Redes de Hopfield

Gonzalo V. Castiglione, Alan E. Karpovsky, Martín Sturla
Estudiantes Instituto Tecnológico de Buenos Aires (ITBA)

12 de Mayo de 2012

Entrega Final - Informe

***Resumen*—El presente informe busca analizar la implementación de una Red de Hopfield con el fin de analizar su funcionalidad y comportamiento como memoria asociativa direccionable por el contenido.**

***Palabras clave*—memoria asociativa direccionable por el contenido, hopfield, imágenes, memorización, ruido, atractores, estados espúreos**

I. Introducción

Se analizó el comportamiento de una Red de hopfield como memoria asociativa direccionable por el contenido.
El trabajo consistió en la implementación una Red de Hopfield en el lenguaje *Java*. Se tomó un conjunto de patrones a memorizar los cuales pueden visualizarse en la sección ***Anexo A: Patrones*** y se realizaron numerosas pruebas con el fin de evaluar cuestiones como ser si los patrones son verdaderos atractores, cuál es la máxima cantidad de patrones que la red puede memorizar, qué sucede si se le da como entrada a la red un patrón con ruido, etc.

Si bien el algoritmo de hopfield es asincrónico, se decidió implementar una variante sincrónica del mismo para comprar los resultados obtenidos con cada uno.

En las siguientes secciones se desrrollará el modelado del problema como así también los resultados obtenidos y las conclusiones que se pueden extraer de ellos.

II. Desarrollo

A. Modelado del problema

Algunas definiciones preliminares:

- Ψ: Conjunto de patrones a memorizar
- N: Longitud de los patrones de entrada (ej. $N = 64$ si cada imagen es de $8x8$ pixels)
- p: Cantidad de patrones distintos que contiene Ψ

Se representó a la Red de Hopfield en *Java* como una clase abstracta debido a la distinción antes nombrada del algoritmo **sincrónico** versus el algoritmo **asincrónico** (nos permite hacer implementaciones concretas que extiendan de ella).
La clase contiene dos variables: un vector y una matriz que están definidos como sigue:

- **float[N][N] weights**: Matriz de pesos W_{ij} correspondientes a los patrones que se desean memorizar.
- **int[N] states;**: Vector con los estados de cada una de las N neuronas.

Nótese que si bien el vector de estados es de tipo entero, los estados definidos para este problema son 1 o -1 dado que las unidades que estamos utilizando son **bipolares**

Por otra parte se decidió representar a los patrones μ como vectores de enteros ***int[] pattern*** los cuales contienen un 1 en la posición i en el caso de que el i-ésimo pixel del patrón sea negro o contienen un -1 en la posición i en el caso de que el i-ésimo pixel del patrón sea blanco.
Todos los patrones de entrada fueron *normalizados*, es decir que las imágenes cuyo mapa de color estaba en escala de grises fueron pasadas a imágenes blanco y negro; asimismo una de las imágenes que no contenía fondo (fondo transparende) fue modificada dejando intacta la imagen pero agregandole un fondo blanco.

B. Inicialización de la red

Se escribió un método *storePatterns* inicializa la matriz de pesos sinápticos de la red neuronal utilizando la regla del producto externo de *Hebb*. Nótese que la matriz de pesos, una vez inicializada, no cambia durante toda la ejecución del algoritmo.
Por cuestiones de eficiencia y dado que la matriz de pesos es simétrica con $W_{ij} = 0$, el algoritmo recorre sólo la diagonal superior de la misma y **espeja** los resultados.

C. Función de activación

La función de activación utizada es la función signo **f(x) = sgn(x)**.

D. Actualización de estados

Como ya se comentó, se implementaron dos variantes para la actualización de estados: el Modelo de Little y el Modelo de Hopfield los cuales se desarrollarán a continuación. En líneas generales ambos buscan iterar hasta la convergencia (el vector de estados de la red no cambia

respecto del paso anterior). Ambas son equivalentes excepto por la distribución del intervalo de actualizaciones. El modelo de Hopfield da una secuencia aleatoria por lo cual hay baja probabilidad de que dos unidades elijan ser actualizadas en el mismo momento.
La primera es útil para simulaciones y la segunda es apropiada para unidades de hardware autónomas.

Es importante destacar que el método de iteración hasta la convergencia puede caer en **ciclos de longitud 2** por lo que es sumamente importante no sólo chequear que el vector de estados no haya cambiado respecto del paso anterior sino también hay que hacer esta verificación respecto de 2 pasos anteriores.

D.1 Sincrónica (Modelo de Little)

En cada paso se actualizan todas las neuronas de forma simultánea de acuerdo a la siguiente regla:

$$S_i(n+1) = sgn\left(\sum_{j=1}^{N} W_{ij} S_j(n)\right) \tag{1}$$

D.2 Asincrónica (Modelo de Hopfield)

En cada paso se actualiza una unidad por ves; la misma es seleccionada de forma completamente aleatoria y se utiliza la ecuación 1 para la actualización.

E. Interfaz gráfica

III. Resultados

A. Verdaderos atractores

B. Estados espúreos

C. Patrones con ruido o incompletos

D. Patrones invertidos

E. Patrones desconocidos

F. Cantidad de patrones almacenables

Como punto sumamente importante se obserbó que la **cantidad máxima de patrones posibles de almacenar** fue de XXXXXXXXXXXXXXXXX. Esto se puede explicar por YYYYYYYY.

IV. Conclusión

Anexo A: Patrones

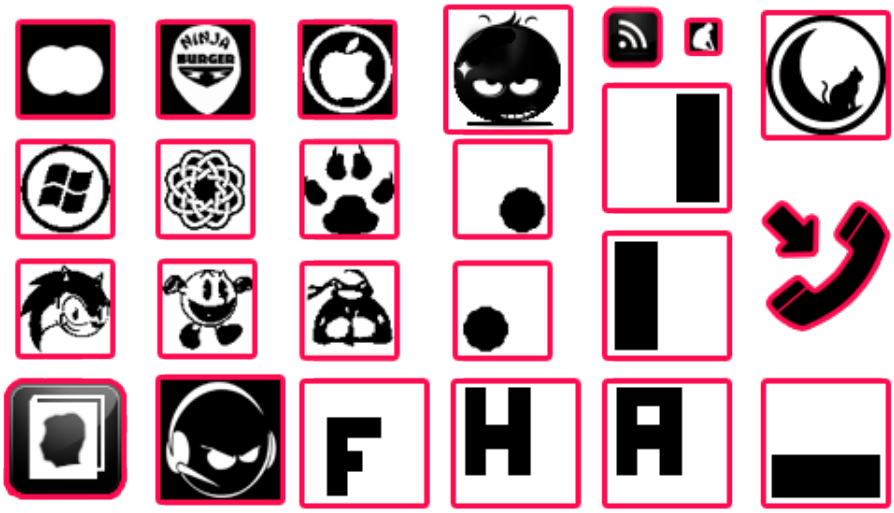

Figura 1: Visualización de los 25 patrones con la que se inicializa a la red