16$ como se muestra en la figura 5. En el panel izquierdo se muestran los gestos recuperados utilizando el trabajo de (Wobbrock, 2007). El panel derecho corresponde a los patrones binarizados que serán almacenados en la red. Nótese que los gestos binarizados de la derecha esta graficados en (x, y) tomando como referencia la suma acumulativa de cada uno de los patrones binarios $\eta^{\mu} \times 2 - 1$. Además, el punto negro representa el punto inicial del gesto. El *dataset* $1 fue utilizado porque sus gestos tienen características similares. Por ejemplo, el gesto 2 y 12 se diferencian en la dinámica con la que se realiza el gesto, es decir, se diferencian en orientación y rotación. Gracias a esto, este es un *dataset* estándar que puede probar la habilidad de reconocimiento de la red atractora conjunta.

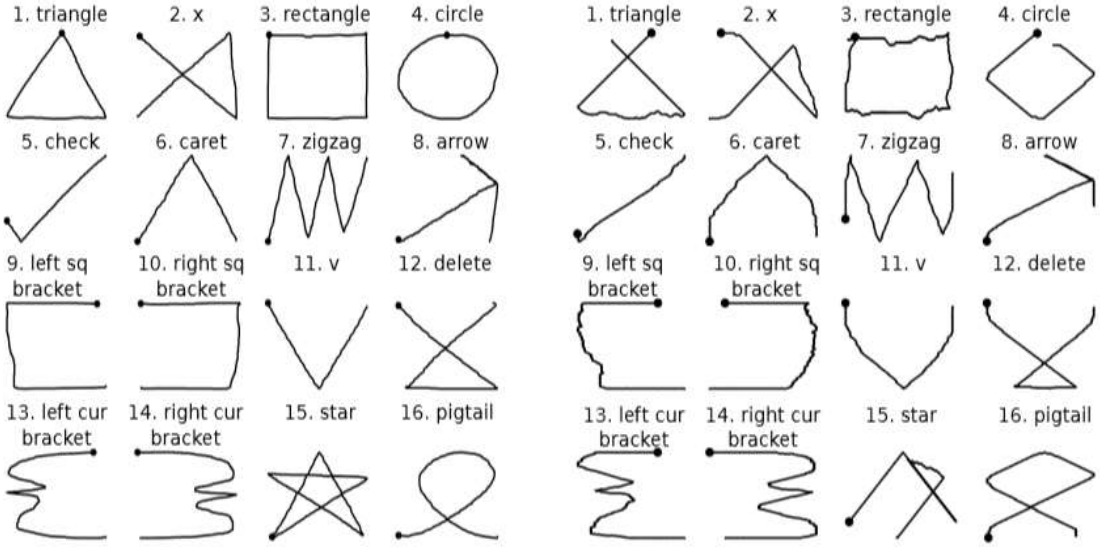


Figura 5. Patrones binarizados

3.2 El modelo

El modelo utilizado para realizar este trabajo es basado en el modelo de Hopfield que consiste en una red neuronal atractora, en los siguientes puntos se explicará la codificación de los módulos, topología y dinámica de aprendizaje y recuperación.

Otro uso de la librería *numpy* fue para trabajar con arreglos y realizar operaciones entre matrices. Estas operaciones entre matrices fueron utilizadas para actualizar el estado de la red en cada uno de los tiempos de procesamiento y para obtener el *overlap* entre los patrones que son aprendidos por la red y el patrón resultante del procesamiento que representa la calidad de recuperación del gesto. El código del programa principal se encuentra en el anexo 3.

Una representación esquemática de una sola red neuronal se encuentra a la izquierda de la figura 6. El índice de conectividad γ está diluido con $K < N$. Un conjunto de gestos ξ es presentado a la red en la fase de aprendizaje, representada con la flecha roja. También, este conjunto de gestos está presente en la fase de recuperación con el propósito de evaluar la habilidad de

procesamiento de la red en términos de carga de patrones recuperados α, y la calidad de recuperación m. Esto es representado por la flecha negra.

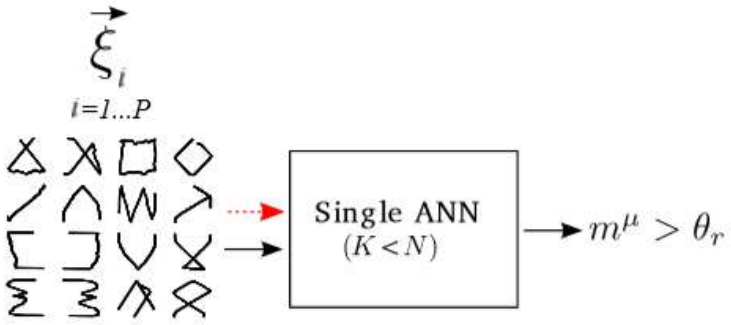


Figura 6. EANN.

En la parte derecha de la figura 7, se muestra una representación esquemática de los n módulos de una red neuronal conjunta. La conectividad de cada uno de los módulos b de ANN_b está altamente diluido con $K_b \ll, b \in \{1, \dots, n\}$, y el índice de conectividad $\gamma_b = K_b/N$. En este trabajo la red individual $n = 1, K = 200, \gamma = 0.2$ es comparada con una EANN de módulos $n = 4, K_b = 50, \gamma_b = 0.05$. El costo computacional de la ANN individual y la EANN son los mismos, siendo $K = K_b \times n$. El conjunto de patrones está dividido en subconjuntos diferentes de tamaño uniforme $P_b = P/n$, y cada subconjunto es aprendido por su correspondiente módulo de la ANN representado por una flecha roja punteada. Por ejemplo, $\{\vec{\xi^\mu},\ \mu = 1, \dots, P/n\}$ para el primer módulo $ANN_b, b = 1$ como se muestra en figura 7. La flecha negra de la misma figura representa la fase de recuperación, en el cual todos los subconjuntos de patrones son procesados por todos los módulos de la EANN para discriminarlos entre ellos. Los patrones objetivo son considerados como recuperados por el módulo de ANN que tenga el *overlap* más alto sobre el umbral de recuperación θ_r. Por ejemplo, $max(m_b^\mu) > \theta_r$. Por temas de comparación, se asume que θ_r toma el mismo valor para cada uno de los componentes del conjunto de redes, así como para la red individual. Por último, también se realizó una configuración $n = 8, K_b = 50, \gamma_b = 0.05$ para evaluar la mejora de la recuperación.

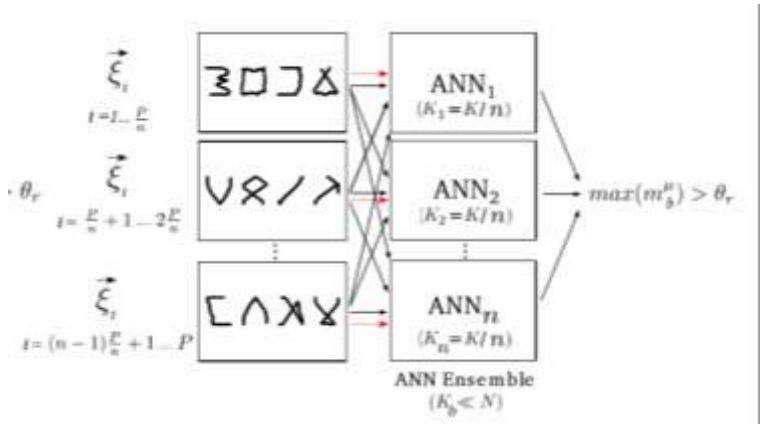


Figura 7. EANN.

4 Resultados

Para presentar los resultados se utilizó el lenguaje R, *script* presentado en el anexo 4, el cual lee los resultados del procesamiento contenidos en un archivo csv y grafica con un color distinto cada una de las subredes.

En esta sección se presenta el rendimiento de la red comparando un módulo atractor con una EANN. El rendimiento se muestra en términos de la correlación m para cada patrón del *dataset*. La calidad de recuperación de cada uno de los gestos con ruido y el estado final representa el trazo del gesto. La red conjunta mejoró el reconocimiento con relación al uso de una sola red y demostró robustez ante la similitud de los gestos y la aparición de ruido. La asignación aleatoria fue necesaria para mejorar la calidad de recuperación de los gestos que poseían una alta similitud, esto abre la posibilidad de optimizar el orden de aprendizaje de los gestos en los distintos módulos de la red.

El rendimiento se mide utilizando el parámetro de correlación, donde un valor de $m = 1$ significa una recuperación perfecta y $m = 0$ significa que no hay recuperación alguna, y los valores intermedios, una recuperación con ruido de $1 - m$.

El promedio de actividad (cantidad de 1s y valores activos) del *dataset* es $a =$

0.5803125, media de la matriz de patrones, lo que resulta en $\theta(a) = -0.17093$ para la dinámica de recuperación utilizando para para mantener la actividad neuronal cercana a la de los patrones aprendidos. El promedio de correlación de los gestos es $\langle m \rangle_{\mu\nu} = 0.1895$ lo cual es elevado, lo que hace que el ruido presente entre patrones sea alto.

4.1 Rendimiento de recuperación de la red

En la figura 8 se representa $n = 1, K = 200, \gamma = 0.2$ con recuperación *online*. El rendimiento de la recuperación *online* donde los patrones son aprendidos y probados uno por uno. Se puede apreciar que el punto crítico entre recuperación alta y baja (m) está alrededor de la mitad del *dataset* (8 patrones). A partir de este punto, la red ya no devuelve nada útil, solamente ruido.

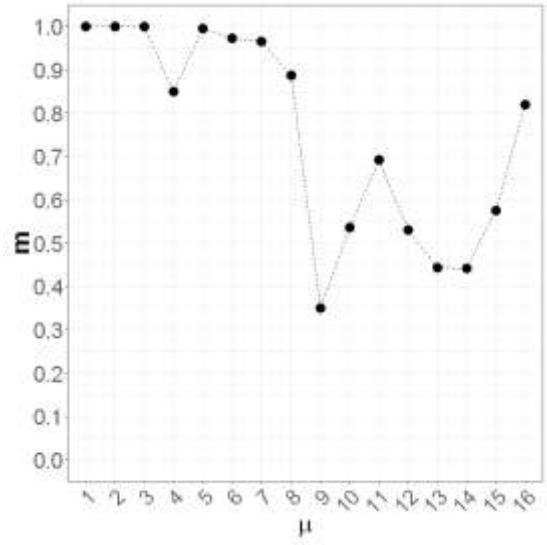


Figura 8. Rendimiento de la red con $n = 1, K = 200, \gamma = 0.2$ y recuperación *online*.

En la figura 9 se muestra la recuperación *offline*, con una red de parámetros $n = 1, K = 200, \gamma = 0.2$, de todo el *dataset* donde todos los patrones $P = 16$ son aprendidos en un solo paso y después evaluados. Solo 4 patrones ($\mu = \{1,4,5,16\}$) son recuperados con una correlación alta ($m^{\mu} > 0.8$). El resto de los patrones no son reconocidos por la red por el grado de similitud que tienen entre sí.

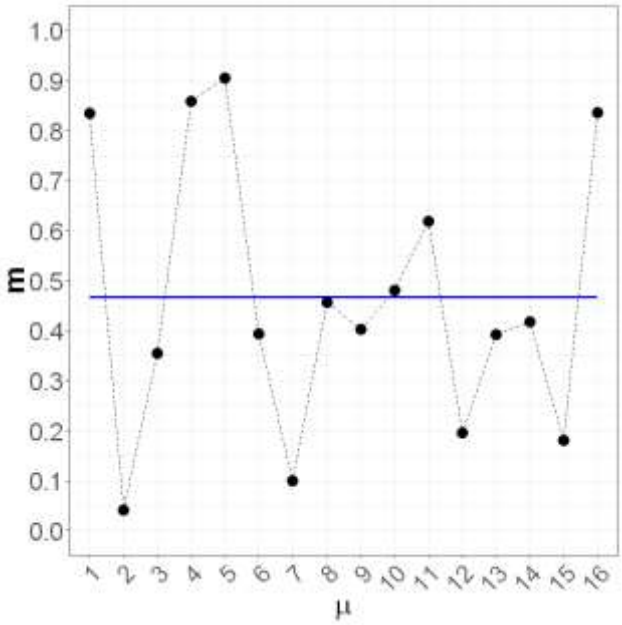


Figura 9. Rendimiento de la red con $n = 1, K = 200, \gamma = 0.2$ y recuperación *offline*

La figura 10 muestra una red conjunta de $n = 4, K_b = 50, \gamma_b = 0.05$. Los 16 patrones son divididos en 4 subconjuntos secuenciales y cada módulo aprende 4 patrones. Comparado con una red individual, el rendimiento aumenta de 4 a 11 patrones recuperados con un *overlap* alto ($m^{\mu} > 0.8$). Los gestos no recuperados son debido a que los subconjuntos de 4 gestos se parecen entre sí, esto se puede mejorar con la asignación aleatoria entre módulos y patrones.

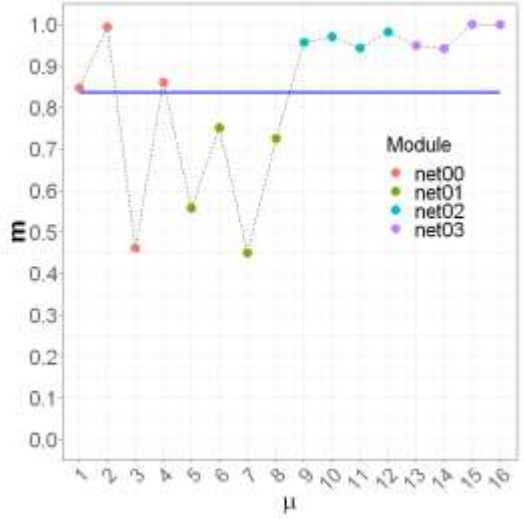


Figura 10. Rendimiento de la red con $n = 4, K = 200, \gamma = 0.2$, recuperación *offline* y asignación secuencial.

La figura 11 representa la recuperación *offline* de una red conjunta de $n = 4, K_b = 50, \gamma_b = 0.05$ con una asignación aleatoria de patrones, en lugar de secuencial como en la figura 9. Este cambio en el orden permite a la red recuperar casi todo el *dataset* ($P = 15$) con una correlación alta $m_b^\mu > 0.8$. La mejora de la recuperación también se puede observar en el promedio de recuperación de todos los patrones representada por la línea azul $\langle m * \rangle_\mu = 0.9519$,. Este valor es mayor a $\langle m * \rangle_\mu = 0.8371$ y $\langle m * \rangle_\mu = 0.4666$ obtenidos por la red con asignación de gestos del *dataset* secuencial y la red con un solo módulo respectivamente. La asignación aleatoria fue seleccionada de acuerdo con una búsqueda sencilla de combinaciones posibles. Una combinación fue obtenida cuando no se aparecía una mejora en el promedio de recuperación ($\langle m \rangle_\mu$) después de un número de pasos. A consecuencia de esto el modelo conjunto puede permitir la optimización de la asignación de patrones en los módulos para mejorar el rendimiento de la recuperación de la red, ya sea en términos de número de patrones recuperados (capacidad) y la calidad de recuperación (*overlap*).

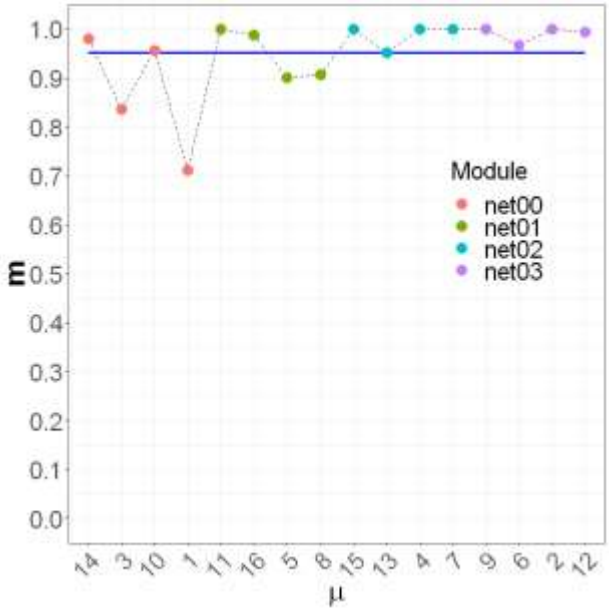


Figura 11. Rendimiento de la red con $n = 4, K = 200, \gamma = 0.2$, recuperación *offline* y asignación aleatoria.

Por último, en la figura 12, se presenta una red conjunta de $n = 8, K_b = 50, \gamma_b =$

0.05. Se puede apreciar que todos los patrones son recuperados perfectamente, pero se está desperdiciando la capacidad de almacenamiento de la red puesto que cada módulo aprende únicamente 2 patrones. Cuando podría aprender tranquilamente 6 patrones que sean totalmente diferentes entre sí.

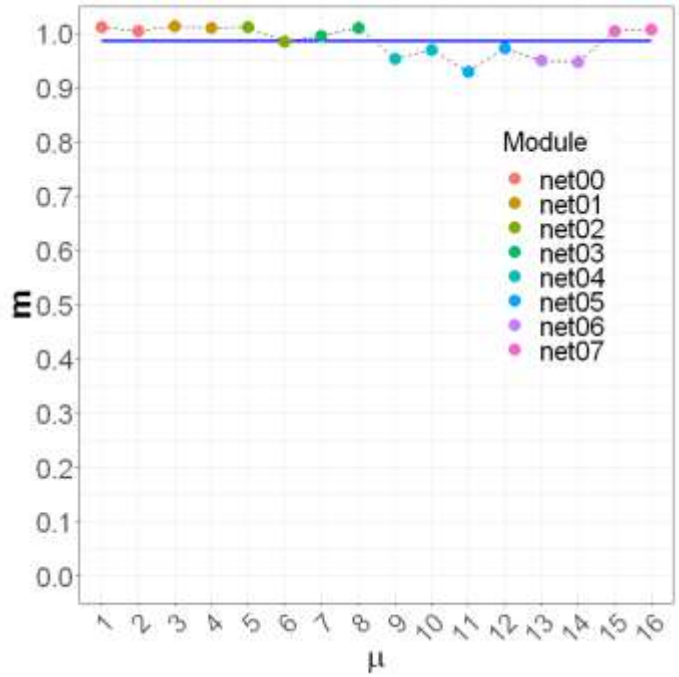


Figura 12. Rendimiento de la red con $n = 8, K = 200, \gamma = 0.2$, recuperación *offline* y asignación aleatoria.

Como se muestra en la tabla 1, la recuperación con $n = 1$ la recuperación es deficiente con un valor de $\langle m^* \rangle_\mu = 0.4666$ con un ruido de $1 - 0.4666 = 0.5334$. Al utilizar $n = 4$ con asignación aleatoria, la recuperación mejora casi el doble con $\langle m^* \rangle_\mu = 0.8371$ y un ruido de $1 - 0.8371 = 0.1629$. Cuando se utiliza $n = 8$ con asignación aleatoria, los resultados mejoran un poco más $\langle m^* \rangle_\mu = 0.9862$ y con un ruido únicamente de $1 - 0.9862 = 0.0138$. Tomando estos datos en cuenta, la mejor recuperación se obtiene de la red con 8 módulos, pero realmente se está desperdiciando todo el potencial de la red porque únicamente se almacenan 2 gestos por red. Por esta la razón la mejor configuración es $n = 4, K = 200, \gamma = 0.2$ porque a la red se la exige a aprender 4 gestos por módulos y tiene un índice de recuperación alto.

Tabla 1.

Comparación general de los modelos.

n	γ	$\langle m * \rangle_{\mu}$
1	0.2	0.4666
4	0.2	0.8371
8	0.2	0.9862

4.2 Calidad de recuperación de gestos

Considerando que la mejor configuración es $n = 4, K = 200, \gamma = 0.2$ se profundizarán más en los resultados obtenidos.

En el Anexo 1, se representan los gestos aprendidos y recuperados por la red conjunta $n = 4, K_b = 50, \gamma_b = 0.05$ con asignación aleatoria de gestos. En la figura 13 se presenta una muestra del anexo 1 donde la figura con título "*Stored*" corresponde al patrón almacenado en la fase de aprendizaje, la figura con la cabecera "*Tested*" corresponde al estado inicial de la red en la fase de recuperación ($\boldsymbol{m^0}$) y por último la figura con el título "*Retrieved*" representa el estado final de la red $\boldsymbol{m}^*$ después de $\boldsymbol{t = 100}$.

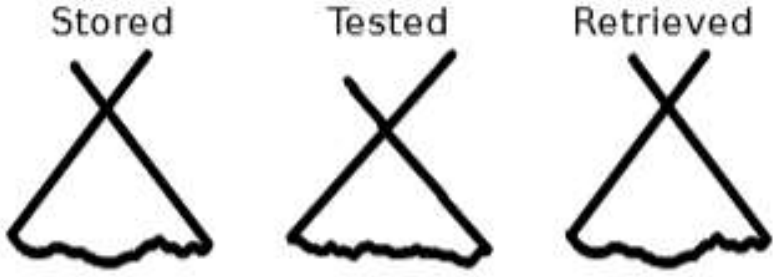


Figura 13. Gesto "*triangle*" utilizado en la red.

Los *overlaps* entre los gestos aprendidos y de evaluación son presentados en la figura 14 en la columna m^0. La correlación promedio entre los gestos aprendidos y de prueba es $\langle m^0 \rangle_{\mu} = 0.7483$ con un valor mínimo de 0.5961, que corresponde al gesto "*star*" con $\mu = 15$ en el panel izquierdo de la figura 14. Y 0.8932, el valor máximo, correspondiente al gesto "*circle*" $\mu = 15$ que se encuentra en el panel

derecho de la figura 14. Esto significa que los estados iniciales de los gestos de prueba tienen un nivel de ruido considerable en comparación de los gestos aprendidos. Los *overlaps* de recuperación se muestran en la figura 14 en la columna m^* estos valores corresponden a la representación gráfica de la figura 11. Los gestos son recuperados efectivamente con una correlación promedio de $\langle m^* \rangle_\mu = 0.9519$ y un valor mínimo de 0.7119, para el gesto "*triangle*". Mientras que el valor máximo es 1, para los gestos "x", "*circle*", "*zigzag*", "*left square bracket*", "v" y "*star*". Como se puede observar, los módulos de la red conjunta son robustos frente al ruido inicial con una gran cuenca de atracción como se puede mostrar en el gesto de la "*star*" ($\mu = 15$) consiguiendo una recuperación perfecta del gesto ($m^* = 1.0$).

μ	Name	m^0	m^*	μ	Name	m^0	m^*
1	triangle	0.8346	0.7119	9	left-square bracket	0.6216	1.0000
2	x	0.8561	1.0000	10	right-square bracket	0.6073	0.9560
3	rectangle	0.6337	0.8368	11	v	0.8187	1.0000
4	circle	0.8932	1.0000	12	delete	0.7858	0.9941
5	check	0.8624	0.9014	13	left-curly bracket	0.6048	0.9517
6	caret	0.8263	0.9663	14	right-curly bracket	0.6584	0.9797
7	zigzag	0.7454	1.0000	15	star	0.5961	1.0000
8	arrow	0.8663	0.9063	16	pigtail	0.7624	0.9884

Figura 14. Correlación inicial y final de los gestos.

5 CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Este trabajo mejora el reconocimiento de gestos de $1 ya que dicho trabajo tiene dificultades para interpretar gestos complejos porque realiza un mapeo de los puntos que componen el gesto mientras que el uso de EANN no es afectado por el tamaño del gesto porque únicamente se deben crear más neuronas que almacenen los puntos binarizados.

Se evaluó el rendimiento del modelo EANN para el rendimiento de aprendizaje y recuperación de un *dataset* de gestos 2D de un solo trazo. La EANN con 4 y 8 módulos supero el rendimiento *offline* de una red de un solo módulo en la recuperación de gestos.

Una asignación aleatoria de los gestos en los módulos del ensamblado mejoró la recuperación de gestos altamente correlacionados, permitiendo a la EANN recuperar todos los gestos, a excepción de uno, con una alta calidad ($\$m\^{}\{*\} > 0.9\$$) al contrario de una red individual. Consecuentemente, una optimización en el ingreso de los gestos puede ser realizado con el objetivo de maximizar la calidad de recuperación y la cantidad de patrones almacenados por la red.
Ésta correcta asignación de gestos en los módulos es crucial cuando se trabaje con *datasets* más grandes. El ensamblado probó ser robusto cuando se presenta un ruido tan alto como 0.4 y por ende recuperó a la perfección el gesto. Un posible uso para este modelo es el reconocimiento de firmas porque para esto se requiere precisión y velocidad de reconocimiento sin que se exija la velocidad en el aprendizaje.

5.2 Recomendaciones

Se recomienda utilizar el modelo descrito para reconocer gestos más complejos que requieran una mayor cantidad de neuronas e implementarlos sistemas de reconocimiento de gestos táctiles con el objetivo de mejorar la seguridad y

precisión del reconocimiento de patrones.

También se recomienda el uso de un *dataset* de prueba con variantes más notables como rotación, traslación, orientación y escala para medir la robustez ante estos cambios más notables. Finalmente, se puede especializar cada uno de los módulos de la red con un gesto único para ampliar la cuenca de atracción del gesto y las variantes puedan ser más notables.

REFERENCIAS

Albert, R. (2002). *Statistical mechanics of complex networks.* Recuperado el 20 de mayo de 2019, de http://barabasi.com/f/103.pdf

Beuvens, F. (2012). *Designing graphical user interfaces integrating gestures.* Recuperado el 28 de mayo de 2019, de http://www.usixml.org/servlet/Repository/beuvens-sigdoc2012-262.pdf

Cripps, A. (1996). *Using artificial neural nets to predict academic performance.* Recuperado el 12 de mayo de 2019, de https://dl.acm.org/citation.cfm?id=331137

Dominguez, D. (2012). *Structured information in sparse-code metric neural networks.* Recuperado el 8 de abril de 2019, de https://www.researchgate.net/publication/233397982_Structured_information_in_sparse-code_metric_neural_networks

Dominguez, D. (2019). *Structured information in small-world neural networks.* Recuperado el 13 de mayo de 2019, de https://www.researchgate.net/publication/24357597_Structured_information_in_small-world_neural_networks

Doria, F. (2016). *Structured patterns retrieval using a metric attractor network: Application to fingerprint recognition.* Recuperado el 23 de abril de 2019, de https://www.sciencedirect.com/science/article/pii/S0378437116002685

Eliasmith, C. (2007). *Scholarpedia.* Recuperado el 8 de abril de 2019, de Hopfieldattractor: http://www.scholarpedia.org/w/images/5/56/Hopfieldattractor.jpg

González, M. (2014). *Retrieval of noisy fingerprint patterns using metric attractor networks.* Recuperado el 10 de mayo de 2019, de https://www.researchgate.net/publication/265863885_Retrieval_of_noisy_fingerprint_patterns_using_metric_attractor_networks

González, M. (2015). *Modeling sustainability report scoring sequences using an attractor network.* Recuperado el 15 de mayo de 2019, de

https://www.sciencedirect.com/science/article/pii/S092523121500621
9

González, M. (2017). *Capacity and Retrieval of a Modular Set of Diluted Attractor Networks with Respect to the Global Number of Neurons.* Recuperado el 10 de mayo de 2019, de https://link.springer.com/chapter/10.1007/978-3-319-59153-7_43

González, M. (2017). *Increase attractor capacity using an ensembled neural network.* Recuperado el 17 de mayo de 2019, de https://www.sciencedirect.com/science/article/pii/S095741741630670
4

Hansen, L. K. (1990). *Neural network ensembles.* Recuperado el 12 de mayo de 2019, de https://ieeexplore.ieee.org/document/58871

Kruczkowski, M. (2014). *Support Vector Machine for Malware Analysis and Classification.* Recuperado el 16 de mayo de 2019, de https://ieeexplore.ieee.org/document/6927654

Mitra, S. (2007). *Gesture recognition: A survey.* Recuperado el 25 de abril de 2019, de https://ieeexplore.ieee.org/document/4154947

Nguyen Ngoc, H. (2018). *A Real-time Multimodal Hand Gesture Recognition via 3D Convolutional Neural Network and Key Frame Extraction.* Recuperado el 12 de abril de 2019, de https://dl.acm.org/citation.cfm?id=3278314

Okutan, A. (2017). *Predicting cyber attacks with bayesian networks using unconventional signals.* Recuperado el 29 de abril de 2019, de https://www.researchgate.net/publication/317012234_Predicting_cyber_attacks_with_bayesian_networks_using_unconventional_signals

Oshita, M. (2017). *Finding rules of attractive human poses using decision tree and generating novel attractive poses.* Recuperado el 10 de mayo de 2019, de https://www.semanticscholar.org/paper/Finding-rules-of-attractive-human-poses-using-tree-Oshita-Yamamura/8b1898f22166ef51af94f9c639661a5ffc52958c

Ran, X. (2017). *Delivering Deep Learning to Mobile Devices via Offloading.* Recuperado el 20 de mayo de 2019, de

https://dl.acm.org/citation.cfm?id=3097903

Rodríguez, M. S. (2011). *Spatially structured information in attractor neural networks using metric connectivity.* Recuperado el 18 de mayo de 2019, de https://repositorio.uam.es/xmlui/handle/10486/8356

Sandeep, C. (2009). *Hidden layer optimization of neural network using computational technique.* Recuperado el 18 de abril de 2019, de https://dl.acm.org/citation.cfm?id=1523214

Skapura, D. M. (1996). *Building neural networks.* Recuperado el 15 de abril de 2019, de https://dl.acm.org/citation.cfm?id=217718

Viet Le, H. (2018). *InfiniTouch: Finger-Aware Interaction on.* Recuperado el 20 de mayo de 2019, de http://sven-mayer.com/wp-content/uploads/2018/08/le2018infinitouch.pdf

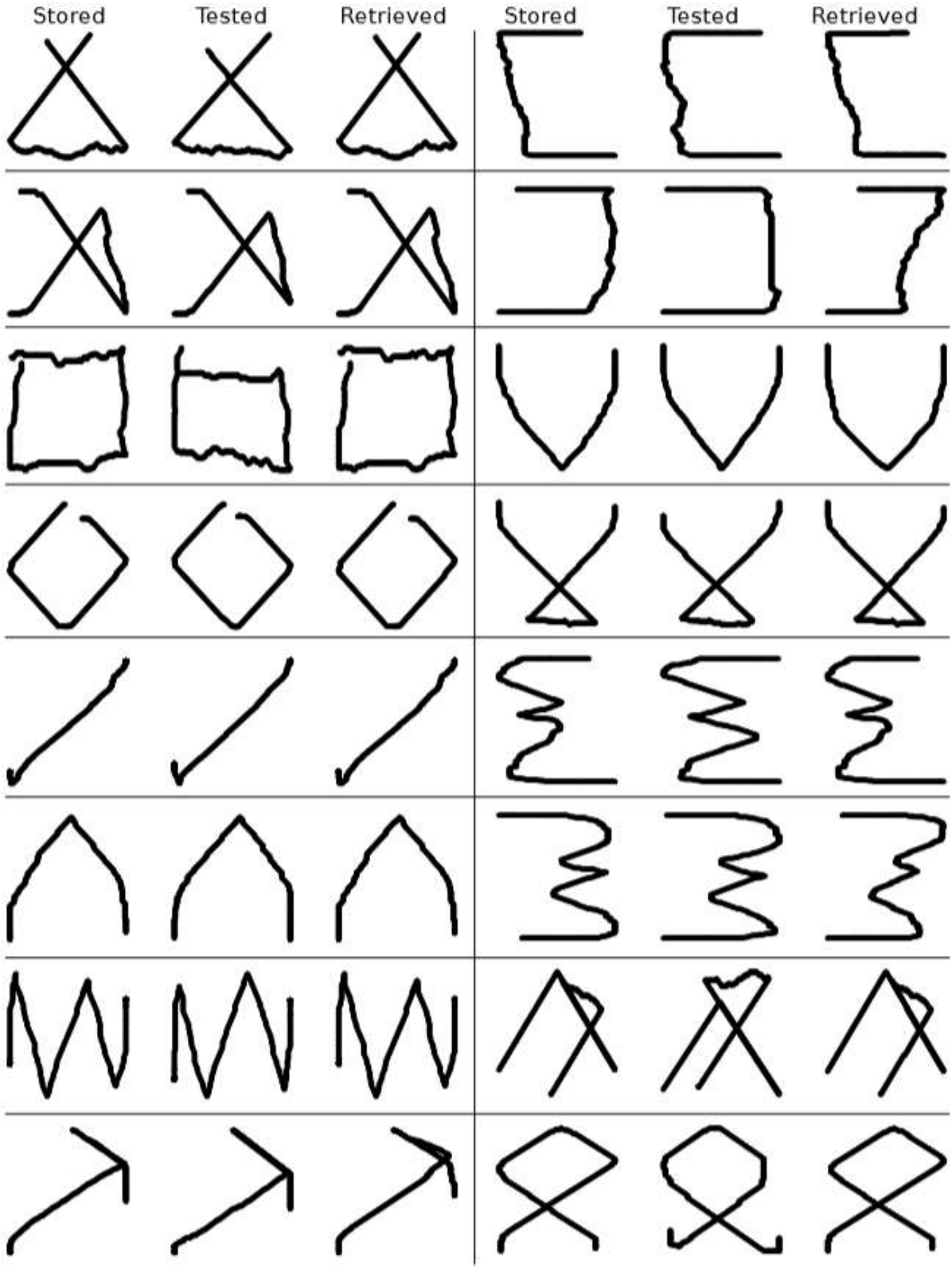
Wobbrock, J. O. (2007). *Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes.* Recuperado el 12 de mayo de 2019, de https://dl.acm.org/citation.cfm?id=1294238

Yanyao, D. (2018). *The Development Overview of Artificial Mind.* Recuperado el 15 de mayo de 2019, de https://dl.acm.org/citation.cfm?id=3241755

You, Y.-S. (2010). *Psychological preference-based optimization framework on the nurse scheduling problem.* Recuperado el 21 de mayo de 2019, de https://www.researchgate.net/publication/220742086_Psychological_preference-based_optimization_framework_on_the_nurse_scheduling_problem

ANEXOS

Anexo 1: Gestos binarizados



Anexo 2: Codificación de gestos

```
import json
import os
from numpy import random as r
import numpy as np
from scipy.signal import resample as rs
import matplotlib.pyplot as plt

Figures =
['triangle','x','rectangle','circle','check','caret','zigzag','arrow','le
ftSBracket','rightSBracket','v','delete','leftCBracket','rightCBracket','
star','pigtail']
Index =
['00','01','02','03','04','05','06','07','08','09','10','11','12','13','1
4','15']

order = np.arange(16)
#np.random.shuffle(order)
#order = np.array([13 , 2 , 9 , 0 ,10 ,15 , 4  ,7, 14 ,12 , 3,  6,  8 ,
5,  1 ,11])
print(order)

InputPath = '/GesturesTestsOut'
OutputPath = '/PatternsOut'
treshold = 0.01
#treshold = 0.0
points = 501
PlotMarkers = ['bo:', 'gx:', 'r^:', 'ch:', 'm>:', 'ys:', 'kp:', 'b*:',
'g<:', 'r+:', 'cv:', 'mD:']

def plotGesture(array):
    X = []
    Y = []
    for e in array:
        X.append(e[0])
        Y.append(e[1])
    plt.plot(X,Y,PlotMarkers[0])
    plt.axis([850, 1300, 600, 0])
    plt.show()

os.chdir(os.getcwd()+InputPath)
```

```
for count,filename in enumerate(order):
    with open(os.listdir()[filename]) as i:
        Rjson = json.load(i)

        name = Rjson[0][0]['Name']

        aux = []

        for p in Rjson[0][0]['Points']:
            aux +=[(p['X'],p['Y'])]
        #plotGesture(aux)

        primerComa = False
        fixedArray = rs(np.array(aux),points)

        prev_X = float(fixedArray[0][0])
        prev_Y = float(fixedArray[0][1])

        os.chdir('..'+OutputPath)

        primerComa = False
        print(name)

        primerComa = False
        if len(str(count)) == 1:
            prefix = '0'
        else:
            prefix = ''
        with open(prefix+str(count)+Index[filename]+'_'+name+'.txt',
"w+") as o:
            for xy in fixedArray[1:]:
                diffX = prev_X - float(xy[0])
                diffY = prev_Y - (-float(xy[1]))

                if(abs(diffX)<=treshold):
                    outX = r.randint(2)
                else:
                    if prev_X <= float(xy[0]):
                        outX = 1
                    else:
                        outX = 0

                if(abs(diffY)<=treshold):
                    outY = r.randint(2)
```

```
else:
                    if prev_Y <= -(float(xy[1])):
                        outY = 0
                    else:
                        outY = 1

                if primerComa:
                    o.write(',')
                o.write(str(outX))
                o.write(',')
                o.write(str(outY))

                prev_Y = -float(xy[1])
                prev_X = float(xy[0])
                primerComa = True

        os.chdir('..'+InputPath)
```

Anexo 3: Red neuronal

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
import time
from PIL import Image
import os
from scipy.signal import resample as rs
NumpyOptions = [-1, 0, 1]
PlotMarkers = ['bo:', 'gx:', 'r^:', 'ch:', 'm>:', 'ys:', 'kp:', 'b*:',
'g<:', 'r+:', 'cv:', 'mD:']

Figures =
['triangle','x','rectangle','circle','check','caret','zigzag','arrow','le
ftSBracket','rightSBracket','v','delete','leftCBracket','rightCBracket','
star','pigtail']
colors = ['black', 'green', 'red', 'blue']
Index =
['00','01','02','03','04','05','06','07','08','09','10','11','12','13','1
4','15']
order = np.array([13 , 2 , 9 , 0 ,10 ,15 , 4  ,7, 14 ,12 , 3,  6,  8 , 5,
1 ,11])
order = np.arange(16)

N_neurons = 20
N_subnetworks = 2
N_neighbors_per_subnet = 10
```

```
N_steps = 60
N_patterns = 4
N_patterns_per_subnet = 2
replaceProbability = 1
oneProbability = 0.5
readPatterns = False
patternsFolderIn = 'PatternsIn'
patternsFolderOut = 'PatternsOut'
overallM = 0.0

initialNetworkState = []
initialNetworkStateToProcess = []
connections = []
weights = []
overlap_Values = []
local_overlap_Values = []
umbral = 0.0
A = 0
a = 0
nomArchivo = 0

Xi_zeroValues = []
Tau_Values = []
sigma_Values = []


def mainFiles():
    global overlap_Values, local_overlap_Values, initialNetworkState,
initialNetworkStateToProcess, overallM
    start_time = time.time()
    os.chdir(os.getcwd()+'/'+patternsFolderIn)
    if(replaceProbability == 1.0):
        initRandomNetwork()
    else:
        initRingNetwork()
    sn = 0
    i = 0
    hasNext = True
    for filename in os.listdir():
        if hasNext:
            initialNetworkState[sn][i] = ReadFile(filename,sn,i)
            i+=1
            if(i>=N_patterns_per_subnet):
                i = 0
                sn += 1
                if(sn==N_subnetworks):
                    hasNext = False
                    break
```

```
calculateConstraints()
    for subn in range(N_subnetworks):
        for patt in range(N_patterns_per_subnet):
            calculateWeigths(subn,patt)
    print("Patterns learnt")
    os.chdir('..')
    os.chdir(os.getcwd()+'/'+patternsFolderOut)
    sn = 0
    i = 0
    hasNext = True
    for filename in os.listdir():
        if hasNext:
            initialNetworkStateToProcess[sn][i] = ReadFile(filename,sn,i)
            i+=1
            if(i>=N_patterns_per_subnet):
                i = 0
                sn += 1
                if(sn==N_subnetworks):
                    hasNext = False
                    break
    for subn in range(N_subnetworks):
        local_overlap_Values = []
        for patt in range(N_patterns_per_subnet):
            processPatronSteps(subn,patt,1)
        overlap_Values.extend(local_overlap_Values)
    overallM = np.mean(np.mean(overlap_Values))
    printM()
    secs_passed = time.time() - start_time
    print("Time taken: %i min %i sec" %
          ((secs_passed//60), (secs_passed % 60)))
    #plotMOverPatterns()
    printCSV()


def mainRandom():
    global overlap_Values, local_overlap_Values, initialNetworkState
    start_time = time.time()
    if(replaceProbability == 1.0):
        initRandomNetwork()
    else:
        initRingNetwork()
    for sn in range(N_subnetworks):
        for i in range(N_patterns_per_subnet):
            initialNetworkState[sn][i]=generatePattern()
    calculateConstraints()
    for sn in range(N_subnetworks):
```

```
for i in range(N_patterns_per_subnet):
            calculateWeigths(sn,i)
    print("Patterns learnt")
    for sn in range(N_subnetworks):
        local_overlap_Values = []
        for i in range(N_patterns_per_subnet):
            processPatronSteps(sn,i)
        overlap_Values.extend(local_overlap_Values)
    print(np.mean(np.mean(overlap_Values)))
    secs_passed = time.time() - start_time
    print("Time taken: %i min %i sec" %
          ((secs_passed//60), (secs_passed % 60)))
    plotMOverPatterns()


def initRandomNetwork():
    global connections, weights
    connections = [[[] for i in range(N_neurons)] for i in
range(N_subnetworks)]
    weights = [[[] for i in range(N_neurons)] for i in
range(N_subnetworks)]
    for sn in range(N_subnetworks):
        sample = np.arange(N_neurons)
        for i in range(0, N_neurons):
            np.random.shuffle(sample)
            while i in sample[:N_neighbors_per_subnet]:
                np.random.shuffle(sample)
            neuronConnection = []
            for j in sample[:N_neighbors_per_subnet]:
                neuronConnection += [j]
            connections[sn][i] = np.array(neuronConnection)
            weights[sn][i] = np.zeros(N_neighbors_per_subnet)
    print("Network initialized")



def initRingNetwork():
    global connections, weights
    connections = [[[] for i in range(N_neurons)] for i in
range(N_subnetworks)]
    weights = [[[] for i in range(N_neurons)] for i in
range(N_subnetworks)]
    for sn in range(N_subnetworks):
        for i in range(N_neurons):
            neuronConnection = []
            listPosibilities = InitializeListPosibilities(i)
            for neig in range(0, N_neighbors_per_subnet):
```

```
prob = int(np.random.choice(NumpyOptions, size=1, replace=True, p=[
                        1-replaceProbability, 0, replaceProbability]))
                if(prob == 0):
                    nearest = NearestNode(i, listPosibilities)
                    neuronConnection.append(nearest)
                    listPosibilities.remove(nearest)
                else:
                    conn = int(np.random.choice(
                        listPosibilities, size=1, replace=True))
                    neuronConnection.append(conn)
                    listPosibilities.remove(conn)
            connections[sn][i] = np.array(neuronConnection)
            weights[sn][i] = np.zeros(N_neighbors_per_subnet)
    print("Network initialized")


def NearestNode(n, listPosibilities):
    nearest = min(range(len(listPosibilities)),
                  key=lambda neurons: abs(listPosibilities[neurons]-n))
    farest = max(range(len(listPosibilities)),
                 key=lambda neurons: abs(listPosibilities[neurons]-n))
    if((abs(listPosibilities[nearest]-n) <= abs(abs(N_neurons -
listPosibilities[farest]) + n) and n < N_neurons/2) or
(abs(listPosibilities[nearest]-n) <= abs(abs(N_neurons +
listPosibilities[farest]) - n) and n > N_neurons/2)):
        return listPosibilities[nearest]
    else:
        return listPosibilities[farest]


def InitializeListPosibilities(n):
    listPosibilities = []
    for a in range(N_neurons):
        if n != a:
            listPosibilities.append(a)
    return listPosibilities


def generatePattern():
    return binom.rvs(n=1, p=oneProbability, size=N_neurons)
    #print("Pattern generated")


def calculateConstraints():
    global umbral, a, A
    a = np.mean(np.mean(initialNetworkState))
    A = a*(1-a)
    umbral = (1-2*a)/(2*np.sqrt(A))
```

```
def calculateWeigths(subnet, i):
    global weights
    alocal = np.mean(initialNetworkState[subnet][i])
    Alocal = a*(1-a)
    #print(alocal, Alocal)
    E = (initialNetworkState[subnet][i]-alocal)/np.sqrt(Alocal)
    for n in range(N_neurons):
        Aux = E[n]*E[connections[subnet][n]].T
        weights[subnet][n] = np.add(weights[subnet][n], Aux)
    #print("Weights updated")


def processPatronSteps(subnet,i,flag=0):
    global local_overlap_Values, Tau_Values, nomArchivo
    nomArchivo += 1
    Xi_zeroValues = (initialNetworkState[subnet][i]-a)/np.sqrt(A)
    m_flag = True
    m = 0
    if(flag==0):
        Tau_Values = np.array(initialNetworkState[subnet][i])
    else:
        Tau_Values = np.array(initialNetworkStateToProcess[subnet][i])
    for it in range(N_steps):
        calculateVariables()
        if(m_flag):
            tau_plusValues = []
            for n in range(N_neurons):
                tau_plusValues +=
[HeavisideFuntion(calculateH(subnet,n))]

            mAux = abs(np.mean(sigma_Values*Xi_zeroValues))

            if(mAux == m):
                m_flag = False

            m = mAux
            Tau_Values = np.array(tau_plusValues)
            if(it == N_steps-1):
                printBinarized(Tau_Values,subnet,i)
                local_overlap_Values.append(m)
                break
        else:
            printBinarized(Tau_Values,subnet,i)
            local_overlap_Values.append(m)
            break
```

```
def calculateVariables():
    global sigma_Values
    q = np.mean(Tau_Values)
    Q = q*(1-q)
    sigma_Values = (Tau_Values-q)/np.sqrt(Q)


def calculateH(subnet,neuronNumber):
    tau_local = Tau_Values[connections[subnet][neuronNumber]]
    qL = np.mean(tau_local)
    QL = qL*(1-qL)
    sigma_Values_Local = (tau_local-qL)/np.sqrt(QL)
    Aux = weights[subnet][neuronNumber]*sigma_Values_Local
    return np.mean(Aux)-umbral

def HeavisideFuntion(h):
    if h >= 0:
        return 1
    else:
        return 0

def plotMOverPatterns():
    simbol = 0
    count = 1
    for i in overlap_Values:
        plt.plot(count, i, PlotMarkers[simbol])
        if (count % (N_patterns_per_subnet)) == 0:
            simbol += 1
        count += 1
    plt.axis(ymin = 0, ymax = 1.1)
    plt.show()

def plotBinarized(array):
    array = (array*2)-1
    count = 0
    X = []
    Y = []
    acummX = 0
    acummY = 0
    while count < N_neurons:
        acummX = array[count]+acummX
        X.append(acummX)
        count += 1
        acummY = array[count]+acummY
        Y.append(acummY)
        count += 1
    plt.plot(X,Y,PlotMarkers[0])
    plt.show()
```

```
def printBinarized(a,sn,i):
    os.chdir('..')
    os.chdir(os.getcwd()+'/Binarized')
    primerComa = False
    if len(str(nomArchivo)) == 1:
        prefix = '0'
    else:
        prefix = ''
    with open(prefix+str(nomArchivo)+'.txt', "w+") as o:
        for it in a:
            if primerComa:
                o.write(',')
            o.write(str(it))
            primerComa = True

def ReadFile(filename,sn,i):
    global initialNetworkState
    with open(filename) as f:
        rawArray = np.fromstring(f.read(), dtype=int, sep=',')
    if(len(rawArray)<N_neurons):
        print("El patron: " + filename + " tiene un tamanio menor al
numero de neuronas")
        exit()
    elif(len(rawArray)==N_neurons):
        return np.array(rawArray)
    else:
        fixedArray = rs(rawArray,N_neurons)
        Pattern = []
        for a in fixedArray:
            if a >= np.mean(fixedArray):
                Pattern += [1]
            else:
                Pattern += [0]
        return Pattern

def DecodeImage(flatArray):
    TwoDArray = np.reshape(flatArray, (32, 32))
    plt.imshow(TwoDArray, cmap='gray')
    plt.show()

def printM():
    for j, overlap in enumerate(overlap_Values):
        print(str(Index[order[j]])+' '+str(Figures[order[j]]) + ': ' +
str(overlap))
```

```
def printCSV():
    os.chdir('..')
    with open('overlaps0.csv','a') as r:
        count = 0
        for sn in range(N_subnetworks):
            for i in range(N_patterns_per_subnet):
                r.write('%i,%i,%s,%f,%s,%s,%f\n' % (count+1,
order[count], Figures[order[count]], overlap_Values[order[count]],
colors[sn], 'net0'+str(sn), overallM))
                count += 1

def setVariables(args):
    global N_neurons, N_subnetworks, N_neighbors_per_subnet, N_steps,
N_patterns, replaceProbability, oneProbability, N_patterns_per_subnet,
initialNetworkState, readPatterns, patternsFolderIn, patternsFolderOut,
initialNetworkStateToProcess
    if(not(len(args) == 7 or len(args) == 8)):
        print(setVariables.__doc__)
        return False
    else:
        N_neurons = int(args[0])
        N_subnetworks = int(args[1])
        N_neighbors_per_subnet = int(int(args[2])/N_subnetworks)
        N_steps = int(args[3])
        if(len(args) == 7):
            N_patterns = int(args[4])
            oneProbability = float(args[6])
            replaceProbability = float(args[5])
            N_patterns_per_subnet = int(N_patterns/N_subnetworks)
        if len(args) == 8:
            patternsFolderIn = str(args[4])
            patternsFolderOut = str(args[5])
            N_patterns_per_subnet = int(int(args[6])/N_subnetworks)
            replaceProbability = float(args[7])
            readPatterns = True
        initialNetworkState = [[[] for i in range(N_patterns_per_subnet)]
for a in range(N_subnetworks)]
        initialNetworkStateToProcess = [[[] for i in
range(N_patterns_per_subnet)] for a in range(N_subnetworks)]
        return True
# Implementation for console usage
if __name__ == "__main__":
    import sys
    if(setVariables(sys.argv[1:])):
        if readPatterns:
            mainFiles()
        else:
            mainRandom()
```

Anexo 4: Script R de gráficos

```
library(ggplot2)
library(latex2exp)

source("https://raw.githubusercontent.com/marsgr6/r-
scripts/master/scripts/multiplot.R")
p <- list()

df <- read.csv("single_online.csv", header = TRUE)

p[[1]] <- ggplot(data=df, aes(x=pattern, y=overlap)) +
  geom_line(linetype = "dashed") +
  geom_point(aes(), size = 4) + theme_bw()
p[[1]] <- p[[1]] + theme(axis.text=element_text(size=20),
                         axis.text.x = element_text(angle = 45, vjust =
1, hjust=1),
         legend.text=element_text(size=20),
         legend.title=element_text(size=20),
axis.title=element_text(size=24,face="bold"))
p[[1]] <- p[[1]] + scale_x_discrete(limits=c(seq(1,16, by=1)))
p[[1]] <- p[[1]] + scale_y_continuous(breaks = c(seq(0, 1, by=0.1)))
p[[1]] <- p[[1]] + labs(x = TeX(sprintf("$\\mu$")), y="m")
p[[1]] <- p[[1]] + coord_cartesian(ylim=c(0,1))

df <- read.csv("single_offline.csv", header = TRUE)

p[[3]] <- ggplot(data=df, aes(x=pattern, y=overlap)) +
  geom_line(linetype = "dashed") +
  geom_line(aes(x=pattern, y=mean_o), size=1, col="blue") +
  geom_point(aes(), size = 4) + theme_bw()
p[[3]] <- p[[3]] + theme(axis.text=element_text(size=20),
                         axis.text.x = element_text(angle = 45, vjust =
1, hjust=1),
         legend.text=element_text(size=20),
         legend.title=element_text(size=20),
axis.title=element_text(size=24,face="bold"))
p[[3]] <- p[[3]] + scale_x_discrete(limits=c(seq(1,16, by=1)))
p[[3]] <- p[[3]] + scale_y_continuous(breaks = c(seq(0, 1, by=0.1)))
p[[3]] <- p[[3]] + labs(x = TeX(sprintf("$\\mu$")), y="m")
p[[3]] <- p[[3]] + coord_cartesian(ylim=c(0,1))

df <- read.csv("overlaps0.csv", header = TRUE)
```

```
p[[2]] <- ggplot(data=df, aes(x=index, y=overlap)) +
  geom_line(linetype = "dashed") +
  geom_line(aes(x=index, y=mean_o), size=1, col="blue") +
  geom_point(aes(colour = Module), size = 4) + theme_bw()
p[[2]] <- p[[2]] + theme(axis.text=element_text(size=20),
                         axis.text.x = element_text(angle = 45, vjust =
1, hjust=1),
         legend.text=element_text(size=20), legend.position=c(.8,.6),
         legend.title=element_text(size=20),
axis.title=element_text(size=24,face="bold"))
p[[2]] <- p[[2]] + scale_x_discrete(limits= as.factor(df$patterns))
p[[2]] <- p[[2]] + scale_y_continuous(breaks = c(seq(0, 1, by=0.1)))
p[[2]] <- p[[2]] + labs(x = TeX(sprintf("$\\mu$")), y="m")

p[[2]] <- p[[2]] + coord_cartesian(ylim=c(0,1))

df <- read.csv("overlaps.csv", header = TRUE)

p[[4]] <- ggplot(data=df, aes(x=index, y=overlap)) +
  geom_line(linetype = "dashed") +
  geom_line(aes(x=index, y=mean_o), size=1, col="blue") +
  geom_point(aes(colour = Module), size = 4) + theme_bw()
p[[4]] <- p[[4]] + theme(axis.text=element_text(size=20),
                         axis.text.x = element_text(angle = 45, vjust =
1, hjust=1),
         legend.text=element_text(size=20), legend.position=c(.8,.6),
         legend.title=element_text(size=20),
axis.title=element_text(size=24,face="bold"))
p[[4]] <- p[[4]] + scale_x_discrete(limits= as.factor(df$patterns))
p[[4]] <- p[[4]] + scale_y_continuous(breaks = c(seq(0, 1, by=0.1)))
p[[4]] <- p[[4]] + labs(x = TeX(sprintf("$\\mu$")), y="m")

p[[4]] <- p[[4]] + coord_cartesian(ylim=c(0,1))

multiplot(plotlist=p, cols=2)
```