Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral

Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones

Ponzoni Cuadra Nelson Eduardo

Directores: Dr. Rufiner Leonardo
Dr. Martínez César

Proyecto final de carrera
Ingeniería en Informática

Santa Fe, Diciembre 2016

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Agradecimientos

Quiero expresar mis agradecimientos a mis directores de proyecto Leonardo y César por haber confiado en mis habilidades para afrontar este desafío.
A mi madre por su esfuerzo, contención y empuje permanente para alcanzar todas mis metas personales.
A Marilen por incentivarme, darme fuerza y acompañarme en cada momento a lo largo de estos años.
A mis amigos y compañeros por las charlas, consejos y anécdotas compartidas dentro y fuera de los pasillos de la facultad.
Por último pero no menos importante, a mi familia de corazón por brindarme su cariño y apoyo incondicional.
A todos ellos, gracias.

Nelson

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Resumen

En la última década se han realizado grandes avances relacionados al aprendizaje automático, con el surgimiento de nuevos algoritmos y dispositivos de hardware ha sido posible experimentar con nuevas y novedosas tecnologías de la inteligencia artificial.

Las redes profundas siempre han sido difíciles de entrenar, opacadas por los costos en cómputo requerido y por falta de algoritmos adecuados permanecieron ocultas por décadas. Gracias a tecnologías de cálculo paralelo y cómputo masivo, tales como lo son las GP-GPU, este tipo de redes artificiales resurgieron captando la atención de la comunidad científica desempeñando el papel de una poderosa herramienta tanto para tareas de clasificación como también modelos generativos.

En este proyecto se implementó una biblioteca que tiene como objetivo crear, entrenar y ajustar redes de creencia profundas, aprovechando el poder brindado por las GP-GPUs para reducir los tiempos y costos de cómputo necesarios. En el proceso de la misma se cubrieron los pasos para el diseño, implementación y despliegue de dicha biblioteca.

Por otro lado, se realizaron experimentos con el software generado en la base de datos estándar para el Aprendizaje Automático, reconocimiento de dígitos manuscritos MNIST y en un caso de aplicación real en la tarea de reconocimiento de emociones con la base de datos emoción RML. Los experimentos en GP-GPU demostraron un incremento de velocidad en las DBNs hasta 18 veces que pruebas de solo CPU.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Índice general

Lista de figuras

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Lista de tablas

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Lista de acrónimos

Símbolos Matemáticos

$\mathbf{1}$	Función indicatriz o característica de un conjunto.
$\mathbb{E}$	Valor esperado.
$\mathcal{L}$	Función de verosimilitud.
$\boldsymbol{\theta}$	Parámetros del modelo de red.

Otros Símbolos

$\langle \cdot \rangle$	Valor esperado.

Acrónimos / Abreviaciones

BLAS	Basic Linear Algebra Subprograms.
CD	Contrastive Divergence.
CD	Contrastive Divergence.
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture.
DBN	Deep Belief Networks.
GP-GPU	General-Purpose Computing on Graphics Processing Units.
MCMC	Markov Chain Monte Carlo.
MFCC	Coeficientes Cepstrales en Escala de Mel.
MLS	Media del Espectro Logarítmico.
MRF	Markov Random Fields.
PCD	Persistent Contrastive Divergence.
RBM	Restricted Boltzmann Machines.
RNA	Red Neuronal Artificial.

SFU Special Function Unit.

SIMT Single Instruction Multiple Thread.

SM Streaming Multiprocessors.

SP Scalar Processors.

UAL Unidad Aritmético Lógica.

UML Unified Modeling Language.

Capítulo 1

Introducción

En la última década el progreso hecho por el aprendizaje maquinal ha sido asombroso, permitiendo el crecimiento de la popularidad del mismo tanto en el campo de la investigación como así también en la utilización de aplicaciones comerciales. Innumerables trabajos han sido desarrollados en una gran diversidad de temáticas, por ejemplo el reconocimiento y descripción de contenido en una imagen, relacionado al campo de la visión computacional y procesamiento del lenguaje natural; sistemas de recomendación multimedia asociado al entretenimiento; administración autónoma de las relaciones entre empresa-cliente relacionado al marketing y muchos otros [51, 49, 13, 47]

Las reglas estáticas y el comportamiento rígido de los algoritmos por computadora han sido reemplazados por otros con capacidades de adaptación y generalización, dando origen a las llamadas redes neuronales artificiales (RNA). Las mismas constituyen modelos inspirados en el comportamiento biológico de las neuronas y del cerebro humano. Uno de los objetivos de las redes artificiales es construir sistemas que sean capaces de aprender, generalizar y resolver problemas a los que hoy en día no pueden dar solución los algoritmos convencionales. Las RNA presentan un amplio campo de aplicación, como ser en la economía, medicina, ingeniería, estadística, entre otras. Frecuentemente utilizadas en tareas de clasificación y regresión de grandes volúmenes de datos, las aplicaciones son muy variadas, donde el campo de aplicación abarca un amplio espectro que va desde la predicción del mercado bursátil a la detección de rostros, optimización de control de vuelo en aeronaves, entre muchos otros [50, 55, 40].

A través del tiempo, las exigencias fueron aumentando sobre las RNAs, así surgieron nuevos modelos, en especial el que se conoce como *redes profundas*. Este tipo de modelos cuentan con varias capas, las cuales brindan la capacidad de extraer de manera jerárquica características abstractas, nivel por nivel. Estas redes fueron concebidas como objeto de investigación hace más de treinta años atrás, eran simples redes (perceptrón multicapa) más extensas. Los métodos de entrenamiento poseían serias limitaciones cuando eran utilizadas en arquitecturas de algunas pocas capas de profundidad o bien no podían hacer frente a la necesidad de gran cantidad de datos etiquetados [3]. Se puede nombrar algunos de los problemas asociados a lo anterior, como ser el desvanecimiento de los gradientes a través del algoritmo de "propagación hacia atrás", inicialización aleatoria de los pesos sinápticos, escasez de datos clasificados, complejidad de cómputo debido a las operaciones involucradas para su entrenamiento entre otros [3, 20].

Por otro lado, aun con la implementación de algoritmos más eficientes en cuestiones del costo computacional, las redes de creencia profunda poseen una desventaja en contraposición a las redes

simples, el tiempo utilizado para su entrenamiento es extenso, requiriendo horas, días o hasta semanas para el proceso de aprendizaje con una sola configuración [22]. Esta característica es notoria a medida de que el volumen de datos procesados escalan en grandes proporciones, tal como sucede en los casos reales, lo que en la práctica resulta ser poco conveniente [12, 41]. Desde que el científico e investigador Geoffrey Hinton introdujo nuevas técnicas de entrenamiento para las redes profundas en el 2006 [22], las redes de creencia profunda captaron el interés de la comunidad científica una vez mas con notorios resultados en tareas relacionadas a la visión o al habla [16], quebrando récords en benchmarks reconocidos [27].

Con el afán de utilizar este tipo de redes en la práctica, es deseable que el tiempo requerido por el proceso de entrenamiento sea el menor posible, es por ello que se recurre a la ejecución de múltiples tareas de forma paralela en hardware con ayuda de las Unidades Gráficas de Procesamiento para Propósito General (GP-GPU). Estas pueden realizar cómputos de forma paralela y a menor costo que las implementaciones hechas puramente en CPU o distribuidas en un cluster [5].

Durante la última década se enfatizó en la investigación relacionada al campo de la Interacción Humano-Computadora. En este sentido, se han intentado aplicar modelos de comunicación con bases en las Ciencias Sociales como la Ciencia de la Comunicación, las Neurociencias, la Psicología, etc. [7]. La comunicación entre personas se da generalmente de manera simultánea a través del contenido del discurso, la prosodia y del lenguaje corporal. Cada uno de estos elementos de comunicación están vinculados en mayor parte a la carga emocional que transmite el mensaje [39, 38]. De este modo, para poder desarrollar sistemas inteligentes que interactúen más naturalmente con los seres humanos, se deberá implementar el reconocimiento de emociones humanas a partir de las expresiones faciales y señales de la voz.

Realizar tareas de clasificación de emociones humanas a través de una secuencia de imágenes o una grabación de sonido no es una tarea sencilla o factible en la actualidad con procesos simples [1]. Con el fin de superar los problemas actuales de los algoritmos científicos para la detección de emociones humanas, se deberá centrarse en la extracción de características, en lugar del proceso de clasificación [36]. Esta tarea se logra gracias a la funcionalidad que las redes de creencia profunda poseen para extraer características de los datos, estas pueden ser aplicadas a procesos relacionados con emociones humanas [45].

Es por ello, resulta de interés generar un sistema que implemente las redes de creencia profunda, que brinde las funcionalidades necesarias para la libre ejecución en GP-GPU y relacionarlo a un caso practico como al reconocimiento de emociones humanas.

1.1. Justificación

Con la funcionalidad que las redes de creencia profunda aportan para la extracción de características sobre los datos de forma jerárquica, resulta de interés aplicarlas a casos prácticos complejos. Producir un software capaz de entrenar dichas redes, que reduzca los tiempos de ejecución y menor costo es de gran utilidad, objetivo que se logra con ayuda de la GP-GPU.

Actualmente, existen diversas librerías que implementan algoritmos para redes profundas, algunas de ellas son pyLearn2, Caffe, Torch, entre otras [32, 48, 6, 30]. Aún si bien son de libre utilización, cualquier modificación ya sea en carácter de mejora, expansión de características o bien corrección de errores, queda en completa responsabilidad por parte del usuario y es este último el que debe poseer

conocimientos exclusivos del lenguaje de programación en que las librerías fueron desarrolladas, de conocer el diseño y la estructura interna de las mismas.

A la fecha las librerías mencionadas no cubren con las características básicas de redes de creencia profunda, poseen implementaciones específicas desarrolladas para algún trabajo de investigación por parte de su desarrollador y no de forma general, lo cual quita la posibilidad de aplicarlas a problemas de otra índole ajena al que fueron concebidas. Es por todo ello que se desea implementar los algoritmos de entrenamiento y prueba de las redes de creencia profunda, y así aplicarlos al reconocimiento de emociones humanas. Todo en un solo paquete de software, una librería con las funcionalidades básicas de entrenamiento de las redes como así también tareas de preprocesamiento de los datos, utilizando la GP-GPU.

El sistema desarrollado será de gran utilidad para la comunidad académica de la Facultad de Ingeniería y Ciencias Hídricas, UNL, investigadores y expertos en el área de la inteligencia computacional podrán realizar sus trabajos valiéndose de esta herramienta para aumentar la productividad, reduciendo los tiempos necesarios y descubrir nuevos horizontes del conocimiento.

1.2. Objetivos

Generales

- Desarrollar una biblioteca para la creación, entrenamiento y prueba Redes de Creencia Profunda y Máquinas de Boltzmann Restringidas que utilice la GP-GPU.

Específicos

- Relevar características de las librerías pyCUDA, Theano, PyLearn2 y Torch.
- Implementar algoritmos para el entrenamiento y prueba de Máquinas de Boltzmann Restringidas y Redes de Creencia Profunda.
- Implementar en GP-GPU los algoritmos para el entrenamiento de RBM y DBN.
- Evaluar el software aplicado al reconocimiento de emociones.
- Analizar el sistema con respecto a otras arquitecturas de redes tradicionales.
- Realizar el informe final y documentación del sistema.

1.3. Alcances

En este proyecto se desarrollará un conjunto de algoritmos valiéndose de técnicas de aprendizaje profundo utilizando el cómputo paralelo de las unidades gráficas de propósito general. Los principales usuarios al cual están dirigidos pertenecen a la comunidad académica de la Universidad Nacional del Litoral.

Se requiere que el software final implemente un algoritmo de entrenamiento para las DBNs robusto y veloz a comparación de implementaciones estándar iterativas. Para tal fin, se debe utilizar hardware especifico de cálculo masivo denominado GP-GPU y omitirse el desarrollo de la interfaz gráfica para el usuario.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Capítulo 2

Fundamentos Teóricos

2.1. Introducción

Las redes artificiales han prosperado desde que Geoffrey Hinton mostró a la comunidad científica a través de su trabajo de investigación que las Máquinas de Boltzmann Restringidas pueden ser apiladas entre si para conformar una red de creencia profunda, ambos modelos generativos de naturaleza probabilística que además son utilizados tanto en tareas de clasificación como regresión [22, 21, 18].

El objetivo de este capitulo es proveer los conceptos básicos sobre estos modelos, las técnicas y algoritmos utilizados para constituir el sistema de aprendizaje profundo, *Redes de Creencia Profunda*. Es común referirse a las mismas por sus siglas tomadas del ingles DBN (*Deep Belief Networks*). De igual manera, con el objetivo de unificar definiciones, utilizaremos el acrónimo RBM para referirnos a las Máquinas de Boltzmann Restringidas (*Restricted Boltzmann Machines*).

Como se verá a continuación, la configuración de una DBN se compone por bloques de RBMs, las mismas provienen de un modelo más general, los energéticos.

De este modo se introducirá la matemática básica, maquinaria fundamental para la compresión de los conceptos básicos sobre dichos modelos, también se incluirá el desarrollo del algoritmo de entrenamiento para las redes en cuestión, Divergencia Contrastiva (CD).

2.2. Modelos Energéticos

Los modelos energéticos asocian una función escalar de energía a cada configuración de las variables que los componen. El aprendizaje se corresponde a la modificación del valor de dicha función, la cual posee una forma con propiedades deseadas [3, 9]. Por ejemplo, es requerido o deseable que las configuraciones *óptimas* del modelo posean un nivel bajo de energía mientras que las otras configuraciones un nivel superior. Los modelos energéticos probabilísticos definen una distribución de probabilidad a través de la función de energía, como la siguiente:

$$p(\mathbf{x}) = \frac{\mathrm{e}^{-\text{Energía}(\mathbf{x})}}{\mathcal{Z}} \tag{2.1}$$

donde $\mathbf{x}$ es la variable aleatoria de entrada. Es común referirse a la energía del modelo como una función que opera en el dominio logarítmico. La misma se puede generalizar como todas aquellas funciones de la familia exponencial.

El factor de normalización $\mathcal{Z}$ es llamado *función de partición* en analogía a los sistemas físicos,

$$\mathcal{Z} = \sum_{\mathbf{x}} \mathrm{e}^{-\mathrm{Energía}(\mathbf{x})} \tag{2.2}$$

donde la sumatoria toma todos los valores del espacio de entrada discreto, o bien, una integral apropiada para el caso continuo. De hecho, algunos modelos energéticos definen $\mathcal{Z}$ donde la sumatoria/integral no existe [9].

Introduciendo variables latentes

En muchos casos de interés $\mathbf{x}$ contiene una gran cantidad de variables x_i, y no es posible observar dichas variables de forma simultanea o bien se desea introducir algunas variables no observadas en orden de incrementar el poder expresivo del modelo [14]. De este modo al conjunto de variables $\mathbf{x}$ lo separamos en dos subconjuntos, considerando a las variables observadas como $\mathbf{v}$ y a las latentes o ocultas $\mathbf{h}$ se puede definir el modelo de la siguiente manera,

$$p(\mathbf{v}, \mathbf{h}) = \frac{\mathrm{e}^{-\mathrm{Energía}(\mathbf{v},\mathbf{h})}}{\mathcal{Z}} \tag{2.3}$$

donde

$$\mathcal{Z} = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \mathrm{e}^{-\mathrm{Energía}(\mathbf{v},\mathbf{h})} \tag{2.4}$$

Como solo $\mathbf{v}$ es observada, la distribución marginal de $\mathbf{v}$ está dada por,

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \mathrm{e}^{-\mathrm{Energía}(\mathbf{v},\mathbf{h})} \tag{2.5}$$

En estos casos, es conveniente mapear la formulación anterior a una que cuente con la forma similar a la ecuación 2.1, se introduce inspirado de la física, la definición de energía libre

$$p(\mathbf{v}) = \frac{\mathrm{e}^{-\mathrm{EnergíaLibre}(\mathbf{v})}}{\mathcal{Z}} \tag{2.6}$$

con $\mathcal{Z} = \sum_{\mathbf{v}} \mathrm{e}^{-\mathrm{EnergíaLibre}(\mathbf{v})}$, se puede definir como:

$$\mathrm{EnegíaLibre}(\mathbf{v}) \quad = \quad \mathcal{F}(\mathbf{v}) \quad = \quad -\log \sum_{\mathbf{h}} \mathrm{e}^{-\mathrm{Energía}(\mathbf{v},\mathbf{h})} \tag{2.7}$$

por lo que la energía libre no es más que la marginalización de las energías en el dominio logarítmico.

El gradiente de la función de verosimilitud[1] logarítmica (útil en el entrenamiento de la red por medio del *gradiente ascendiente estocástico* como se verá más adelante) posee una forma interesante. A partir de la ecuación 2.5 y con parámetros $\boldsymbol{\theta}$, la función de verosimilitud logarítmica para un solo punto, ejemplo del espacio de entrada $\mathbf{v}$ es:

[1] Función de verosimilitud, es una función de los parámetros de un modelo estadístico que permite realizar inferencias acerca de su valor a partir de un conjunto de observaciones. En la literatura puede encontrarse generalmente con el símbolo $\mathcal{L}$.

$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v}) &= \log p(\mathbf{v}|\boldsymbol{\theta}) \\ &= \log \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} \\ &= \log \sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} - \log \mathcal{Z} \\ &= \log \sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} - \log \sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}\end{aligned} \tag{2.8}$$

El gradiente con respecto a los parámetros de la función anterior tiene la forma de,

$$\begin{aligned}\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}} \left(\log \sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} \right) - \frac{\partial}{\partial \boldsymbol{\theta}} \left(\log \sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} \right) \\ &= -\frac{1}{\sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} \frac{\partial \text{Energía}(\mathbf{v},\mathbf{h})}{\partial \theta} \\ &\quad + \frac{1}{\sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}} \sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})} \frac{\partial \text{Energía}(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \\ &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial \text{Energía}(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial \text{Energía}(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}\end{aligned} \tag{2.9}$$

en el último paso se utilizo la probabilidad condicional, gracias al teorema de Bayes, escrita de la siguiente forma:

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{h},\mathbf{v})}{p(\mathbf{v})} = \frac{\frac{1}{\mathcal{Z}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}}{\frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}} = \frac{\mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{h}} \mathrm{e}^{-\text{Energía}(\mathbf{v},\mathbf{h})}}$$

Notar que la ecuación 2.9 es la suma entre dos esperanzas[2],

$$\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v})}{\partial \boldsymbol{\theta}} = -\mathbb{E}_{p(\mathbf{h}|\mathbf{v})} \left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right] + \mathbb{E}_{p(\mathbf{h},\mathbf{v})} \left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right] \tag{2.10}$$

Hallar el valor del primer termino, basta con calcularlo analíticamente en un solo paso, lo cual es factible. Por otro lado, conocer el valor del segundo termino, la probabilidad conjunta sobre el modelo P es muy costoso de calcular para el sistema computacional, debido a que es necesario visitar todos los valores posibles de las variables, esta situación no es viable en casos prácticos.

Con ayuda de las técnicas conocidas como *Monte Carlo vía Cadenas de Markov* se puede obtener una aproximación del segundo termino para gradiente estocástico de la función de verosimilitud logarítmica [9].

2.3. Cadenas de Markov

En la teoría de la probabilidad se conoce como cadena de Markov a un proceso estocástico discreto que cumple con la propiedad de Markov, es decir, si se conoce la historia del sistema hasta su instante actual, su estado presente resume toda la información relevante para describir en probabilidad su

[2] Aquí $\mathbb{E}$ representa la esperanza matemática o valor esperado.

estado futuro. En este modelo la probabilidad de que ocurra un evento depende *solamente* del evento inmediatamente anterior. Esta característica de falta de memoria recibe el nombre de propiedad de Markov.

Una cadena de Márkov es una secuencia $X_1, X_2, X_3, \ldots, X_n$ de variables aleatorias. El dominio de estas variables es llamado espacio estado; el valor de X_n es el estado del proceso en el tiempo n. Si la distribución de probabilidad condicional de X_{n+1} en estados pasados es una función de X_n por sí sola, entonces:

$$p(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, \cdots, X_2 = x_2, X_1 = x_1) = p(X_{n+1} = x_{n+1} | X_n = x_n)$$

donde X_n es un estado al momento n. Es claro que la probabilidad de transición de un estado a otro esta dada unicamente por el estado actual n y el anterior $n-1$.

Las cadenas de Markov juegan un papel importante en el entrenamiento de las RBMs, debido a que proveen del soporte para el muestreo de distribuciones de probabilidad complejas.

Las técnicas de *Monte Carlo vía Cadenas de Markov* (MCMC) simulan una cadena de Markov cuyos estados siguen una probabilidad dada en un estado de espacios de grandes dimensiones. Esencialmente generan muestras de una probabilidad que se desconoce. El método de muestreo que se analizará es el asociado a una distribución de Gibbs, para los campos aleatorios de Markov (MRF).

Muestreo de Gibbs

El muestreo de Gibbs pertenece al grupo de algoritmos de Metropolis-Hastings. Este es un algoritmo MCMC para obtener una secuencia de observaciones las cuales son aproximaciones a una distribución de probabilidad especifica, útil cuando el muestreo directo es difícil de realizar. Esta secuencia puede ser utilizada para aproximar la distribución conjunta, aproximar la distribución marginal de una de las variables o un subconjunto de ellas. Típicamente algunas de las variables involucradas se corresponden a observaciones cuyo valores son conocidos y, además no hay necesidad de muestrear.

La idea básica del algoritmo se centra en la construcción de una cadena de Markov y luego la actualización de cada variable con probabilidades a posteriori basándose en la distribución condicional dada por el estado de las otras variables. El muestreo de Gibbs puede ser utilizado para producir muestras (aproximadas) de una distribución de Gibbs de un MRF [14].

Considerando un MRF, $\mathbf{X} = (X_1, \ldots, X_N)$ y un grafo no dirigido $G = (V, E)$ donde $V = \{1, \ldots, N\}$. La variable aleatoria $X_i,\ i \in V$ toma valores de un conjunto finito Λ y $\pi(\mathbf{x}) = \frac{1}{\mathcal{Z}} \mathrm{e}^{-\varepsilon(\mathbf{x})}$ es la distribución de probabilidad conjunta de $\mathbf{X}$. Además, si se asume que el MRF cambia su estado a través del tiempo, podemos considerar $X = \{\mathbf{X}^{(k)} | k \in \mathbb{N}_0\}$ como una cadena de Markov que toma los valores de $\Omega = \Lambda^N$. Luego $\mathbf{X}^{(k)} = (X_1^{(k)}, \ldots, X_N^{(k)})$ describe el estado del MRF al tiempo $k \geq 0$. Entre dos sucesivos puntos en el tiempo, el nuevo estado de la cadena es producido por el siguiente procedimiento. Primero, una variable $X_i,\ i \in V$ es elegida de forma aleatoria con una probabilidad $q(i)$ dada por una distribución de probabilidad estrictamente positiva de q sobre V. Luego, el nuevo estado de la variable x_i es muestreado basado en la distribución de probabilidad condicional dada por el estado $(x_v)_{v \in V \setminus i}$ dada todas las otras variables $(X_v)_{v \in V \setminus i}$. Entonces tenemos, $\pi(\mathbf{x}_i | (x_v)_{v \in V \setminus i}) = \pi(\mathbf{x}_i | (x_\omega)_{\omega \in \mathcal{N}_i}$ debido a la propiedades de los MRF [9, 14]. La probabilidad de transición p_{xy} para dos estados $\mathbf{x}, \mathbf{y}$ de un MRF con $\mathbf{x} \neq \mathbf{y}$ es,

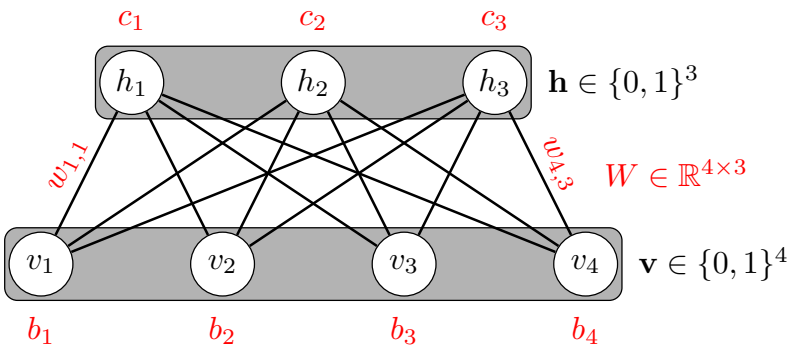


Fig. 2.1 Ejemplo de una Máquina de Boltzmann Restringida binaria. Con unidades visibles $v_i, i = 1,2,3,4$ y unidades ocultas $h_j, j = 1,2,3$. Cada una con sus respectivos bias, $\mathbf{b}$ y $\mathbf{c}$. La matriz de pesos $W_{4,3}$ conecta cada unidad visible con las unidades ocultas. Notar que $W_{4,3} = W_{3,4}$

$$p_{\mathbf{xy}} = \begin{cases} q(i)\pi(y_i|(x_v)_{v\in V\setminus i}), & \text{si } \exists i \in V \text{ entonces } \forall v \in V \text{ con } \ v \neq i : x_v = y_v \\ 0, & \text{cualquier otro caso.} \end{cases}$$

Y la probabilidad, de que el estado $\mathbf{x}$ del MRF permanezca igual es, $p_{\mathbf{xx}} = \sum_{i\in V} q(i)\pi(x_i|(x_v)_{v\in V\setminus i})$
En la sección 2.4.3 se desarrollará el algoritmo especifico.

2.4. Máquinas de Boltzmann Restringidas

Al inicio del capitulo se mencionaron las RBMs como un elemento clave para el proceso de aprendizaje de las DBNs, en este capitulo nos enfocaremos en como entrenar éstos modelos.

Una *máquina de Boltzmann* pertenece a la categoría de los modelos generativos, es una red artificial recurrente de conexiones simétricas, cuyas unidades (neuronas) son activadas de forma estocástica, lo que permite aprender regularidades complejas presentes en los datos de entrenamiento [14]. Al agregar ciertas restricciones a las conexiones, se obtiene una versión restringida de la red, llamada máquina restringida de boltzmann (RBM), que consiste en una capa de unidades ocultas $\mathbf{h}$ y otra de unidades visibles $\mathbf{v}$. Su estructura es tal, que las conexiones entre ambas capas $\mathbf{W}$ son simétricas y no se permiten conexiones entre unidades del tipo visible-visible u oculta-oculta. De ésta manera se forma un grafo bipartito no dirigido como en la figura 2.1, donde $\mathbf{b}$ y $\mathbf{c}$ son vectores de biases para las unidades visibles y ocultas respectivamente y $\mathbf{W}$ es la matriz de pesos (sinápticos) entre ellas.

Mas formalmente, una RBM consiste en n unidades visibles $\mathbf{V} = (V_1, \ldots, V_n)$ que representan los datos observados, además se tiene m unidades ocultas o latentes $\mathbf{H} = (H_1, \ldots, H_m)$ que capturan las dependencias entre las variables observadas. Existen varios tipos de RBMs según el dominio soporte de las unidades que la componen. Se realizará la mayor parte del análisis para las RBM Binarias, aunque este puede extenderse a otros tipos de RBM (ver la sección 2.4.6).

2.4.1. RBM Binaria: unidades visibles-ocultas binarias

Este es el tipo de RBM más común y debido a su naturaleza es el más simple. En él, las variables aleatorias $(\mathbf{V}, \mathbf{H})$ toman valores $(\mathbf{v}, \mathbf{h}) \in \{0,1\}^{n+m}$ y la distribución de probabilidad conjunta sobre el modelo está dada por la distribución de Gibbs $p(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}} \mathrm{e}^{-\mathrm{E}(\mathbf{v},\mathbf{h})}$ [3]con función de energía,

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{n} b_i v_i - \sum_{j=1}^{m} c_j h_j - \sum_{i=1}^{n}\sum_{j=1}^{m} h_j v_i w_{ij} \tag{2.11}$$

o en notación vectorial,

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} - \mathbf{v}^\top W \mathbf{h}$$

para todo $i \in \{1, \ldots, n\}$ y $j \in \{1, \ldots, m\}$, w_{ij} es una matriz de valores reales, que asocia a las unidades V_i y H_j. Los otros términos involucrados, b_i y c_j son los *bias*, éstos toman valores reales asociados a las unidades observadas y ocultas respectivamente.

La energía libre para una RBM binaria, según la ecuación 2.7 tiene la forma,

$$\mathcal{F}(\mathbf{x}) = -\mathbf{b}'\mathbf{x} - \sum_{j} \log \sum_{h_j} \mathrm{e}^{\mathrm{h_j(c_j + W_j x)}} \tag{2.12}$$

La distribución condicional $p(\mathbf{h}|\mathbf{v})$ y $p(\mathbf{v}|\mathbf{h})$ se factorizan de forma muy simple, resultando en expresiones compactas [18, 14, 3]. Por ejemplo, para $p(\mathbf{h}|\mathbf{v})$ se tiene,

$$\begin{aligned} p(\mathbf{h}|\mathbf{v}) &= \frac{\exp(\mathbf{b}'\mathbf{v} + \mathbf{c}'\mathbf{h} + \mathbf{v}\mathbf{W}\mathbf{h}')}{\sum_{\tilde{\mathbf{h}}} \exp(\mathbf{b}'\mathbf{v} + \mathbf{c}'\tilde{\mathbf{h}} + \mathbf{v}\mathbf{W}\tilde{\mathbf{h}}')} \\ &= \frac{\prod_j \exp(c_j h_j + h_j W_j v)}{\prod_j \sum_{\tilde{h}} \exp(c_j \tilde{h}_j + \tilde{h}_j W_j v)} \\ &= \prod_j \frac{\exp(h_j(c_j W_j v))}{\sum_{\tilde{h}} \exp(\tilde{h}_j(c_j + W_j v))} \\ &= \prod_j p(h_j|\mathbf{v}) \end{aligned} \tag{2.13}$$

por lo tanto,

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{m} p(h_j|\mathbf{v}) \quad \text{y} \quad p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{n} p(v_i|\mathbf{h}) \tag{2.14}$$

Por otro lado, es necesario determinar como se activan las unidades estocásticas. Considerando la ecuación 2.11 para una RBM con n unidades visibles y m unidades ocultas, tomando una unidad visible v_l la ecuación de energía se puede rescribir como,

[3]El símbolo E representa la Energía, se utilizará esta simplificación para facilitar la lectura.

$$
\begin{aligned}
E(\mathbf{v},\mathbf{h}) &= -\sum_{i=1}^{n} a_i v_i - \sum_{j=1}^{m} b_j h_j - \sum_{i=1}^{n}\sum_{j=1}^{m} h_j v_i w_{ij} \\
&= -a_l v_l - \sum_{j} w_{lj} v_l h_j - \sum_{i\neq l} a_i v_i - \sum_{i} b_i h_i - \sum_{i\neq l}\sum_{j} w_{ij} v_i h_j \\
&= -v_l \underbrace{\left(a_l + \sum_{j} w_l j h_j \right)}_{\beta_l} \underbrace{- \sum_{i\neq l} a_i v_i - \sum_{i} b_i h_i - \sum_{i\neq l}\sum_{j} w_{ij} v_i h_j}_{\alpha_l} \\
&= -v_l \beta_l + \alpha_l
\end{aligned}
\tag{2.15}
$$

denotando al conjunto de las unidades visibles $\mathbf{v}$ sin la unidad v_l como $\mathbf{v_{-l}}$ se tiene,

$$
\begin{aligned}
p(v_l = 1|\mathbf{h}) = p(v_l = 1|\mathbf{v_{-l}}, \mathbf{h}) &= \frac{p(v_l = 1, \mathbf{v_{-l}}, h)}{p(\mathbf{v_{-l}}, \mathbf{h})} \\
&= \frac{p(v_l = 1, \mathbf{v_{-l}}, h)}{p(v_l = 0, \mathbf{v_{-l}}, \mathbf{h}) + p(v_l = 1, \mathbf{v_{-l}}, \mathbf{h})} \\
&= \frac{\mathrm{e}^{-\mathrm{E}(\mathrm{v_l}=1, \mathbf{v_l}, \mathbf{h})}}{\mathrm{e}^{-\mathrm{E}(\mathrm{v_l}=0, \mathbf{v_l}, \mathbf{h})} + \mathrm{e}^{-\mathrm{E}(\mathrm{v_l}=1, \mathbf{v_l}, \mathbf{h})}} \\
&= \frac{\mathrm{e}^{-(-1\beta_l + \alpha_l)}}{\mathrm{e}^{-(-0\beta_l + \alpha_l)} + \mathrm{e}^{-(-1\beta_l + \alpha_l)}} \\
&= \frac{\mathrm{e}^{-\alpha_l}\mathrm{e}^{1\beta_l}}{\mathrm{e}^{-\alpha_l}\mathrm{e}^{0\beta_l} + \mathrm{e}^{-\alpha_l}\mathrm{e}^{1\beta_l}} \\
&= \frac{\mathrm{e}^{\beta_l}}{1 + \mathrm{e}^{\beta_l}} \\
&= \frac{1}{1 + \mathrm{e}^{-\beta_l}} \\
&= \sigma(\mathbf{v_l})
\end{aligned}
$$

donde σ es la función sigmoidea, $\sigma(\mathbf{x}) = 1/(1 + \mathrm{e}^{-\mathbf{x}})$.

Teniendo en cuenta el resultado anterior y la ecuación 2.13 podemos generalizar, la función de activación para cualquier unidad (probabilidad de que se dispare o encienda) se corresponde a la función sigmoidea,

$$
p(H_j = 1|\mathbf{v}) = \sigma\left(\sum_{i=1}^{n} w_{ij} v_i + c_j \right) \tag{2.16}
$$

$$
p(V_i = 1|\mathbf{h}) = \sigma\left(\sum_{j=1}^{m} w_{ij} h_j + b_i \right) \tag{2.17}
$$

Lo siguiente es observar la forma de la distribución de probabilidad sobre $\mathbf{V}$, la distribución marginal de las variables visibles,

$$
\begin{aligned}
p(\mathbf{v}) &= \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) \\
&= \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \mathrm{e}^{-\mathrm{E}(\mathbf{v},\mathbf{h})} \\
&= \frac{1}{\mathcal{Z}} \sum_{h_1} \sum_{h_2} \cdots \sum_{h_m} \mathrm{e}^{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{b_i v_i}} \prod_{j=1}^{m} \mathrm{e}^{\mathrm{h_j}\left(\mathrm{c_j}+\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_{ij} v_i}\right)} \\
&= \frac{1}{\mathcal{Z}} \mathrm{e}^{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{b_i v_i}} \sum_{h_1} \mathrm{e}^{\mathrm{h_1}\left(\mathrm{c_1}+\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_{1i} v_i}\right)} \sum_{h_2} \mathrm{e}^{\mathrm{h_2}\left(\mathrm{c_2}+\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_{2i} v_i}\right)} \cdots \sum_{h_m} \mathrm{e}^{\mathrm{h_m}\left(\mathrm{c_m}+\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_{mi} v_i}\right)} \\
&= \frac{1}{\mathcal{Z}} \mathrm{e}^{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{b_i v_i}} \prod_{j=1}^{m} \sum_{h_j} \mathrm{e}^{\mathrm{h_j}\left(\mathrm{c_j}+\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_{ij} v_i}\right)} \\
&= \frac{1}{\mathcal{Z}} \prod_{i=1}^{n} \mathrm{e}^{\mathrm{b_i v_i}} \prod_{j=1}^{m} \left(1 + \mathrm{e}^{\mathrm{c_j}+\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w_{ij} v_i}}\right)
\end{aligned}
\tag{2.18}
$$

La ecuación anterior define el comportamiento de una RBM (marginalizada) como un modelo del tipo *productos de expertos*, donde el número de expertos de cada componente individual son combinados de forma multiplicativa. Cualquier distribución sobre $\{0,1\}^n$ puede ser modelada arbitrariamente bien por una RBM con n unidades visibles y $k+1$ unidades ocultas donde k denota la cardinalidad del conjunto soporte de la distribución objetivo, esto es, el número de elementos de entrada de $\{0,1\}^n$ que tienen probabilidad no nula de ser observados [14].

2.4.2. El gradiente de la función de verosimilitud

El gradiente 2.9 posee dos términos. En primer lugar se encuentra, el valor esperado del gradiente de energía sobre la distribución condicional de las variables ocultas dado un ejemplo de entrenamiento $\mathbf{v}$, el cual puede ser computado eficientemente debido a la simplificación del mismo. Por ejemplo, para el parámetro w_{ij} tenemos:

$$
\begin{aligned}
\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} &= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) h_i v_j \\
&= \sum_{\mathbf{h}} \prod_{k=1}^{n} p(h_k|\mathbf{v}) h_i v_j \\
&= \sum_{h_i} \sum_{h_{-i}} p(h_i|\mathbf{v}) p(\mathbf{h}_{-i}|\mathbf{v}) h_i v_j \\
&= \sum_{h_i} p(h_i|\mathbf{v}) h_i v_j \underbrace{\sum_{h_{-i}} p(\mathbf{h}_{-i}|\mathbf{v})}_{=1} \\
&= p(H_i = 1|\mathbf{v}) v_j \\
&= \sigma\left(\sum_{j=1}^{m} w_{ij} v_j + c_i\right) v_j
\end{aligned}
\tag{2.19}
$$

El segundo termino de la ecuación 2.9, esperanza del gradiente de la energía sobre la distribución de una RBM, puede ser reescrito como,

$$
\sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \qquad \text{o bien} \qquad \sum_{\mathbf{h}} p(\mathbf{h}) \sum_{\mathbf{v}} p(\mathbf{v}|\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \tag{2.20}
$$

también se reduce el costo computacional al aplicarse el mismo tipo de factorización en la sumatoria interna; de todas formas el cálculo permanece intratable para RBMs de tamaño regular debido a que su complejidad crece de manera exponencial con el tamaño de las capas. Esto es, la suma externa al igual que la interna recorre 2^n o 2^m estados posibles para las unidades binarias.

Para ejemplificar esta cuestión, consideremos una RBM con 1000 unidades visibles y 500 ocultas. Las dos operaciones de suma recorren todos los posibles valores que pueden tomar cada unidad, en este caso tenemos $2^{1000} \times 2^{500} = 2^{1500}$ combinaciones distintas. Por otra parte, este cálculo debe realizarse para cada actualización de la red.

Entonces, el gradiente de la función de verosimilitud para un solo patrón de entrenamiento $\mathbf{v}$ con respecto a w_{ij} es:

$$
\begin{aligned}
\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v})}{\partial w_{ij}} &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) h_i v_j - \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) h_i v_j \\
&= p(H_i = 1|\mathbf{v}) v_j - \sum_{\mathbf{v}} p(\mathbf{v}) p(H_i = 1|\mathbf{v}) v_j
\end{aligned}
\tag{2.21}
$$

El valor medio de esta derivada sobre el conjunto de entrenamiento $S = \{v_1, \ldots, v_l\}$ a menudo puede encontrase en la bibliografía con la siguiente notación [21, 20],

$$\begin{aligned}\frac{1}{l}\sum_{\mathbf{v}\in\mathbf{S}}\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v})}{\partial w_{ij}} &= \frac{1}{l}\sum_{\mathbf{v}\in\mathbf{S}}\left[-\mathbb{E}_{p(\mathbf{h}|\mathbf{v})}\left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right]+\mathbb{E}_{p(\mathbf{v},\mathbf{h})}\left[\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right]\right] \\ &= \frac{1}{l}\sum_{\mathbf{v}\in\mathbf{S}}\left[-\mathbb{E}_{p(\mathbf{h}|\mathbf{v})}\left[v_i h_j\right]+\mathbb{E}_{p(\mathbf{v},\mathbf{h})}\left[v_i h_j\right]\right] \\ &= \langle v_i h_j\rangle_{p(\mathbf{h}|\mathbf{v})Q(\mathbf{v})} - \langle v_i h_j\rangle_{p(\mathbf{h},\mathbf{v})} \\ &\propto \langle v_i h_j\rangle_{\text{datos}} - \langle v_i h_j\rangle_{\text{modelo}} = \textit{fase positiva} - \textit{fase negativa}\end{aligned} \tag{2.22}$$

donde Q representa la distribución empírica (datos) [14, 3]. El operador $\langle\cdot\rangle$ denota el valor esperado.

De manera análoga a la ecuación 2.21 se pueden obtener los gradientes con respecto a los parámetros restantes de la red. Para el parámetro b_i de la i-ésima unidad visible se tiene,

$$\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v})}{\partial b_i} = v_i - \sum_{\mathbf{h}} p(\mathbf{v}) v_i \tag{2.23}$$

y para el parámetro c_j de la j-ésima unidad oculta,

$$\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{v})}{\partial c_j} = p(H_j = 1|\mathbf{v}) - \sum_{\mathbf{v}} p(H_j = 1|\mathbf{v}) \tag{2.24}$$

Es imperativo evitar la suma de orden exponencial del segundo termino de las ecuaciones 2.21 y 2.23. Para llevar a cabo esto, se aproximan dichos términos por medio del muestreo de Gibbs ya mencionado.

2.4.3. Aproximar el gradiente de la función de verosimilitud

Todos los algoritmos de entrenamiento para las RBMs aproximan el gradiente de la función de verosimilitud para el conjunto de datos, que es requerido por el algoritmo de optimización, *gradiente ascendente estocástico* [3, 9, 22]. Nos enfocaremos en un único algoritmo de entrenamiento, en este caso **Divergencia Contrastiva** (CD), la cual desarrollaremos a en la próxima sección.

Muestreo de Gibbs en RBMs

Muestrear en una RBM es requerido por los algoritmos de aprendizaje, es un proceso necesario para capturar la distribución empírica de los datos con que la red entrena [14].

Extenderemos el desarrollo de la sección 2.3 para las RBMs. La independencia condicional entre las variables de la misma capa hace que el muestreo de Gibbs sea sencillo de ejecutar: en vez de muestrear nuevos valores para todas las variables de forma secuencial, los estados son muestreados de forma conjunta. Además, el proceso de muestreo puede realizarse en solo dos pasos: primero inferir una muestra del nuevo estado de $\mathbf{h}$ para las unidades ocultas dado $p(\mathbf{h}|\mathbf{v})$, por último tomar una muestra del estado de $\mathbf{v}$ para las unidades de la capa visible basado en $p(\mathbf{v}|\mathbf{h})$. Este procedimiento se conoce como *muestreo Gibbs por bloques* [25, 3].

Para t pasos de Gibbs, empezando por un ejemplo de entrenamiento (la distribución empírica se definida como $\hat{P}$, la distribución del modelo P) se tiene:

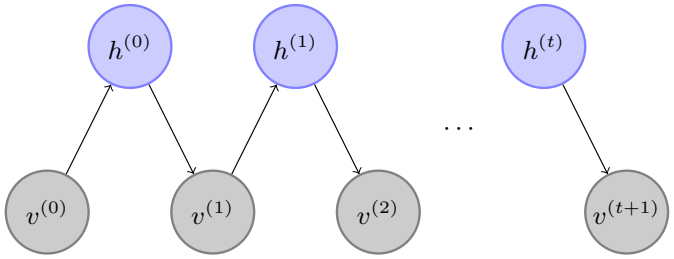


Fig. 2.2 Muestreo de Gibbs para una RBM con unidades visibles $\mathbf{v}$ y ocultas $\mathbf{h}$. La cadena inicia con $\mathbf{v}^0$: dato del conjunto de entrenamiento. El proceso itera t veces hasta la distribución estacionaria.

$$\begin{aligned}
\mathbf{v}^0 &\sim \hat{P}(\mathbf{v}) \\
\mathbf{h}^0 &\sim P(\mathbf{h}|\mathbf{v}^0) \\
\mathbf{v}^1 &\sim P(\mathbf{v}|\mathbf{h}^0) \\
\mathbf{h}^1 &\sim P(\mathbf{h}|\mathbf{v}^1) \\
&\vdots \\
\mathbf{v}^t &\sim P(\mathbf{v}|\mathbf{h}^{t-1}) \\
\mathbf{h}^t &\sim P(\mathbf{h}|\mathbf{v}^t) \\
\mathbf{v}^{t+1} &\sim P(\mathbf{v}|\mathbf{h}^t) \\
&\vdots
\end{aligned} \tag{2.25}$$

donde $\hat{P}$ corresponde a la distribución de los datos, P se asocia a la distribución del modelo. Tiene sentido comenzar la cadena desde un *ejemplo* debido a que el modelo se vuelve cada vez mejor al capturar la distribución de los datos de entrenamiento, las distribuciones P y $\hat{P}$ son cada vez más similares. El proceso puede representarse gráficamente, tal como se observa en la figura 2.2. Notar que si se comienza la cadena desde P en si misma, se debería converger en un solo paso de muestreo, por lo que comenzar desde $\hat{P}$ es un buen camino para asegurar la convergencia en solo pocos pasos [3].

2.4.4. Divergencia Contrastiva

Obtener estimadores puros del gradiente de la función de verosimilitud logarítmica utilizando métodos MCMC típicamente requieren muchos pasos de muestreo (hasta el equilibrio del modelo) por lo que es costoso a nivel a nivel de cómputos en aplicaciones prácticas. Sin embargo, se ha mostrado que pueden obtenerse estimadores lo suficientemente "aceptables" ejecutando una cadena de Markov de tan solo unos pocos pasos. Esta técnica es denominada *aprendizaje por divergencia contrastiva* el cual es un caso particular del muestro de Gibbs, se ha vuelto un estándar en algoritmos de entrenamiento para RBMs [3, 14, 18, 21].

La idea del aprendizaje por divergencia contrastiva de k-pasos (CD-k) es simple: en vez de aproximar el segundo termino de gradiente 2.21 y 2.23 desde la distribución del modelo (lo cual requiere correr una cadena de Markov hasta que la se llegue a la estacionalidad en la distribución), simplemente se ejecutan una cantidad finita y reducida de pasos k (usualmente $k = 1$) y el algoritmo

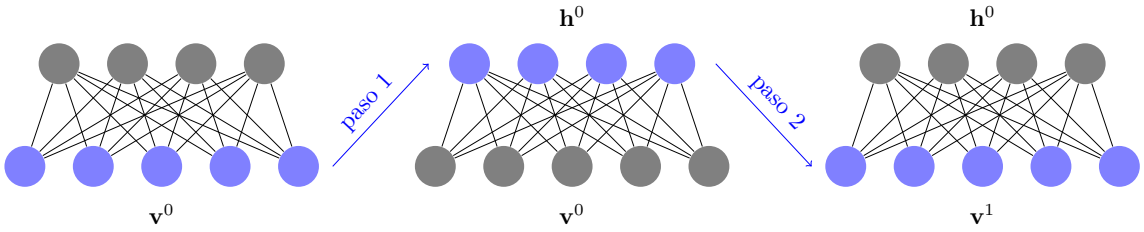


Fig. 2.3 Divergencia Contrastiva, inicio en $\mathbf{v}^0$ hasta k pasos $\mathbf{v}^k$. En este ejemplo $k = 1 \rightarrow CD_1$.

no espera llegar al equilibrio estacionario. La cadena de Gibbs comienza con un ejemplo $\mathbf{v}^{(0)}$ del conjunto de entrenamiento y continua hasta tomar la muestra $\mathbf{v}^{(k)}$ después de k pasos. Cada paso consiste en tomar una muestra $\mathbf{h}^{(k)}$ de $p(\mathbf{h}|\mathbf{v}^k)$ y seguidamente tomar una muestra $\mathbf{v}^k$ desde $p(\mathbf{v}|\mathbf{h}^k)$. El gradiente 2.9 con respecto a los parámetros $\boldsymbol{\theta}$ de la función de verosimilitud logarítmica para un patrón de entrenamiento $\mathbf{v}^0$ es aproximado por,

$$CD_k(\boldsymbol{\theta}, \mathbf{v}^0) = -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}^0)\frac{\partial E(\mathbf{v}^0, \mathbf{h})}{\partial \boldsymbol{\theta}} + \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}^k)\frac{\partial E(\mathbf{v}^k, \mathbf{h})}{\partial \theta} \tag{2.26}$$

Las derivadas en la dirección de cada parámetro son obtenidas por "estimación" de las esperanzas sobre $p(\mathbf{v})$ en (2.21 y 2.23) gracias a una sola muestra $\mathbf{v}^{(k)}$. En su versión *batch*, un conjunto entero de datos S es utilizado para el cálculo de la aproximación del gradiente en cada paso. Sin embargo, puede realizarse dicha operación de forma más eficiente tan solo considerando un subconjunto $S' \subset S$ en cada iteración, lo cual reduce la sobrecarga computacional en la actualización para cada uno de los parámetros. El subconjunto S' es llamado *mini-batch*. Si este contiene un solo elemento (dato) para la estimación del gradiente, todo el proceso es denominado *aprendizaje online*. El pseudo-algoritmo puede verse en el algoritmo 2.1. Mientras que una representación gráfica del mismo en la figura 2.3.

Típicamente la distribución estacionaria no es alcanzada después de k-pasos de muestreo. Por lo que $\mathbf{v}^{(k)}$ no es una muestra de la distribución del modelo, sino que es una muestra parcial del mismo [4, 14]. Esta no tiene en cuenta el termino bias, ya que el mismo se desvanece a medida que $k \rightarrow \infty$. El aprendizaje de la red por medio de CD produce buenos resultados a pesar de que solo una estimación del gradiente de la función de verosimilitud logarítmica para los datos de entrenamiento [18, 46]. El método reduce el tiempo necesario para el entrenamiento de una RBM y es ampliamente aplicado en la practica [21].
La representación gráfica del muestreo puede observarse en la figura 2.3.

Divergencia Contrastiva Persistente

Una muy simple y ligera modificación en CD, insertando la propiedad de memoria de estados histórico genera la versión *persistente* del algoritmo, denominado **Divergencia Contrastiva Persistente** (PCD).

La única diferencia con CD se destaca al inicio cada actualización, en vez de inicializar $\mathbf{v}^0$ con un patrón del conjunto de entrenamiento, se almacena o *persiste* el estado del batch anterior en dichas unidades. Esto brinda la capacidad de memoria para el modelo, y se estima que a medida de sucesivas iteraciones de actualización la distribución $\hat{P}$ se acerca cada vez más a P [18, 3].

Algoritmo 2.1 Divergencia Contrastiva de k-pasos

Entrada: RBM $(V_1, \cdots, V_m, H_1, \cdots, H_n)$, conjunto entrenamiento S
Salida: Aproximación al gradiente $\Delta w_{ij}, \Delta b_i, \Delta c_j \quad \forall i, j \quad i = 1, \cdots, n; j = 1, \cdots, m$ inicializar $\Delta w_{ij} = \Delta b_i = \Delta c_j = 0 \quad$ para $\quad i = 1, \cdots, n; j = 1, \cdots, m$

función CD(V, H, S)
  para todo $\mathbf{v} \in S$ **hacer**
    $\mathbf{v}^0 \leftarrow \mathbf{v}$
    para $t = 0, \cdots, k-1$ **hacer**
      para $i = 1, \cdots, n$ **hacer** ▷ toma una muestra
        ${h_i}^{(t)} \sim p(h_i|\mathbf{v}^t)$
        $\vdots$
      fin para
      para $j = 1, \cdots, m$ **hacer** ▷ toma una muestra
        ${v_j}^{(t+1)} \sim p(v_j|\mathbf{h}^t)$
        $\vdots$
      fin para
    fin para
    para $i = 1, \cdots, n, \quad j = 1, \cdots, m$ **hacer**
      $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1|\mathbf{v}^0)v_j^{(0)} - p(H_i = 1|\mathbf{v}^k)v_j^{(k)}$
    fin para
    para $j = 1, \cdots, m$ **hacer**
      $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$
    fin para
    para $i = 1, \cdots, n$ **hacer**
      $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1|\mathbf{v}^0) - p(H_i = 1|\mathbf{v}^k)$
    fin para
  fin para
fin función

Visible	Oculta	Denominación	Referencia
Binaria	Binaria	RBM Binaria	[18]
Gaussiana	Binaria	RBM Gaussiana	[53]
Jerárquica	Binaria	RBM Jerárquica	[42]
Jerárquica Múltiple	Binaria	Softmax replicada/ LDA no dirigido	[23]
Gaussiana	Gaussiana	PCA no dirigido	[35]
Binaria	Gaussiana	PCA no dirigido binario	[53]

Tabla 2.1 Clases más comunes de Máquinas de Boltzmann Restringidas. La distinción es con respecto al tipo de unidades visibles/ocultas que poseen.

2.4.5. RBM Gaussiana: unidades visibles de valores reales

Hasta ahora hemos considerado solo observaciones representadas por medio de vectores binarios, pero a menudo deben ser considerados otro tipo de modelos, los cuales se rigen por valores reales. Basta con modificar levemente la función de energía 2.11 por una que contemple los valores continuos, reemplazando el espacio de estados posibles $\{0,1\}^n$ de $\mathbf{V}$ por uno $[0,1]^n$. De manera análoga pueden derivarse las ecuaciones 2.16 y 2.17 para unidades gaussiana (solo cambia la unidad visible, por lo que la ecuación 2.16 es idéntica a la RBMs Binarias). La función de energía debe tener la forma de,

$$E(\mathbf{v},\mathbf{h}) = -\sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_j c_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij} \tag{2.27}$$

donde σ_i^2 es la varianza para la unidad i. El resto de los parámetros, $\{\mathbf{v},\mathbf{h},\mathbf{b},\mathbf{c},\mathbf{w}\}$ se corresponden a las RBM binarias ya mencionados.

Existen formas de sobrepasar esta limitación (cambiar las unidades), solo se debe adaptar un modelo de RBM binarias para que sea capaz de representar una distribución continua por medio de un simple "truco". Los datos de entrada son normalizados al rango [0,1] y la probabilidad modelada por las unidades visibles también se deben escalar. Esto es, en vez de tomar una muestra sobre vectores binarios, la probabilidad $p(V_i = 1|\mathbf{h})$ es considerada como el estado actual de la variable V_i. A excepción de que las variables visibles se fijan a valores continuos, el resto del proceso de entrenamiento permanece sin cambios [22].

Cuando se utilizan unidades Gaussianas, la tasa de aprendizaje debe ser pequeña [18, 21], además no hay límites inferiores o superiores (no acotados) a los que la unidades pueden alcanzar y por ende el modelo puede diverger por falla precisión numérica.

2.4.6. Otras clases

En la bibliografía puede hallarse diversas clases de RBMs, en general reciben el nombre según el tipo de unidades que las componen. RBMs más comunes se encuentran en la tabla 2.1.

2.4.7. Análisis de comportamiento del modelo

Considerando la función de energía 2.1, si existiese una transformación capaz de mapear cada configuración de la red según los parámetros $\boldsymbol{\theta}$ sobre el eje coordenado, la gráfica correspondiente al plano bidimensional tendría una forma similar a la figura 2.4. Bajo repetidas actualizaciones, la red tiende a modificar su energía asignando valores cada vez menores a aquellas configuraciones que

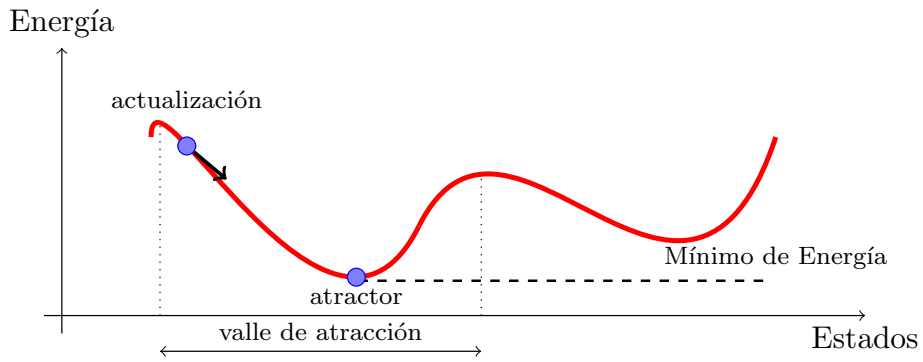


Fig. 2.4 Valle de energía de una RBM en su forma más general, la región encerrada por las lineas punteadas debajo de la curva de energía, denominada valle de atracción, el estado actual de la red y su actualización hacia un punto atractor, a la que con el tiempo la red converge. Dicho estado representa un mínimo de energía.

generen distribución de probabilidad más símiles a las de los patrones de entrenamiento. Por otro lado, asigna valores superiores a configuraciones que representen de forma errática la distribución de los datos. Dichos conjuntos de valores asociados se conocen como valle de energía, donde los primeros conforman "mínimos locales", también llamados punto *atractor* a cada uno de ellos, analogía a las redes de Hopfield [3, 21].

2.4.8. Consideraciones prácticas

A continuación se mencionan algunas cuestiones prácticas a tener en cuenta a la hora de efectuar el entrenamiento de una máquina de Boltzmann restringida. Estas fueron proporcionadas por Hinton a través de años de experiencia en [21].

Inicialización de los pesos y bias de la red

Al comienzo del aprendizaje, es recomendable asignar a la matriz de pesos valores muestreados desde una distribución Gaussiana con media nula y desviación estándar alrededor de 0.01. Los bias de las unidades visibles inicializados con $p_i/(1-p_i)$, donde p_i es el porcentaje de los puntos de entrenamiento sobre las unidades visibles i. Por otro lado, para los bias de las unidades ocultas se recomienda fijarlos con valores nulos, esto produce buenos resultados en la práctica [21].

Controlar el progreso de aprendizaje

Es importante tener conocimiento de la evolución del aprendizaje es útil para determinar si es necesario finalizar el proceso en caso de potencial sobre-ajuste, así aumentar el poder de generalización de la red. Para tal fin se pueden monitorear el error de reconstrucción, la energía libre o los filtros de las unidades ocultas.

Error de reconstrucción

Computar el error cuadrático medio entre el conjunto de datos y las reconstrucciones (estados de las unidades visibles) de la red puede dar una buena idea sobre el progreso de aprendizaje: el error de reconstrucción típicamente decrece de forma rápida y constante al inicio del proceso de aprendizaje y luego oscila suavemente debido al ruido producido por los mini-batches. A pesar que parezca apropiado valerse del error de reconstrucción como "estadístico" de aprendizaje, el mismo es un mal indicador. El algoritmo de divergencia contrastiva no optimiza dicho valor, por lo que no se ve reflejado en ello [21].

Energía libre

Una manera sencilla y eficaz de comprobar si la red está "sobre-aprendiendo" los patrones (perdiendo generalización) es realizar la comparación entre los vectores de energía libre, ecuación 2.12, a partir del conjunto de datos de entrenamiento y validación [21]. Si la diferencia entre las dos energías libres aumenta o crece (la razón entre la energía libre de los datos de validación y la energía libre de los datos de entrenamiento es significativamente mayor que la unidad), entonces el modelo comienza a perder generalización, una propiedad que en general no es deseada. Notar que las magnitudes de la energía libre no tiene significado alguno, aunque la razón entre ellas es de utilidad (en la cual la función de partición $\mathcal{Z}$ se cancela).

Examinar los filtros

Para muchos dominios donde las unidades visibles cuentan con una estructura temporal o espacial (por ejemplo en imágenes o el habla) resulta muy útil graficar para cada unidad oculta, los pesos que conectan dichas unidades con las respectivas unidades visibles. Los gráficos generados son llamados *filtros*.

Los filtros de las unidades ocultas pueden ser vistas como detectores de características. En el domino de la visión computacional, dada una unidad visible y su correspondiente píxel en la imagen, surge la siguiente interrogante ¿Qué forma deben poseer las imágenes observadas para activar una unidad oculta más que las otras? La respuesta a la misma se corresponde a un arreglo de dibujos (cuadrados) con respecto a los pesos de la red los cuales conectan a una unidad oculta según la conexión con las unidades visibles correspondientes. La imagen resultante visualiza el patrón de entrada, el cual es llamado como filtro de aprendizaje o, en términos biológicos, el campo receptivo. Esos campos receptores son un buen camino para visualizar que características las unidades ocultas estan "aprendiendo".

Cuando es necesario graficar los campos receptivos de muchas unidades ocultas, es conveniente utilizar diferentes escalas de grises, es menos atractivo visualmente pero resulta más informativo que coloridos y enredados gráficos [21].

Generalización: Dropout

Al igual que las redes neuronales clásicas, las técnicas para ganar generalización también pueden ser aplicadas a las RBMs. Una que ha dados buenos resultados prácticos es *dropout*.

Descripción del modelo con dropout Considerando una RBM binaria con unidades visibles $\mathbf{v} \in \{0,1\}^n$ y unidades ocultas $\mathbf{h} \in \{0,1\}^m$. Se define la distribución de probabilidad de la sección 2.4.1 con mayor detalle,

$$p(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}) = \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \exp(\mathbf{v}^\top W \mathbf{h} + \mathbf{c}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}).$$

donde $\boldsymbol{\theta} = \{W, \mathbf{b}, \mathbf{c}\}$ son los parámetros del modelo. El termino $\mathcal{Z}$ es la función de partición.

A la definición estándar de RBM anterior, es acopla un vector de variables aleatorias binarias $\mathbf{r} \in \{0,1\}^m$. Cada variable aleatoria r_j toma el valor unitario con la probabilidad p, independiente de las otras. Si $r_j = 1$, la unidad oculta h_j es retenida, en otro caso la misma es desechada del modelo para la iteración actual [44]. Una ilustración puede verse en la figura 2.5 La distribución conjunta para una RBM con dropout puede expresarse como alguna de las siguientes formas,

$$\begin{aligned}
p(\mathbf{r}, \mathbf{h}, \mathbf{v}; p, \boldsymbol{\theta}) &= p(\mathbf{r}; p) p(\mathbf{h}, \mathbf{v} | \mathbf{r}; \boldsymbol{\theta}) \\
p(\mathbf{r}; p) &= \prod_{j=1}^{m} p^{r_j} (1-p)^{1-r_j} \\
p(\mathbf{h}, \mathbf{v} | \mathbf{r}; \theta) &= \frac{1}{\mathcal{Z}'(\boldsymbol{\theta}, \mathbf{r})} \exp(\mathbf{v}^\top W \mathbf{h} + \mathbf{c}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}) \prod_{j=1}^{m} g(h_j, r_j) \\
g(h_j, r_j) &= \mathbf{1}(r_j = 1) + \mathbf{1}(r_j = 0)\mathbf{1}(h_j = 0)
\end{aligned}$$

donde $\mathcal{Z}'(\theta, \mathbf{r})$ es la constante de normalización. $g(h_j, r_j)$ impone la condición de que si $r_j = 0$, entonces h_j debe ser nulo. La distribución sobre $\mathbf{h}$ dada $\mathbf{v}$ y $\mathbf{r}$ es el factorial,

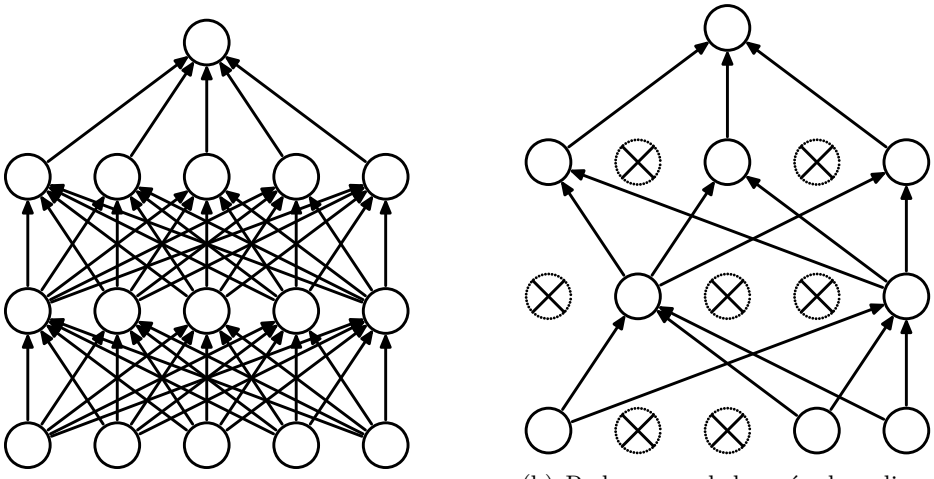
$$\begin{aligned}
p(\mathbf{h} | \mathbf{r}, \mathbf{v}) &= \prod_{j=1}^{n} p(h_j | r_j, \mathbf{v}) \\
p(h_j = 1 | r_j, \mathbf{v}) &= \mathbf{1}(r_j = 1) \sigma \left(c_j + \sum_i W_{ij} v_i \right)
\end{aligned} \tag{2.28}$$

donde la distribución sobre $\mathbf{v}$ dada $\mathbf{h}$ es la misma que la RBM de las ecuaciones 2.17 y 2.14.

Tasa de dropout La técnica de dropout introduce un hiperparámetro extra, la probabilidad p de retener una unidad. Este controla la intensidad de eliminación, con $p = 1$ no existe dropout y a bajos valores de p significa mayor dropout. Valores típicos para las unidades ocultas están dentro del rango [0.5,0.8]. Para capas de entrada, la elección depende del tipo de la misma, en caso de valores reales (imagen o habla por ejemplo) un valor típico es de 0.8. Para unidades ocultas, la elección de p está acoplada a la cantidad de unidades ocultas m. Cuando p es pequeño, se requiere que m sea mayor, lo cual hace que el entrenamiento se ralentice y conduce al no ajuste de los pesos. Con p grande puede que no se produzca suficiente dropout, lo cual previene el sobre-entrenamiento [44].

2.5. Redes de Creencia Profunda

Al igual que las RBMs, las redes de creencia profunda pertenecen a la categoría de modelos generativos y probabilísticos, se valen de variables ocultas para aprender las características de los datos con cual se la entrena. A diferencia de las primeras, las DBNs se componen por varias capas de unidades



(a) Red neuronal estándar

(b) Red neuronal después de aplicar Dropout.

Fig. 2.5 **Izquierda**: Una red neuronal estándar con dos capas ocultas. **Derecha**: Topología similar a la red (a), a la cual se le aplico dropout. La unidades con una cruz han sido eliminadas. Imagen extraída de [44].

ocultas. Esta propiedad permite al modelo aprender representaciones de mayor nivel junto con la relación interna de dichas representaciones, como si las mismas se pudiesen organizar de forma jerárquica [19, 42].

El comportamiento anterior es objeto de interés debido a que es asociado al comportamiento de como los seres humanos percibimos nuestro entorno que nos rodea. Para ejemplificar esto, consideremos una situación del campo de la visión computacional, en primer lugar percibimos la unidad, a nivel *píxel*, luego en un nivel superior *trazos*, seguido de detección de *bordes*, y así hasta el ensamblando de *objetos* para culminar en la formación del *contexto* para los mismo. Este proceso es repetido hasta que la información en conjunto sea lo suficientemente extensa para transmitir un mensaje [3, 54]. La idea básica se ejemplifica en la figura 2.6, donde a partir de un vector crudo de entrada, se transforma gradualmente en representaciones de mayor nivel a medida en que se asciende en la jerarquía de abstracción, hasta conformar el mensaje (hombre sentado...).

Las DBNs fueron originalmente concebidas con unidades binarias estocásticas, en la actualidad existe una extensa variedad de las mismas. Las variantes están dadas como se verá a continuación, por los bloques de RBMs por la que las DBNs se componen. En la figura 2.7 puede observarse la representación gráfica de este modelo.

Modelo Híbrido

En una red de creencia profunda, l capas modelan una distribución conjunta entre los vectores observados $\mathbf{v}$ y los ocultos $\mathbf{h}^k$ como sigue:

$$p(\mathbf{v}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = \left(\prod_{k=0}^{l-2} p(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) p(\mathbf{h}^{l-1}, \mathbf{h}^l) \tag{2.29}$$

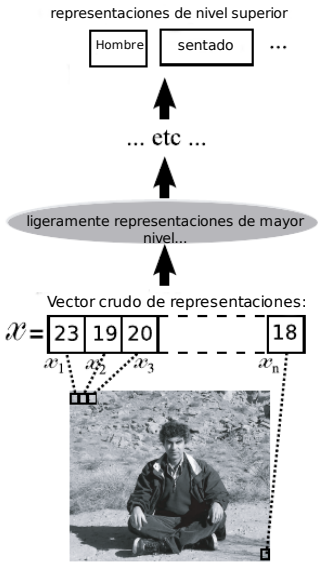


Fig. 2.6 Representaciones Jerárquicas, se transforma gradualmente en representaciones de mayor nivel, representado funciones asociadas al vector de entrada (píxeles, bordes, trazos, etc) hasta llegar al nivel superior en donde el mensaje es transmitido (Hombre sentado...). Imagen extraída de [3].

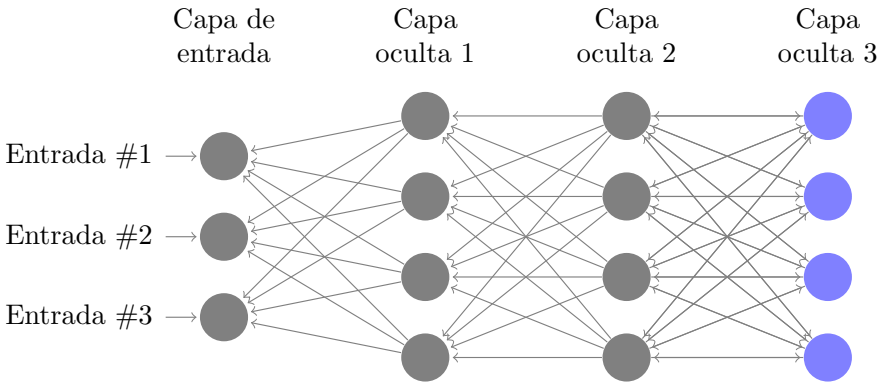


Fig. 2.7 Red de creencia profunda. Notar que solo los pesos son bidireccionales entre las capas 2 y 3 (últimas capas)

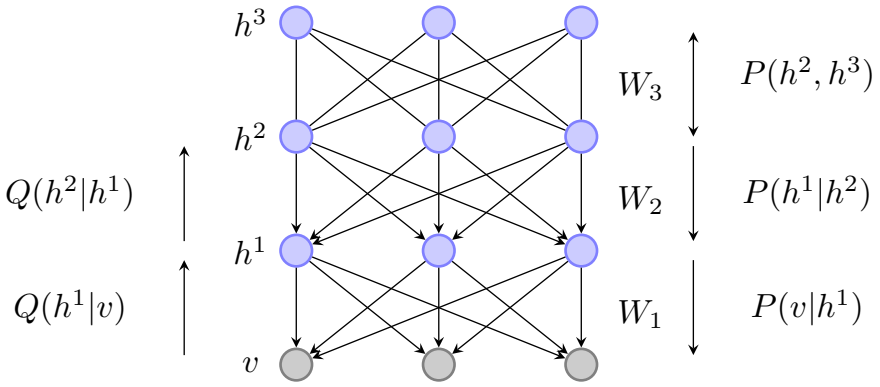


Fig. 2.8 Una red de creencia profunda como modelo generativo (el camino generativo denotado con las distribuciones P y flechas hacia abajo) y las distribuciones que extraen múltiples niveles de representación de las entradas (camino de "reconocimiento" denotado con Q, flechas hacia arriba). El nivel superior, compuesto por las capas h^2 y h^3 conforman una RBM. Las capas inferiores a estas últimas forman un modelo de grafo dirigido.

donde $\mathbf{v} = \mathbf{h}^0, p(\mathbf{h}^{k-1}|\mathbf{h}^k)$ es una distribución condicional *visible dada una oculta* asociada a una RBM del nivel k de una DBN, y $p(\mathbf{h}^{l-1}, \mathbf{h}^l)$ es la distribución conjunta de la RBM del nivel superior. La representación gráfica del modela puede encontrarse en la figura 2.8.

La distribución condicional $P(\mathbf{h}^k|\mathbf{h}^{k+1})$ y la distribución conjunta del nivel superior $P(\mathbf{h}^{l-1}, \mathbf{h}^l)$ definen el modelo generativo. Introduciendo Q para denotar la aproximación o valor exacto de las probabilidades a posterior del modelo, utilizadas para tareas de inferencia en el entrenamiento. Las probabilidades Q, son todas aproximaciones a excepción en la capa superior $Q(\mathbf{h}^l|\mathbf{h}^{l-1})$, donde es igual a $P(\mathbf{h}^l|\mathbf{h}^{l-1})$ debido a que $(\mathbf{h}^l|\mathbf{h}^{l-1})$ forma parte de una RBM, donde la inferencia exacta es posible de calcular [3].

Cuando se entrena una DBN capa por capa por medio de un algoritmo heurístico[4] (ver algoritmo 2.2), cada una es inicializada como una RBM, aquí denotamos $Q(\mathbf{h}^k, \mathbf{h}^{k-1})$ como la k-ésima RBM entrenada de esta manera, donde $P(\dots)$ describe las probabilidades según la DBN. Utilizando $Q(\mathbf{h}^k, \mathbf{h}^{k-1})$ como una aproximación a $P(\mathbf{h}^k, \mathbf{h}^{k-1})$ (más sencillo de computar) luego de tomar una muestra desde $Q(\mathbf{h}^k, \mathbf{h}^{k-1})$ [3, 4]. Estos $Q(\mathbf{h}^k, \mathbf{h}^{k-1})$ pueden ser utilizados para construir una representación de los vectores de entrada $\mathbf{v}$. Para obtener una representación de todos los niveles, utilizamos el siguiente procedimiento. Primero tomamos una muestra $\mathbf{h}^1 \sim Q(\mathbf{h}^1|\mathbf{v})$ desde el primer nivel de RBM, o alternativamente utilizando el enfoque de campos-medios se puede computar $\hat{\mathbf{h}}^1 = E[\mathbf{h}^1|\mathbf{v}]$ en lugar de tomar una muestra de $\mathbf{h}^1$, donde el valor esperado sobre la distribución $Q(\mathbf{h}^1|\mathbf{v})$ de una RBM. Como resultado se obtiene justamente el vector de salidas para las unidades ocultas, que en el caso común (RBM binarias): $\hat{\mathbf{h}}^1_i = \sigma(\mathbf{b}^1 + W^1_i \mathbf{v})$. Luego se toman los vectores de campo-medio $\hat{\mathbf{h}}^1$ o bien las muestras $\mathbf{h}^1$ como entrada al siguiente nivel de RBM, se computa $\hat{\mathbf{h}}^2$ o $\mathbf{h}^2$ y así sucesivamente como tantas capas se las DBN posee.

[4]Heurístico: en computación, dos objetivos fundamentales son encontrar algoritmos con bajos tiempos de ejecución y soluciones optimas. Una heurística es un algoritmo que abandona uno o ambos objetivos; por ejemplo, normalmente encuentran buenas soluciones, aunque no hay pruebas de que la solución no pueda ser arbitrariamente errónea en algunos casos

Como modelo discriminativo

De DBN entrenada de la forma en que se ejemplifica en el algoritmo 2.2, se pueden extraer los parámetros W^i (pesos de las RBMs) y $\mathbf{c}^i$ (bias ocultos de cada RBM) de cada capa y utilizarlos para inicializar una red multicapa profunda (perceptrón multicapa) [3, 22]. Dichos parámetros pueden ajustarse con respecto a otro criterio, típicamente un entrenamiento supervisado.

Por último, pueden aplicarse técnicas de generalización tradicionales como "dropout", ver sección 2.4.8. Es necesario realizar un ajuste de los pesos obtenidos en la fase de pre-entrenamiento heurístico, lo cual consiste en escalar la matriz de pesos W^l para la capa l con su correspondiente tasa de dropout p a un factor $1/p$ [44].

Resumiendo el modelo discriminativo Para entrenar la red, primero se debe realizar un pre-entrenamiento heurístico como fue descripto con anterioridad, para que la red aprenda las características de las características. Luego, con los pesos se aplica un algoritmo de ajuste fino como ser el "propagación hacia atrás" *(backpropagation)* [17] a toda la red con el objetivo de poder discriminar entre los datos etiquetados. Este enfoque tiende a disminuir muchos problemas del algoritmo de propagación hacia atrás puro:

- El algoritmo de backpropagation no necesita que aprender las características de los datos inicialmente. Esta tarea fue realizada paso previo por el pre-entrenamiento heurístico de la DBN. Disminuye el problema de desvanecimiento del gradiente [3].
- El algoritmo cuenta con menor probabilidad de quedar atrapado en un mínimo local de energía por su naturaleza estocástica e inicializacion de los pesos [19].
- Son requeridos menos datos etiquetados para alcanzar tasas de error similares. El entrenamiento previo (heurístico) no requiere datos etiquetados, por lo que es plenamente un entrenamiento no-supervisado. Los datos etiquetados son un recursos escasos, y obtenerlos requiere un gran esfuerzo manual. Requerir menor cantidad de datos etiquetados en cualquier algoritmo, abre las puertas a una mayor cantidad de conjuntos de datos [21].

Como modelo generativo

Una muestra $\mathbf{v}$ de un modelo de DBN generativo puede obtenerse de la siguiente forma:

1. Tomar una muestra del vector visible $\mathbf{h}^{l-1}$ desde la RBM superior. Esto puede realizarse aproximando mediante la ejecución de cadenas de Gibbs, en donde las mismas se alternan entre $\mathbf{h}^l \sim P(\mathbf{h}^l|\mathbf{h}^{l-1})$ y $\mathbf{h}^{l-1} \sim P(\mathbf{h}^{l-1}|\mathbf{h}^l)$ por medio de muestreo de Gibbs, (ver sección 2.4.3). Comenzando una cadena desde una representación $\mathbf{h}^{l-1}$ obtenida desde el conjunto de entrenamiento (como las Q inferiores), es posible que se requiera algunos pasos de Gibbs.
2. Para $k = l - 1$ hasta 1, tomar muestras $\mathbf{h}^{k-1}$ dado $\mathbf{h}^k$ de acuerdo al nivel k de la distribución condicional oculta-a-visible $P(\mathbf{h}^{k-1}|\mathbf{h}^k)$.
3. $\mathbf{v} = \mathbf{h}^0$ es la muestra de la DBN final.

Resumiendo el modelo generativo

- Obtener una muestra de la RBM superior cuando la misma llegue al equilibrio (realizando el muestreo alternado de Gibbs entre las dos capas superiores)

- Comenzando desde las capas ocultas superiores contiguas a las anteriores, utilizar los pesos arriba-abajo generativos para pasar a través de la red

Como una observación interesante, las DBNs son recursivas: removiendo la capa visible de una red de creencia profunda con más de tres capas, como resultado se obtiene otra red (DBN), solo que con una capa menos.

Algoritmo 2.2 Entrenamiento no supervisado DBN

Entrada: DBN$(\epsilon, l, \mathbf{W}^k, \mathbf{b}^k, \mathbf{c}^k, \hat{P})$

ϵ tasa de aprendizaje,
l numero de capas a entrenar,
$\mathbf{W}^k$ matriz de pesos para el nivel k, con $k = 1, \ldots, l$,
$\mathbf{b}^k$ vector de bias visibles para las RBM del nivel k,
$\mathbf{c}^k$ vector de bias ocultos para las RBM del nivel k,
$\hat{P}$ conjunto entrenamiento.

Salida: Matriz de pesos ajustados $\mathbf{W}^k$.

función DBN$(\epsilon, l, \mathbf{W}^k, \mathbf{b}^k, \mathbf{c}^k, \hat{P})$
  mientras criterio de parada no alcanzado **hacer**
    $\mathbf{h}^0 \sim \mathbf{v}$ ▷ toma una muestra de $\hat{P}$
    para $i = 1, \cdots, k-1$ **hacer**
      para $j = 1, \ldots, m; m \in \mathbf{h}^i$ **hacer**
        $\mathbf{h}^i_j \sim Q(\mathbf{h}^i_j = 1 | \mathbf{h}^{i-1})$ ▷ toma una muestra de Q
      fin para
    fin para
    RBMActualizar$(\mathbf{h}^{k-1}, \epsilon, W^k, \mathbf{b}^k, \mathbf{c}^k)$
  fin mientras
fin función

Capítulo 3

Desarrollo

Cumplir con uno de los objetivos establecidos, desarrollar una biblioteca que brinde una interfaz para la gestión de redes de creencia profundas, implica generar los procedimientos para entrenamiento y ajuste de las DBNs y consecuentemente a las entrenamiento para las RBMs. Con la idea de disminuir el tiempo en los procesos, la biblioteca debe ser capaz de aprovechar las funcionalidades de cómputo utilizando la unidad GP-GPU.

Se realizó una exploración de desarrollos similares al propuesto, estos pueden hallarse en Internet y documentos científicos. Una síntesis de los paquetes de software con más referentes en la actualidad junto a sus características puede encontrarse en la sección A.1. Para la implementación se tuvieron en cuenta varios aspectos, como ser el lenguaje de programación a utilizar, las librerías de apoyo para la comunicación con la GP-GPU, cuestiones de diseño, entre otros (ver apéndice A).

Con la propuesta de ampliar a la mayor cantidad de personas de la comunidad que puedan hacerse con la biblioteca, debe facilitarse su instalación. Es por ello que los requerimientos de software externo y no esencial deben ser mínimos, solo dependencias necesarias para el libre funcionamiento de la biblioteca reduciendo posibles conflictos con otras librerías.

En la fase de diseño, en primer lugar se debió analizar las herramientas y recursos disponibles. Por un lado se cuenta con el hardware de cómputo GP-GPU, en adelante *dispositivo*, donde se realizarán los cálculos masivos; la CPU, denominada *huésped* ejecuta el sistema operativo, procesos, etc. además de contener al dispositivo. La tecnología seleccionada encargada de gestionar los cálculos y pasos necesarios, llamada *Theano*, el cual brinda características de cómputo paralelo. Una vez relevados los aspectos fundamentales de las herramientas, se deben tener en cuenta las funcionalidades necesarias a implementar. Los algoritmos más críticos a desarrollar están relacionados al entrenamiento de las RBMs y DBNs, así como también el *ajuste fino* para estas últimas.

En las siguientes secciones se desarrollará una breve descripción sobre las unidades gráficas. También se contempló una síntesis de las decisiones tomadas para en la fase de implementación, las mismas abarcan la elección del lenguaje de programación y librerías soporte, cuestiones de diseño entre otras.

La biblioteca producida ha sido bautizada con el nombre de **CUPYDLE**, esta cubre con los objetivos propuestos al inicio del proyecto. Es de código abierto[1] y se encuentra en expansión de nuevas funcionalidades.

[1] El código fuente, instrucciones y detalles puede encontrarse el sitio: https://github.com/lerker/cupydle

.

3.1. Análisis

Existen muchas definiciones para requerimiento, tales como: “una condición o capacidad que debe estar presente en un sistema para satisfacer un contrato, estándar, especificación u otro documento formal”, “una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo”, o bien “una representación documentada de una condición o capacidad de los tipos enunciados anteriormente”. Se realiza el análisis de requerimientos en función de las necesidades que tiene que satisfacer el sistema.

Los requerimientos pueden dividirse en *funcionales* y *no funcionales*. Los primeros definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que produce sobre las entradas para generar las salidas. Los requerimientos no funcionales están relacionados con características que de una u otra forma pueden limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad, robustez del sistema, disponibilidad de equipo, mantenimiento, seguridad, etc.

A continuación se detallan los requerimientos de la biblioteca, algunos son heredados o provienen del los requerimientos del proyecto:

Requerimientos funcionales

1. Crear una red de creencia profunda, entrenarla de forma heurística por medio de métodos no supervisados.
2. Ajustar la red de creencia profunda con técnicas supervisadas mediante el algoritmo de propagación hacia atrás.
3. Crear una máquina de Boltzmann restringida, entrenarla de forma no supervisada.

El primer requerimiento se encuentra especificado en el objetivo principal del proyecto. El software final debe proveer la funcionalidad de crear y gestionar dichas redes, y de los métodos disponibles de entrenamiento debe elegirse uno heurístico a ser implementado.

El segundo requerimiento se encuentra relacionado al *ajuste fino* de la red. Una vez entrenada la misma por un método no supervisado, el usuario puede (o no) ajustarla con datos etiquetados. Con ello logrará mejores resultados para las tareas de clasificación o regresión.

Por último, el tercer requerimiento también presente en los objetivos del proyecto y el cual es necesario en la etapa de entrenamiento de las DBNs. Se debe brindar al usuario la funcionalidad de fijar individualmente los parámetros capa a capa de las DBNs, que consisten en RBMs propiamente. Por lo cual debe abstraerse la gestión de éstas últimas en las redes de creencia profunda para su entrenamiento.

Requerimientos no funcionales

1. Generar gráficos sobre los estadísticos retornados del proceso de entrenamiento.
2. Almacenar los parámetros, pesos de la red.
3. Manipular grandes volúmenes de datos en GP-GPU, automatizar la transferencia de datos entre el huésped y el dispositivo.
4. Requerir la menor cantidad posible de dependencias externas.

5. Los algoritmos deben ser robustos y veloces.
6. El software debe ser portable.

El primer requerimiento no funcional, generar gráficos de los estadísticos, es útil para obtener un panorama de forma rápida sobre el proceso de entrenamiento de la red. Con dicha funcionalidad el usuario recibe la *retroalimentacion* de manera más eficiente que sin ella, realizando así los cambios necesarios para el siguiente proceso o etapa de entrenamiento.

Para el siguiente requerimiento, persistir en disco los parámetros y datos de la red para el posterior análisis. Es necesario para portar los resultados hacia otro software o para su posterior análisis.

El tercer requerimiento, relacionado a la gestión de grandes volúmenes de datos es propio a las limitaciones del hardware (memoria de almacenamiento en la placa gráfica) y a problemas reales con los que se ponen a prueba el software. Debido a que la memoria de la GP-GPU es fija y limitada debe contemplarse la gestión de transferencia de datos de forma automática sin intervención del usuario para cuando estos exceden la capacidad de la memoria disponible.

Con respecto al cuarto, requerir la menor cantidad posible de dependencias externas, se asocia a la justificación del proyecto. Surge luego de observar que implementaciones de software con funcionalidades similares no eran posibles instalar en las PCs, debido a no poder satisfacer las dependencias de los mismos por incompatibilidades del sistema operativo, software ya presente, etc. Cuando menor sea la cantidad de software externo necesario para la ejecución e instalación de la biblioteca, mayor será el número de PCs donde podrá ser instalada sin dificultad.

Los dos últimos requerimientos se asocian también a la justificación del proyecto. Los algoritmos utilizados deben ser veloces con el objetivo de mejorar los tiempos de entrenamiento de DBNs en GP-GPU a alternativas iterativas.

3.2. Diseño

Un estándar para modelar los sistemas son las herramientas que proporciona UML (Unified Modeling Language), el cual es un lenguaje utilizado para describir o modelar cualquier sistema de información. UML provee un vocabulario y reglas que permiten representar sistemas mediante gráficos sencillos o texto y así obtener modelos explícitos del proceso. Estos modelos mejoran la comunicación cliente-desarrollador durante la fase de diseño, y luego ayudan en el desarrollo, al poder ser interpretados por personas ajenas del proyecto sin ninguna ambigüedad.

En esta sección, se describen funcionalidades de la biblioteca a partir de UML mediante los diagramas de casos de uso y clases. También se desarrollará un análisis sobre la librería de apoyo para la comunicación con la GP-GPU *Theano* junto a los diagramas necesarios.

Elección de herramientas UML

La selección de dichas herramientas está dada por la naturaleza del proyecto. El mismo no requiere de interfaz gráfica, es manipulado por un único usuario, los procesos son independientes entre sí, no requiere de agentes externos, entre otras características. La elección de otros diagramas no aportarán información relevante, solo ofuscarán las funcionalidades del sistema.

Los diagramas de casos de uso determinan los usuarios del sistema, límites de mismo y sus funcionalidades. Solo se expondrán casos de uso de aquellos elementos que resulten de principal interés. Los diagramas de clases es justificada debido al tipo de implementación a realizar, *orientado a objetos*.

Estos resultan útiles para representar objetos, relaciones entre ellos, así como también sus atributos y métodos. Los diagramas de grafos son utilizados para representar la mecánica de operación de Theano. No siguen ningún estándar de la UML, esto no es requerido ya que dichos diagramas son herramientas para la codificación interna de los métodos del programa.

3.2.1. Diagrama de Casos de Uso

Un caso de uso es la descripción de un conjunto de acciones que un sistema ejecuta y produce un determinado resultado de interés para un usuario. El diagrama de caso de uso se utiliza para capturar el comportamiento deseado del sistema en desarrollo, sin tener que especificar cómo se implementa dicho comportamiento [43]. En esta sección se desarrolla la funcionalidad básica del la biblioteca y las operaciones que realiza el usuario, por medio de los diagramas de caso de uso y sus descripciones textuales. Solo se expondrán los casos de uso de elementos que resulten de interés.

A continuación se explica en detalle los casos de uso mas destacados. En la figura 3.1 se observa el diagrama de caso de uso de la biblioteca.

```
CU01 - Crear DBN
CU02 - Agregar Capa
CU03 - Crear RBM
CU08 - Entrenar RBM
CU09 - Pre-entrenar DBN
CU11 - Crear MLP
CU15 - Entrenar MLP
CU10 - Ajuste DBN
```

Caso de Uso: CU01 - Crear DBN **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza cuando el usuario crea la clase DBN.	
2) El sistema se inicializa, comprobando la comunicación con la GP-GPU en caso de ser requerida.	2.1) No se puede establecer la comunicación con la GP-GPU. 2.2) Se informa al Usuario del incidente. El caso de uso finaliza.
3) El usuario tiene control sobre la DBN. El caso de uso finaliza.	

Tabla 3.1 Caso de Uso CU01 - Crear DBN

Caso de Uso: CU02 - Agregar Capa **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza cuando el usuario ingresa los parámetros para una nueva capa de la DBN. 2) Se invoca al caso de uso CU03 - Crear RBM 3) Se agregan los parámetros de la RBM al sistema. El caso de uso finaliza.	

Tabla 3.2 Caso de Uso CU02 - Agregar Capa

Caso de Uso: CU03 - Crear RBM **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza cuando el usuario agrega una nueva capa a la DBN. 2) Se inicializan los parámetros de la RBM. 3) El caso de uso finaliza.	

Tabla 3.3 Caso de Uso CU03 - Crear RBM

Caso de Uso CU09 - Pre-entrenar DBN **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza cuando el usuario ingresa datos no etiquetados para el entrenamiento de la DBN. 2) Por cada capa de la DBN se invoca el caso de uso CU08 - Entrenar RBM. 3) Se almacenan los pesos de las capas en la DBN. 4) El caso de uso finaliza.	2.1) La DBN no contiene capas. Se emite mensaje de advertencia.

Tabla 3.4 Caso de Uso CU09 - Pre-entrenar DBN

Caso de Uso CU08 - Entrenar RBM **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza por invocación del caso de uso CU09 - Pre-entrenar DBN. 2) Se ejecuta el algoritmo de entrenamiento para RBM, divergencia contrastiva o divergencia contrastiva persistente 3) Se almacenan los pesos de la RBM entrenada. 4) Se imprimen los estadísticos y resultados. 5) El caso de uso finaliza.	3.1) Ocurre fallo de comunicación sobre la GP-GPU por sobre-calentamiento. 3.2) Se informa el error. 3.3) El caso de uso finaliza.

Tabla 3.5 Caso de Uso CU08 - Entrenar RBM

Caso de Uso CU10 - Ajuste DBN **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza cuando el usuario ingresa datos etiquetados para la tarea de clasificación de la DBN. 2) Se crea el MLP con las capas para el ajuste fino. Se invoca al caso de uso CU11 - Crear MLP. 3) Por cada capa de la DBN se invoca el caso de uso CU15 - Entrenar MLP. 4) Se almacenan los pesos de las capas en la DBN. 5) El caso de uso finaliza.	3.1) La DBN no contiene capas. Se emite mensaje de error. 3.2) El caso de uso finaliza.

Tabla 3.6 Caso de Uso CU10 - Ajuste DBN

Caso de Uso CU11 - Crear MLP **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza por invocación del caso de uso CU10 - Ajuste DBN 2) Se inicializan los parámetros del MLP. 3) El caso de uso finaliza.	

Tabla 3.7 Caso de Uso CU11 - Crear MLP

Caso de Uso CU15 - Entrenar MLP **Actor: Usuario del Sistema** **Curso Normal**	**Curso Alternativo**
1) El caso de Uso comienza por invocación del caso de uso CU10 - Ajuste DBN. 2) Se ejecuta el algoritmo de entrenamiento para MLP, gradiente descendiente estocástico	
3) Se almacenan los pesos del MLP entrenado.	3.1) Ocurre fallo de comunicación sobre la GP-GPU por sobre-calentamiento. 3.2) Se informa el error. 3.3) El caso de uso finaliza.
4) Se imprimen los estadísticos y resultados. 5) El caso de uso finaliza.	

Tabla 3.8 Caso de Uso CU15 - Entrenar MLP

3.2.2. Diagrama de Clases

Un diagrama de clase describe un conjunto de clases y sus relaciones, siendo muy utilizados en el modelado de sistemas orientados a objetos [43]. Estos diagramas se usan para modelar la vista de diseño estática de un sistema, incluyendo el vocabulario del mismo. Son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables aplicando ingeniería directa e inversa.

En la figura 3.2 se observa el diagrama diseñado para la biblioteca. El mismo contiene solo las clases `RBM`, `DBN`, `MLP`, `Capa` (Abstracta), `CapaClasificacion`, `CapaRegresion`, `Funcion` (Abstracta), `FuncionSigmoidea`, `Unidad` (Abstracta), `UnidadBinaria`.

OpenBlas

También es posible unir a la implementación de Numpy y Theano, la librería de OpenBlas[2]. Esta genera mayores beneficios en las implementaciones en CPU. OpenBlas es una librería para operaciones multi-hilo en CPU, y como es de esperarse ésta aumenta la velocidad de los algoritmos brindando capacidades multi-hilo,cuando no se dispone de una GP-GPU.

3.2.3. Theano

Theano es una librería desarrollada en Python, brinda al usuario funcionalidades para definir, optimizar y evaluar expresiones matemáticas, en especial aquellas que involucran arreglos multi-dimensionales (*numpy.ndarrays*). Al utilizar Theano es posible aumentar la velocidad de ejecución de software desarrollado a comparación a aquel que solo fue desarrollado en lenguaje Python o `C`, y que los mismos involucren grandes volúmenes de datos para su manipulación. Además es capaz de sobrepasar en varios ordenes de velocidad a implementaciones puras en `C` [33].

Theano combina aspectos de sistemas de álgebra computacional (CAS por sus siglas en ingles) con aspectos de un compilador para la optimización. Además es capaz de generar código personalizado en lenguaje `C` para muchas operaciones matemáticas. La combinación de ambos aspectos es muy útil para

[2]Sitio oficial de OpenBlas: http://www.openblas.net/

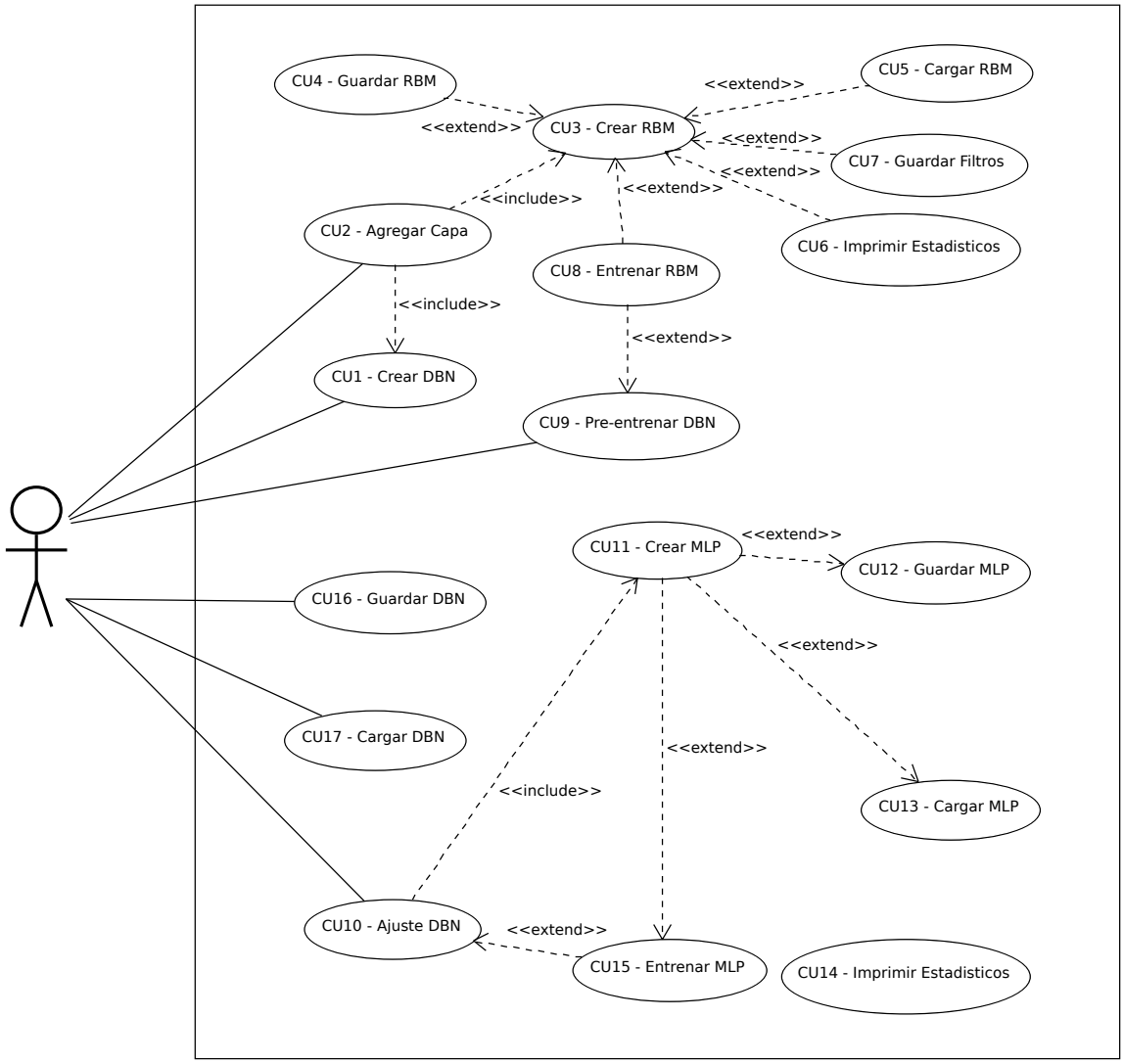


Fig. 3.1 Diagrama de Casos de Usos simplificado del sistema.

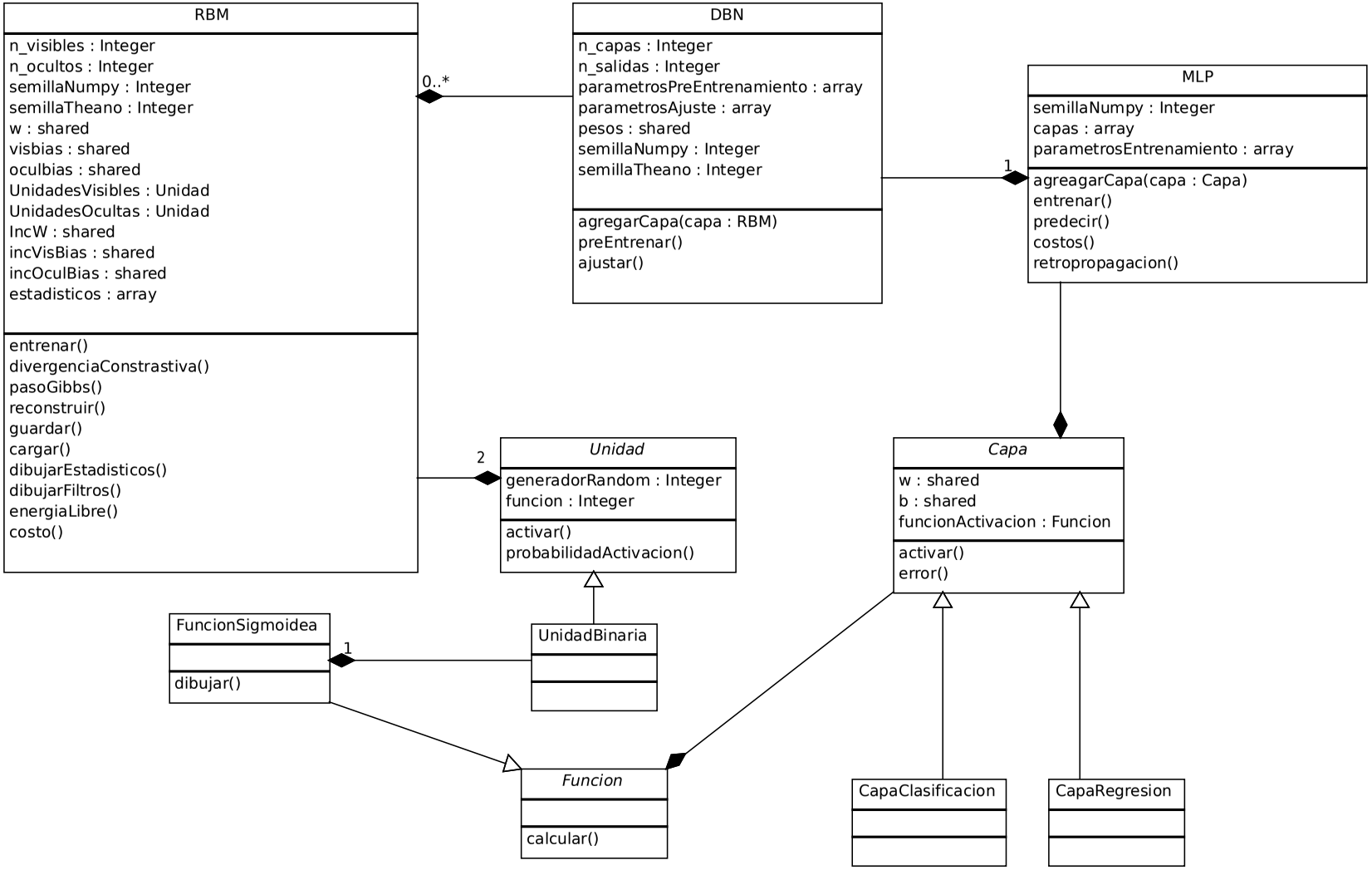


Fig. 3.2 Diagrama de Clases simplificado del sistema. Solo se muestran las clases más importantes.

expresiones matemáticas complicadas, donde las mismas son evaluadas de forma reiterada un gran número de veces, y donde la velocidad de evaluación de dichas expresiones es critica. En situaciones donde varias expresiones pueden ser agrupadas o simplificadas, Theano se encarga de reducirlas, lo que disminuye la cantidad de sobrecarga en la compilación/análisis.

La metodología de Theano es, expresiones matemáticas representarlas en grafos, y luego realizar optimizaciones según se requiera. Además de poder insertar código CUDA cuando se habilite el uso de la GP-GPU. Theano gestiona las optimizaciones según la complejidad de las expresiones (simbólicas), estas incluyen pero no están limitadas a:

- utilizar la GP-GPU para realizar cálculos.
- completado de constantes.
- combinar subgrafos similares, evitando cómputos redundantes.
- simplificación aritmética, por ejemplo: `x*y/x` → `y`; `--x` → `x`
- inserta operaciones eficientes del tipo BLAS cuando sea posible.
- utiliza operaciones 'inplace' para no interferir con otros cálculos.
- fusiona ciclos del tipo 'for-loop' para expresiones elemento a elemento.
- mejora la estabilidad numérica en operaciones, por ejemplo obtiene el resultado de `log(1+exp(x))` cuando `x` es muy pequeño.

Panorama general

Como cualquier librería para el cómputo, Theano tiene múltiples funcionalidades. Una de ellas es brindar soporte para la creación y manipulación de variables simbólicas, utilizadas en expresiones más generales en los grafos de expresiones. La diferenciación simbólica es otra característica muy útil. Un ejemplo de su aplicación es en el algoritmo de propagación hacia atrás en redes multicapas.

A continuación, se ejemplifica un algoritmo sencillo que multiplica dos matrices (A y B) y retorna su resultado en C. En este ejemplo puede observarse la mecánica de funcionamiento.

Algoritmo 3.1 Ejemplo Theano

```
1
from theano import tensor
3 from theano import function
import numpy.random
5
A = tensor.imatrix("A")
7 B = tensor.imatrix("B")
C = tensor.dot(A, B)
9 productoPunto = function(inputs=[A, B], outputs=C)

11 a = numpy.random.randint(0, 100, (1000,1000))
b = numpy.random.randint(0, 100, (1000,1000))
13 c = productoPunto(a, b)
```

Las primeras lineas se corresponden a los *imports* o llamadas de librerías externas, luego se definen las matrices A y B como tensores de segundo orden. Se asigna a C el resultado del producto punto entre A y B. Para realizar la operación deseada, se debe indicar a Theano que *nodos* tiene que incluir en el grafo de compilación, en este caso A, B y C, señalando cuales son las entradas y las salidas del mismo, en la instrucción *function* ejecuta la operación. Notar que hasta este punto no se asigno ningún valor a las variables, solo las definiciones.

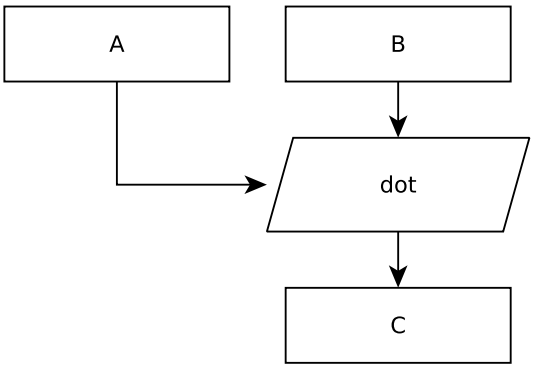


Fig. 3.3 Grafo producido por Theano sobre el Algoritmo 3.1. Las *cajas* definen las variables, los *paralelogramos* definen las funciones/operaciones.

Por último, se crean variables **no** simbólicas, `a` y `b` (1000x1000) tipo numpy.ndarray que almacenan valores enteros aleatorios. La variable `c` toma el valor que retorna `productoPunto` entre `a` y `b`.

En resumen, para cualquier algoritmo es necesario definir variables simbólicas, las mismas no se almacenan dato alguno pero que son utilizadas para constituir las operaciones deseadas. Luego se definen las variables *dato*, que son ingresadas en la función. Las entradas fluyen a través del grafo de compilación desde los nodos de entrada hacia la salida. La representación gráfica del proceso puede observarse en la figura 3.3.

El Algoritmo 3.1, se ejecuta sin problemas en la unidad central de procesamiento, si se quisiera utilizar el dispositivo es necesario indicar a Theano, por medio de variables de entorno globales, que la compilación del grafo debe incluir a la GP-GPU. En consecuencia, internamente los *datos* son transferidos de la memoria del huésped hacia la memoria del dispositivo, este último realiza las operaciones y luego se transfiere nuevamente los datos hacia la memoria de la CPU. Para poder lograr esto, basta con indicar que nodos deben ser transferidos, a su vez que tipo de variables y la cantidad de memoria requerida para ser reservada en la GP-GPU, aunque Theano puede lidiar con ello generalmente de forma automática sin intervención del usuario. Todo el proceso puede resumirse en el algoritmo 3.2.

Algoritmo 3.2 Ejemplo Theano GPU

```
1  ... #imports

3  a = numpy.random.randint(0, 100, (1000,1000))
   b = numpy.random.randint(0, 100, (1000,1000))
5
   A = tensor.shared(a, int32)
7  B = tensor.shared(b, int32)
   C = tensor.dot(A, B)
9  productoPunto = function(inputs=[], outputs=C)

11 c = productoPunto(a, b)
```

Como se observa, con respecto al algoritmo 3.1 solo se cambian las instrucciones de creación de variables simbólica por una '`shared`', lo que indica a Theano que dicha variable es compartida con la GP-GPU. Cabe destacar que el grafo generado es idéntico al anterior, como se presenta en la figura 3.4. Por cuestiones de claridad se agregaron las operaciones `transferir`, estas indican que los

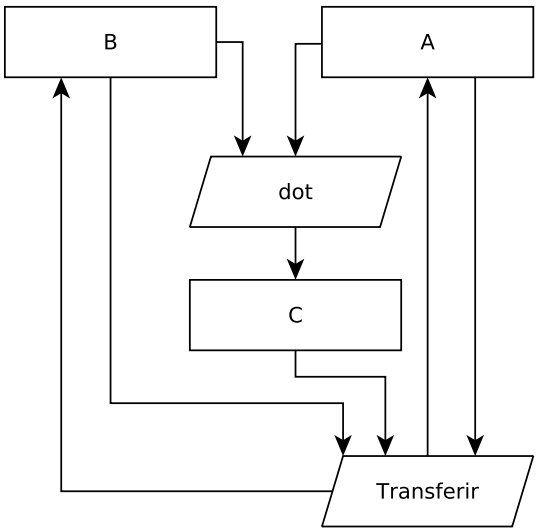


Fig. 3.4 Grafo producido por Theano sobre el Algoritmo 3.2, se han agregado las funciones de transferencia de memoria para este diagrama especifico.

datos deben moverse desde la memoria central a la memoria del dispositivo y vise-versa. El resultado, `c` de retornado como numpy.ndarray, por lo que también es necesario transferirlo hacia la CPU.

Además de los módulos `tensor`, `shared` y `function` existen otros a tener en cuenta, como ser el módulo `scan`, que provee las directrices para realizar ciclos o bucles del tipo *for-loop* de forma eficiente en GP-GPU.

La librería Theano brinda una gran libertad a la hora de diseñar los algoritmos. El diseño de los mismos para ambas arquitecturas es casi idéntico. Solo basta con cambiar la configuración en Theano, su compilador de funciones, para que la implementación que se genera en tiempo de ejecución, utilice unicamente en la CPU o bien, en GP-GPU. Esto realiza los cambios necesarios en los algoritmos, reemplazando código Python por código CUDA cuando fuese posible, liberando al programador de configuraciones, diseño de algoritmos en CUDA, etc.

Un ejemplo de ello puede verse en el siguiente script, `theanoTest.py` el mismo puede descargarse desde la pagina oficial. Se observan los tiempos de ejecución para ambas arquitecturas,

Algoritmo 3.3 Ejemplo Theano GPU/CPU

```
root@2root:~/cupydle# python3 cupydle/test/theano_test1.py

Using gpu device 0: Tesla C1060 (CNMeM is disabled, cuDNN not available)
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>),
HostFromGpu(GpuElemwise{exp,no_inplace}.0)]
Looping 1000 times took 1.038315 seconds
Result is [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
  1.62323296]
Used the gpu

root@2root:~/cupydle# vi ../.theanorc
... cambiar la variable 'gpu' por 'cpu'

root@root:~/cupydle# python3 cupydle/test/theano_test1.py
[Elemwise{exp,no_inplace}(<TensorType(float32, vector)>)]
Looping 1000 times took 4.390988 seconds
```

```
Result is [ 1.23178029  1.61879337  1.52278066 ...,  2.20771813  2.29967761
  1.62323284]
Used the cpu
```

3.2.4. Grafo muestreo de Gibbs: compilación Theano

Uno de los algoritmos que requiere la mayor cantidad de recursos, es el "simple" muestreo de Gibbs, es utilizado para el entrenamiento de las RBMs (ver algoritmo 2.1). Cuando el volumen de datos involucrados es grande, el mismo itera reiteradas veces sobre los mismos datos realizando sucesivas operaciones de multiplicación, adición y muestreo.

Es de gran utilidad realizar un grafo de compilación, figura 3.5, que luego el mismo se implementará con ayuda de la función *scan* de Theano. Esta función simplifica y optimiza ciclos iterativos del tipo `for-loop` junto al muestreo de Gibss en el proceso.

3.3. Implementación

A la hora de implementar los algoritmos se deben contemplar cuestiones relacionadas a los cálculos, si es conveniente distribuir el algoritmo de entrenamiento del modelo o distribuir las operaciones y cálculos del mismo. Para el primer enfoque, en general lo que se realiza es ejecutar el mismo algoritmo de entrenamiento en distintos nodos (CPUs) y luego con una operación reducción se promedian los resultados, este último se utiliza para actualizar los parámetros de la red. Esta metodología es generalmente aplicada en arquitecturas distribuidas, donde la comunicación es por medio de mensajes (clusters de CPUs con MPI[3]).

Para el segundo enfoque, se realizan las operaciones algebraicas aprovechando las funcionalidades del hardware con el cual se cuenta, ya sea CPU o GP-GPU. Operaciones del tipo BLAS[4] pueden realizarse en ambas arquitecturas, aunque en GP-GPU se observa un gran incremento en cuanto a velocidad [34, 5]. Debido a los algoritmos a desarrollar y a la especificaciones del proyecto se optó realizar los cálculos utilizando la GP-GPU el segundo enfoque mencionado.

3.3.1. Hardware: CUDA

En esta sección se abordará sucintamente los detalles de la arquitectura de hardware sobre la cual se realizará la implementación. Comprender las *ventajas* como así sus *limitaciones* sirven de apoyo para las decisiones de diseño de los algoritmos.

La GP-GPU puede ser programada utilizando Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo) o simplemente CUDA. Esta fue introducida en el mercado a inicios de 2007 por la empresa NVIDIA. La misma hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo que permiten a los programadores usar una extensión del lenguaje de programación C para codificar algoritmos en GP-GPU. Fue concebida para el cálculo intensivo (principalmente porque libera a la CPU de la tarea de la renderización de los objetos en la

[3]Interfaz de Paso de Mensajes, o en ingles Message Passing Interface (MPI) conforman una biblioteca de funciones para la explotación de múltiples procesadores. Una implementación de la misma puede encontrarse en https://www.open-mpi.org/

[4]BLAS: Basic Linear Algebra Subprograms. Es un conjunto de rutinas a bajo nivel para realizar operaciones algebraicas comunes como ser la adición de vectores, multiplicación escalar, producto punto, combinación lineal y la multiplicación matricial. Para más información visite la pagina oficial: http://www.netlib.org/blas/

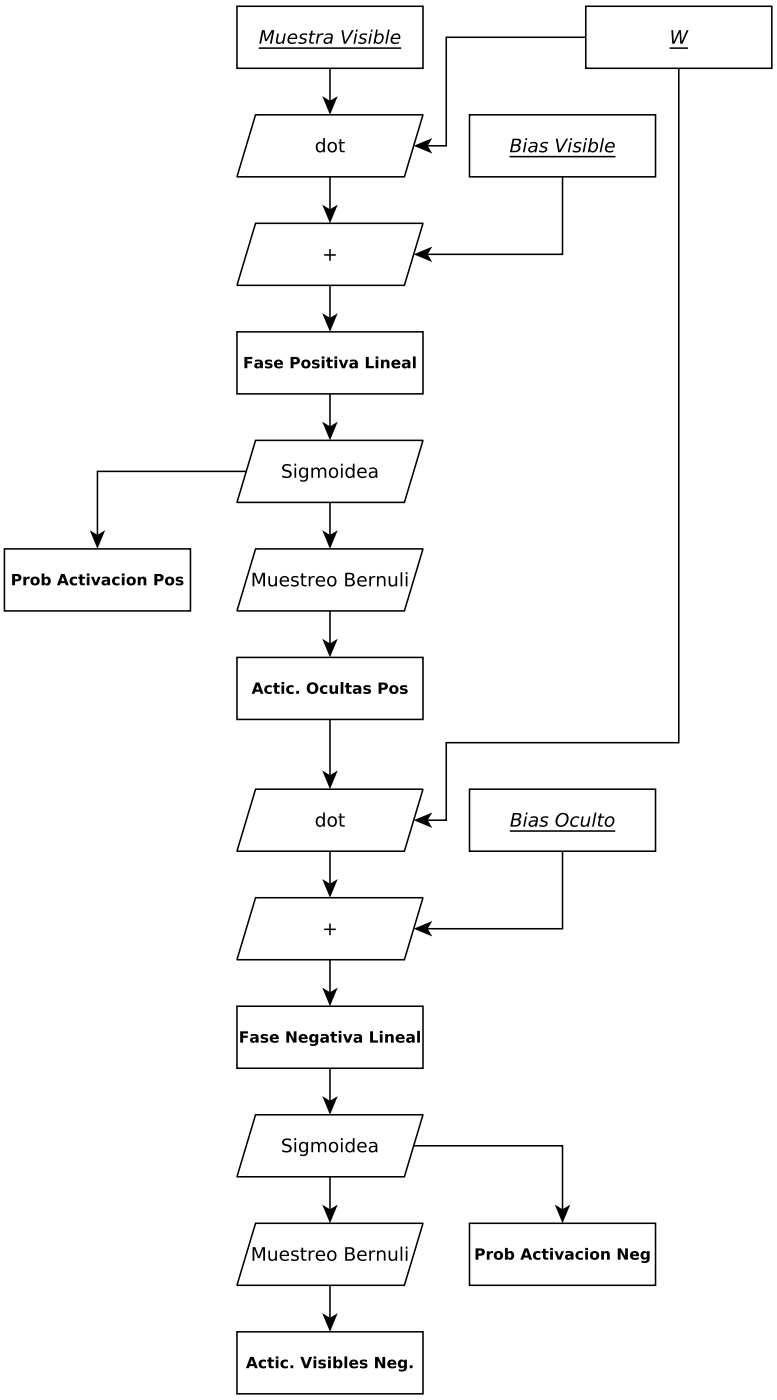


Fig. 3.5 Grafo de compilación Theano. esquematiza la secuencia de un paso de Gibbs para el proceso de muestreo. El grafo se aplica a la función SCAN de Theano. Las *entradas* se encuentran en *itálicas* y subrayado, los **retornos** en **negritas**

Especificaciones de Arquitectura	**Capacidad de Cómputo**												
	Tesla				Fermi		Kepler			Maxwell		Pascal	
	1.0	**1.1**	**1.2**	**1.3**	**2.0**	**2.1**	**3.0**	**3.5**	**3.7**	**5.0**	**5.2**	**6.0**	**6.1**
Cant. ALUs para operaciones aritméticas de enteros, precisión simple y de punto flotante.	8				32	48	192			128		64	128
Núm. de funciones de precisión simple y de punto flotante por unidad de funciones trascendentales.	2				4	8	32					16	32
Núm. de unidades filtro de texturas por cada direccionamiento de unidades de renderización.	2				4	8	16			8			
Cantidad de planificadores *wrap*.	1				2		4					2	4

Tabla 3.9 Mayores diferencias entre las Capacidades de Cómputo. Los nombres de los dispositivos sobre las mismas definen el *codename* del dispositivo en CUDA.

pantalla) y observar la creciente curva de rendimiento obtenida a los largo de los últimos años, se comenzó a utilizar para cálculos de propósito general.

Internamente la GP-GPU implementa un enorme número de unidades aritmético lógicas (UAL), estas realizan operaciones elementales, mientras que presenta un escaso número de unidades de control de flujo, lo contrario que sucede en una CPU convencional. Como resultado, la GP-GPU posee un mayor número de características para procesamiento masivo en comparación a una CPU.

Las funcionalidades del dispositivo se encuentran unificadas en las versiones de *arquitectura de hardware GP-GPU* y la *versión de CUDA*. En general, estas se pueden agrupar en las llamadas "*capacidades de cómputo*", para más detalle ver la tabla 3.9.

Elementos de una arquitectura CUDA

En la jerga de GP-GPU, el hardware gráfico se identifica como *dispositivo*, mientras que la CPU que la soporta como *huésped*. Un huésped puede soportar varios dispositivos (en principio tanto como puertos PCI-Express se tengan). Cada dispositivo implementa básicamente 4 tipos de componentes: los *streaming multiprocessors* (SM), las *unidades de funciones especiales* (SFU), las de *doble precisión* y el *caché*. Sin entrar en detalles, los SM presentan un conjunto de ocho procesadores escalares denominados SP (*scalar processors*) capaces de procesar evaluandos de precisión simple. Las SFU implementan esquemas de evaluación optimizados. Las unidades de doble precisión realizan operaciones con evaluandos de doble precisión. Finalmente el caché realiza operaciones análogas presentes en un sistema huésped, en pocas palabras preserva un conjunto de datos e instrucciones (siguiendo cierta lógica) cuya lectura logra un mayor rendimiento que en otro tipo de memorias [37, 8].

Modelo de organización y ejecución

Una función es un conjunto de transformaciones en una secuencia establecida, organizada en entradas, operaciones y salidas, aquellas a utilizarse en un dispositivo se denominan *kernel* o *núcleo*. Para su procesamiento CUDA conforma lo que se denomina grilla o *grid*. La grilla presenta en cada nodo un bloque de hilos o *threads*. Un hilo se define como la mínima unidad de ejecución posible, éstos

pueden agruparse en un conjunto de hilos. El dispositivo gestiona varios niveles de subdivisiones donde distribuye dichos bloques de hilos sobre los SM para ser procesados. El orden de reparto es desconocido. Un ejemplo puede observarse en la figura 3.6.

Existe al menos una unidad dentro del dispositivo encargada de la organización de los bloques de hilos dentro de cada SM, denominada SIMT. La misma tiene la responsabilidad siempre que pueda de mantener a los SP activos. Para ello cada bloque de hilos es subdividido en *warps*, que no es mas que un conjunto de 32 hilos, cada uno puede ser identificado por un índice.

Así la unidad SIMT desglosa éstos bloques de hilos y distribuye warps sobre los SP. La cantidad de warps activos por SM es dependiente de la Capacidad de Cómputo del dispositivo. A su vez la unidad administra múltiples colas de warps (en espera, listo para ejecución, entre otras).

Por otro lado tenemos el intercambio de procesos en CPU es una tarea costosa, dado que requiere almacenar el estado del programa para que en el próximo quantum de tiempo de procesador que le sea asignado pueda conocer el punto desde donde debe continuar (pila de instrucciones) y los datos que tenía disponible (así también los recursos a los que poseía acceso); contrariamente en una GP-GPU los recursos son asignados por hilo, por lo cual no hay intercambios de estados entre ellos. Así el intercambio de hilos se hace sin una carga adicional de rendimiento. Como resultado, se puede reducir notablemente la latencia de operaciones, en comparación a arquitecturas tradicionales como la PC.

El modelo de ejecución de la arquitectura CUDA se basa en lo que se conoce como SIMT *single instruction multiple thread*, es decir, que todos los hilos dentro de un bloque ejecuten la misma instrucción cada uno a sus respectivos datos, y de esta forma se amortiza la búsqueda de la instrucción en memoria.

Tanto el sistema huésped como el dispositivo cuentan con sus distintivas memorias físicas. Cada memoria se encuentra separada por un bus de PCI-Express, por el mismo puede llevarse a cabo la transferencia de los datos entre las dos arquitecturas, figura 3.7. En general, este proceso es uno de los más críticos, la velocidad de transferencia está limitada físicamente, la misma no depende de la velocidad de procesamiento de los núcleos sino del Bus de datos. Según como se manipulan de los datos y su transferencia hacen que el software final se ejecute con mayor o menor velocidad.

A continuación se detallan los pasos principales de ejecución utilizando CUDA en una GP-GPU.

Proceso de ejecución general

Un proceso maestro inicia la operación en la CPU, el cual realiza los siguientes pasos:

1. Inicializa la GP-GPU.
2. Reserva memoria en la unidad huésped y el dispositivo.
3. Copia los datos desde el huésped hacia la memoria del dispositivo.
4. Lanza múltiple instancias de ejecución de los kernels en el dispositivo en caso de ser posible.
5. Los datos se copian desde el dispositivo hacia la memoria del huésped.
6. En caso de ser necesarios los pasos 3-5 son repetidos.
7. Se libera la memoria y terminan los procesos.

A más bajo nivel, en la GP-GPU se ejecutan los siguientes procesos

- Cada instancia de kernel se ejecutan en un SM.
- Si el número de instancias exceden el numero de SMs disponibles, se ponen en cola de espera hasta que se liberen recursos y ejecuta luego.

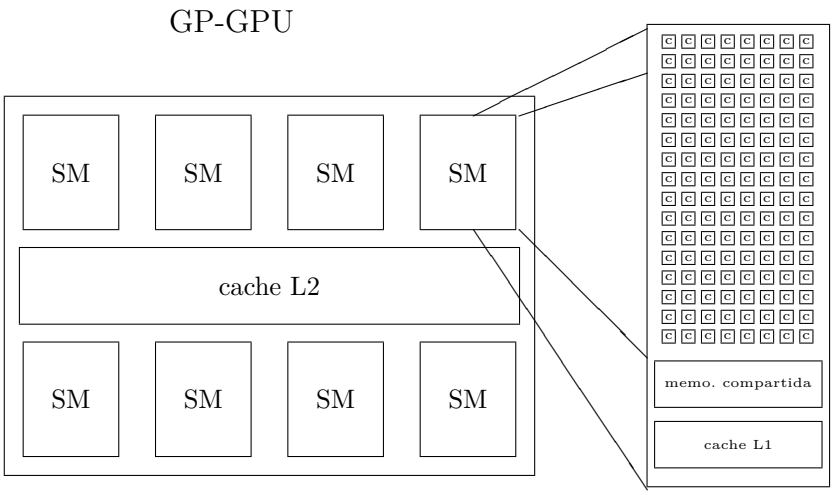


Fig. 3.6 Descomposición del dominio de resolución de los bloques de hilos, con cierta cantidad de núcleos CUDA (c) y posterior asignación a los SM. En este ejemplo se cuenta con ocho SM. Cada SM cuenta con su propia memoria interna.

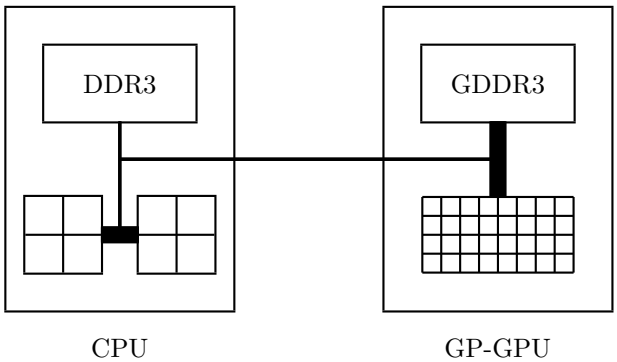


Fig. 3.7 *Izquierda* CPU, dos núcleos los cuales ejecuta algunos hilos por núcleo, memoria física DDR3. *Derecha* GP-GPU, por bloque ejecuta mayor cantidad de hilos, memoria GDDR3. La conexión entre ambas unidades es PCI-Express.

- Todos los hilos entre una instancia tienen acceso a la memoria local pero no a las otras instancias (incluso si están en el mismo SM).
- No hay garantía alguna del orden de ejecución de las instancias.

3.3.2. Despliegue

Surgieron inconvenientes a la hora de proceder a la instalación de las tecnologías básicas en la PC, único recurso disponible e indispensable para este proyecto. Tecnologías como ser Python en su tercera versión, Numpy, Scipy ecosystem y Theano. Tampoco pudieron ser satisfechas las dependencias para la instalación del *driver Nvidia* (versión 340.29) y *caja de herramientas* de CUDA. En general, debido a la incompatibilidad entre paquetes ya instalados en la PC. En adición a lo anterior, no era factible la modificación de la configuración del huésped, ya que es un bien de uso público junto a otras personas que desempeñan su labor en el ámbito académico de la facultad. Se tuvo que recurrir a la instalación por otros medios no convencionales.

Para efectuar la tarea, se eligió métodos o técnicas de *virtualización*, más precisamente el despliegue de contenedores utilizando el aplicativo *docker*[5]. Este proceso no fue planificado en el cronograma, y no forma parte de los requerimientos originales del proyecto. La implementación de docker, trajo consigo beneficios extras como generalización y automatización de la instalación tanto de los requerimientos internos como externos de forma sencilla.

Para el despliegue de contenedores es recomendable crear un archivo `Dockerfile`, en el cual se especifican todos los detalles del sistema crear, como así la asignación de recursos e imposición de restricciones. A continuación se desarrolla una breve descripción de Docker junto a los pasos de instalación en la PC disponible para el proyecto.

Docker

Docker es un proyecto de código abierto que automatiza el despliegue[6] de aplicaciones dentro de *contenedores* de software, proporcionando una capa adicional de abstracción y automatización de virtualización del sistema operativo. En el sistema operativo Linux, docker utiliza características de aislamiento de recursos del kernel, tales como cgroups y espacios de nombres (namespaces) para permitir que contenedores independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener la ejecución como en el caso de máquinas virtuales.

Mediante el uso de contenedores, los recursos pueden ser aislados, los servicios restringidos, y se otorga a los procesos la capacidad de tener una visión casi completamente privada del sistema operativo con su propio identificador de espacio de nombres para el proceso, la estructura del sistema de archivos, y las interfaces de red. Contenedores múltiples comparten el mismo núcleo, pero cada contenedor puede ser restringido a utilizar sólo una cantidad definida de recursos como CPU, memoria y recursos de E/S.

Gracias a docker, el entorno se vuelve independiente y libre de conflictos con otros paquetes. Para lograr el despliegue es necesario ejecutar una serie de pasos para instalar los requerimientos necesarios en docker. Todos los pasos pueden estar contenidos en un archivo, llamado `Dockerfile`, para facilitar el proceso. Los dos comandos a ejecutar para disponer del entorno completamente funcionando son:

[5] Sitio oficial de Docker: https://www.docker.com/
[6] Despliegue: Proceso por el que una aplicación informática pasa a estar lista para utilizarse.

```
admin@gpu:~$ docker build -t cupydleOS:latest .
#... creando la imagen

admin@gpu:~$ sudo docker run --name cupydle -it --privileged=true \
--device /dev/nvidiactl:/dev/nvidiactl \
--device /dev/nvidia-uvm:/dev/nvidia-uvm \
--device /dev/nvidia0:/dev/nvidia cupydleOS:latest /bin/bash
# ejecutando de forma iterativa
```

Para más información se puede acceder a la documentación oficial de docker[7]. A continuación se detalla el `Dockerfile` utilizado en el proyecto.

Algoritmo 3.4 Dockerfile

```
1
FROM ubuntu:14.04
MAINTAINER Ponzoni Nelson <npcuadra@gmail.com>
LABEL ubuntu="14.04" python="3" nvidia="340.29" cuda="6.5" theano="0.82"

# Main workdir
WORKDIR /root/

# Kernel, add repo for downloading
RUN echo "deb http://cz.archive.ubuntu.com/ubuntu precise-updates main" >>
/etc/apt/sources.list

# Install packages and dependencies
RUN apt-get update && \
    apt-get install -y --no-install-recommends --force-yes \
    wget \
    build-essential \
    python3-numpy \
    python3-scipy \
    python3-dev \
    python3-pip \
    python3-nose \
    python3-matplotlib \
    libopenblas-dev \
    git \
    gcc-4.6 \
    g++-4.6 \
    linux-headers-3.2.0-97-generic # linux-headers-`uname -r`

# Install theano with python3
RUN pip3 install Theano

# Change the default gcc
RUN update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.6 20 && \
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 10 && \
    update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.6 20 && \
    update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 10 && \
    update-alternatives --install /usr/bin/cc cc /usr/bin/gcc 30 && \
    update-alternatives --set cc /usr/bin/gcc && \
    update-alternatives --install /usr/bin/c++ c++ /usr/bin/g++ 30 && \
    update-alternatives --set c++ /usr/bin/g++

# Setting up .theanorc for CUDA
RUN echo -e
"[global]\ndevice=gpu\nfloatX=float32\noptimizer_including=cudnn\n[lib]\ncnmem=1\n[nvcc
]\nfastmath=True\nflags=-D_FORCE_INLINES\n"
```

[7]Pagina Oficial Docker: https://www.docker.com/what-docker

```
    >> /root/.theanorc
47
   # Download driver, toolkit and samples
49 RUN wget --progress=bar:force
   http://developer.download.nvidia.com/compute/cuda/6_5/rel/installers/cuda_6.5.14
       _linux_64.run
51  && \
       chmod a+x cuda_6.5.14_linux_64.run
53
   # Install them all
55 RUN echo "Installing the driver" && ./cuda_6.5.14_linux_64.run -silent --driver
   --override && \
57     echo "Installing the tooltkit" && ./cuda_6.5.14_linux_64.run -silent
       --toolkit
59     --override && \
       echo "Intalling samples" && ./cuda_6.5.14_linux_64.run -silent --samples
61     --samplespath=/home/ --override && \
       rm cuda_6.5.14_linux_64.run && \
63     ln -s cuda-6.5 /usr/local/cuda && \
       rm -rf /var/lib/apt/lists/*
65
   # Setting up environment variables
67 ENV PATH /usr/local/cuda/bin:${PATH}
   ENV LD_LIBRARY_PATH /usr/local/cuda/lib64:${LD_LIBRARY_PATH}
69 ENV PYTHONPATH /root/cupydle/
```

3.3.3. La biblioteca: Cupydle

El software, resultado de este proyecto recibe el nombre de **CUPYDLE**, siglas provenientes de ***CU**DA **PY**thon **D**eep **LE**arning*; desarrollado en lenguaje Python, implementa las redes de creencia profunda y brinda la posibilidad de despliegue tanto en CPU únicamente o bien en GP-GPU.

Para que la biblioteca desarrollada sea considera como tal, se deben proveer de un conjunto de funcionalidades y características que brinden un soporte de forma integral al usuario en diferentes aspectos. Por ejemplo no basta con solo proveer de los algoritmos para el entrenamiento de las DBNs sino que también sería de gran utilidad disponer de métodos para el almacenamiento del estado actual de modelo, graficar resultados, preprocesamiento y manipulación de datos, etc. Todas ellas pueden ser invocadas por el usuario en una capa o nivel superior del software, evitando la codificación de las mismas.

También se invirtió esfuerzo en optimizar el código desde el enfoque de utilización de memoria. Se focalizó en la reserva de memoria en GP-GPU de manera contigua, para diminuir la fragmentación. Por otro lado, se realizaron pruebas para evaluar y optimizar el uso de la memoria Ram, en conjunto lograr así la mejor utilización de los recursos disponibles.

Algunas de las funcionalidades finales con la que se cuentan hasta el momento son:

- *Generales*
  - *MacroBatch*, copia por chunks hacia la GP-GPU de forma automática e inteligente.
  - Métodos óptimos para la persistencia en disco, optimizada la reserva de memoria.
  - Gráficos generales. Resultados de entrenamiento. Filtros de las unidades ocultas. Resultados del ajuste fino. Matriz de confusión.
- *Máquinas de Boltzmann Restringidas (RBM)*
  - Tipos: Binaria y Gaussiana.

- Almacenamiento y recuperación de las representaciones ocultas.
- Entrenamiento:
  - Algoritmo de Divergencia Contrastiva con n pasos de Gibbs (CD_n)
  - Algoritmo de Divergencia Contrastiva Persistente con n pasos de Gibbs (PCD_n)
  - Aprendizaje por batchs, termino de momento.
  - Función de costo: Error cuadrático medio (MSE), entropia cruzada, energía libre.
- Evaluación:
  - Muestreo desde el modelo con n pasos.

- *Redes de Creencia Profunda (DBN)*
  - Entrenamiento:
    - No-supervisado: Apilado de RBMs por capas. Entrenamiento heurístico.
    - Ajuste de los pesos por medio del algoritmo de propagación hacia atrás.
  - Evaluación: predicción de patrón.
- *Clasificador estándar multicapa (MLP)*
  - Algoritmo de optimización: Gradiente descendiente estocástico (SGD), aprendizaje por batchs y termino de momento.
  - Numero variable de capas: capa logística, capa softmax.
  - Funciones de activación: sigmoidea, tanh, ReLu, gaussiana.
  - Funciones de costo: entropia cruzada, error cuadrático medio (MSE).
  - Términos de regularización l_1 y l_2.
  - Técnicas de parada temprana y evaluación: Paciencia, iteraciones, tolerancia al error, tiempo transcurrido, evaluación rápida en históricos.
- *Validación y prueba de los modelos, manipulación de datos*
  - Validación Cruzada.
  - Segmentación: K_fold, LabelKFold, LeaveOneOut, StratifiedKFold,
  - Normalización: z_score, Blanqueo, Rango [min,max].
- *Búsqueda de hiperparámetros.*
  - Búsqueda por grilla exhaustiva y búsqueda por grilla aleatoria.

Cabe destacar que se han desarrollado clases y métodos abstractos para disversas funcionalidades. Por ejemplo, para los tipos de RBM se codifico una clases `Unidad` abstracta, si un usuario desea crear un nuevo tipo de unidad (ejemplo: softmax) ésta deberá heredar de 'unidad', por lo que se obliga a implementar todos sus métodos (compatibilidad y consistencia con el resto del software).

Requerimientos

Los requerimientos de software ajeno a la biblioteca necesarios para la ejecución de la misma se detallan en la tabla 3.10. El primer requerimiento es simplemente el lenguaje de programación, Python, los siguientes dos son librerías para el cómputo científico, cálculo vectorial. En la cuarta posición, Theano, librería de cómputo masivo y distribuido en GP-GPU. La quinta es requerida para realizar gráficos.

Librería	Versión
Python	>= 3.0
Numpy	>= 1.8
Scipy	>= 0.11
Theano	>= 0.82
Matplotlib	>= 1.3
OpenBlas*	>= 1.13
h5py*	>=2.5

Tabla 3.10 Requerimientos de la biblioteca CUPYDLE. * Recurso opcional.

La sexta es utilizada para operaciones multi-hilo en CPU. Por último, h5py brinda la funcionalidad de almacenamiento en disco de manera eficiente. Estas dos últimas son opcionales.

Documentación

La documentación del software no pertenece a los objetivos originales del proyecto, aunque como se aclaró en el apéndice A, cualquier software sin documentación clara y coherente limita su vida útil (ver tabla A.1). La documentación del código fuente posibilita la continuidad de futuros desarrollos de manera más sencilla y productiva. Gracias a buenas prácticas de programación, como ser la documentación *in situ* de los métodos, clases y funciones se pudo generar la documentación de la biblioteca sin grandes complicaciones. La misma se encuentra disponible de forma online[8]. En la pagina web se puede acceder a la versión PDF. Actualmente se encuentra en desarrollo y se encuentra una copia de la misma en el apéndice C. También son accesibles online los esquemas de diseño, el código fuente, etc., como se observa en las capturas de pantalla 3.8.

Ejemplo de uso

Utilizar las funcionalidad de la biblioteca es muy sencillo, basándose en técnicas y metodologías de otras bibliotecas populares se sintetizan las siguientes funciones: «`DBN`» crea una red de creancia profunda, «`agregarCapa`» agrega una nueva capa a la red con los sus parámetros (tasa de aprendizaje, tasa de dropout, cantidad de pasos de Gibbs, etc), el siguiente método es «`entrenar`» como su nombre lo indica realiza el entrenamiento no supervisado del modelo, el método «`ajustar`» efectua el ajuste fino del modelo con los parámetros análogos a un clasificador estándar (tasa de aprendizaje, momento, épocas, etc), por último se cuenta con las funciones de «`guardar`» y «`cargar`» necesarias para persistir y recuperar los modelos en disco. En el algoritmo 3.5 ejemplifica la secuencia antes mencionada.

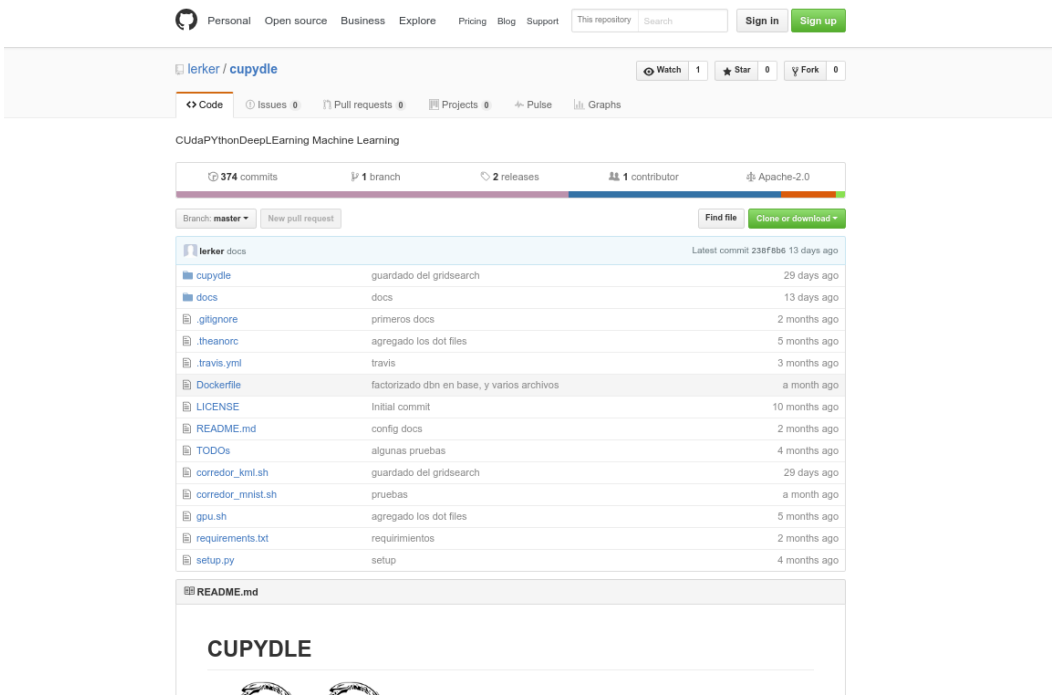
Algoritmo 3.5 Ejemplo oprativo de cupydle

```
1
from cupydle.dnn.dbn import
3 import numpy as np

5 misDatos = [...]

7 # crea la DBN
miDBN = DBN(nombre='midbn')
9
# agrega una capa a la DBN de 784 unidades ed entrada y 100 unidades ocultas,
```

[8]https.//www.http://cupydle.readthedocs.io

```
# tamanio del batch, tasa de aprendizaje para los pesos, tipo de red.
miDBN.agregarCapa(n_visible=784, n_hidden=500, epocas=100, tamMiniBatch=10,
lr_pesos=0.1, pasosGibbs=1, momento=0.01, tipo='binaria')

# entrena la red con el conjunto de datos, tipo de algoritmo, imprime filtros..
miDBN.entrenar(dataTrn=datosEntrenamiento, pcd=False,
guardarPesosIniciales=False, filtros=True)

# ajusta la red con los parametros para la tarea de clasificacion
miDBN.ajuste(datos=datosmisDatos, fnActivacion="sigmoidea")

# persiste en disco el modelo entrenado
miDBN.guardar('miModelo')

# recupera desde disco el modelo almacenado
miDBN = DBN.cargar('miModelo)


#fin
```

(**a**)



(**b**)

Fig. 3.8 Documentación y código fuente de la biblioteca Cupydle. (**a**) Captura de pantalla del sitio *readthedocs* (https.//www.http://cupydle.readthedocs.io) donde se encuentra alojada la documentación. Disponible también la descarga en archivo PDF. (**b**) Repositorio *git* del código fuente (https://github.com/lerker/cupydle)

Capítulo 4

Experimentos y Resultados

Las pruebas fueron realizadas en una computadora con una GP-GPU instalada, disponible como recurso para este proyecto. Las características del dispositivo se detalla en la tabla 4.1. Información más detallada sobre la GPU puede encontrase en [11]. La computadora donde se encuentra instalada la unidad posee un procesador Quad Core 2 Duo, tal como se detalla junto a otras características propias en la tabla 4.2.

En las siguientes secciones se mostrará el desempeño de las redes de creencia profundas frente a experimentos clásicos del aprendizaje maquinal. Es común en el ámbito académico-científico realizar pruebas comparativas con la base de datos de dígitos manuscritos llamada MNIST [27]. Por otro lado, se realizaron pruebas sobre el conjunto de datos multimedia de registros con emociones humanas, llamada *RML emotion Database*.

4.1. Consideraciones previas

Para todas las pruebas los conjuntos de datos se ha fraccionado y evaluados en lo posible reiteradas veces para obtener estimadores con mayor confianza utilizando técnicas de validación cruzada. Se generaron particiones de entrenamiento y prueba, diez para MNIST y seis para RML. Se dividió el conjunto de entrenamiento en validación y entrenamiento propiamente, asignando un 20 % de los patrones de forma aleatoria al primero y el restante 80 % al segundo conjunto. Dado que los datos están balanceados, es decir, se presentan misma cantidad de datos etiquetados de cada clases en cada conjunto, no es necesario un balanceo artificial para elegir las particiones.

Se propone realizar un análisis comparativo sobre el desempeño del sistema en su versión iterativa utilizando CPU y masiva con GP-GPU. Debido a los tiempos requeridos para el entrenamiento de una DBN son extensos, demostrados en los trabajos científicos y como el mismo Hinton declara en sus investigaciones [22, 18] deben limitarse las pruebas realizadas para que las mismas sean viables. Dos límites podemos mencionar, la cantidad de épocas a entrenar y por otra parte, reducir el tamaño de la base de datos conjuntos de pruebas. Si bien el primer enfoque no es aconsejable aplicarlo (reducir épocas de entrenamiento puede que el modelo no consiga aprender lo suficiente sobre los datos) se opto por esta metodología ya que cubre con los objetivos.

Especificaciones	Valor
Capacidad de Cómputo	1.3
Multiprocesadores / Núcleos CUDAxMP	30 / 8
N° de núcleos de procesamiento para streaming	240
Frecuencia de los núcleos de procesamiento	11296 MHz (1.3 Ghz)
Memoria global dedicada	4294770688 (4GiB)
Frecuencia de la memoria	800 MHz
Interfaz de memoria	512-bit GDDR3
Ancho de banda de memoria	102 GB/sec
Interfaz del sistema	PPCI Express x16 Generación 2
Memoria Total constante	65536 (64kiB)
Memoria compartida por MP	16384 (16kiB)
Registros por MP	16384 (16kiB)
Hilos por warp	32
Cantidad máxima de hilos por bloque	512
Máxima dimensión de hilos	(512,512,64)
Máxima dimensión de la grilla	(65535,65535,1)

Tabla 4.1 Especificaciones de la GP-GPU Nvidia® Tesla C1060

Especificaciones	Valor
Procesador	Intel® Core™ 2 Quad Q9300
Número de núcleos (hilos)	4 (1)
Velocidad del reloj	2.50 GHz
Cache	3 MB
Tipo Memoria	2x DDR3-800 MHz
Memoria del sistema	4037584 kB ≈ 4 GB
Sistema Operativo	Ubuntu Linux 12.04.5 LTS
Kernel	GNU/Linux 3.2.0-97-generic x86_64

Tabla 4.2 Especificaciones de la CPU disponible para el proyecto.

Para todas las pruebas, se han considerado momentos en que la carga de la CPU se encontrase en su mínimo ($\sim 0.0\,\%$); además se ha reducido toda actividad en la mismas para obtener estimadores de performance con mayor precisión.

Cabe destacar que el hardware utilizado al momento de redacción de este informe es antiguo (tanto la CPU como la GP-GPU datan ∼2008/09). En la actualidad se han realizado mejoras significativas en ambas arquitecturas, por lo cual lo resultados expuestos en esta sección pueden variar pero aun así mantener la relación entre si. Por otro lado, no se tuvo en cuenta la paralelización de los algoritmos en CPU, lo cual influye en los tiempos necesarios en las pruebas iterativas.

4.2. MNIST: clasificación de dígitos manuscritos

La base de datos del MNIST contiene 60.000 imágenes de entrenamiento y 10.000 imágenes de test de dígitos escritos a mano del 0 al 9 por alrededor de 250 autores distintos [26]. Las imágenes han sido previamente normalizados en tamaño y centrados en un cuadrado de 28 píxeles de alto por 28 píxeles de ancho, por lo tanto se tienen un total de 784 valores/datos por imagen. La intensidad de cada uno de los píxeles también ha sido normalizada de forma que se sitúe entre 0 y 255. Para utilizarlos en un modelo, únicamente se debe normalizar los valores de los dígitos dentro del rango

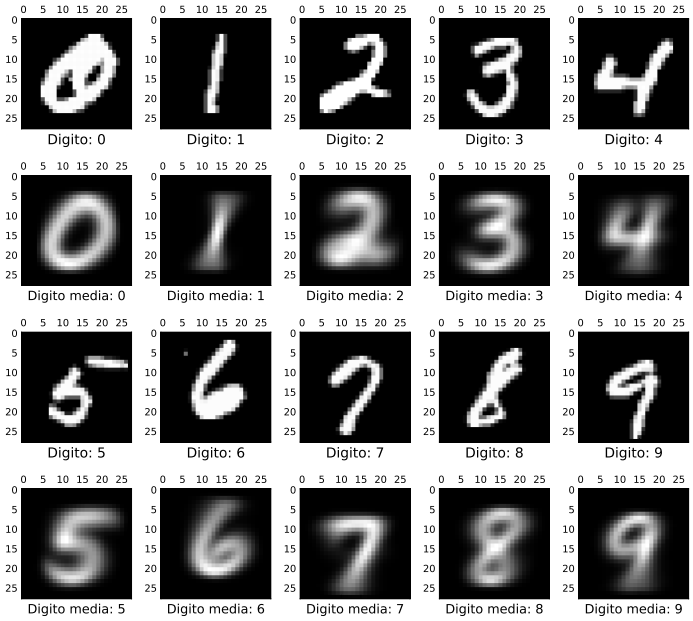


Fig. 4.1 Ejemplo de la base de datos MNIST, dígitos manuscritos del $0, \ldots, 9$. Imágenes de 28×28 píxeles. En la primer y tercer fila de imágenes cada clase seleccionadas de forma aleatoria del conjunto. En la segunda y cuarta fila, la media de las imágenes para su clase correspondiente.

[0,1] o alguna otra normalización deseada. En la figura 4.1 podemos observar una muestra de cada clase del conjunto de datos en la primer y segunda fila; junto al promedio de todos los ejemplos de la misma clase correspondientes en tercer y cuarta fila.

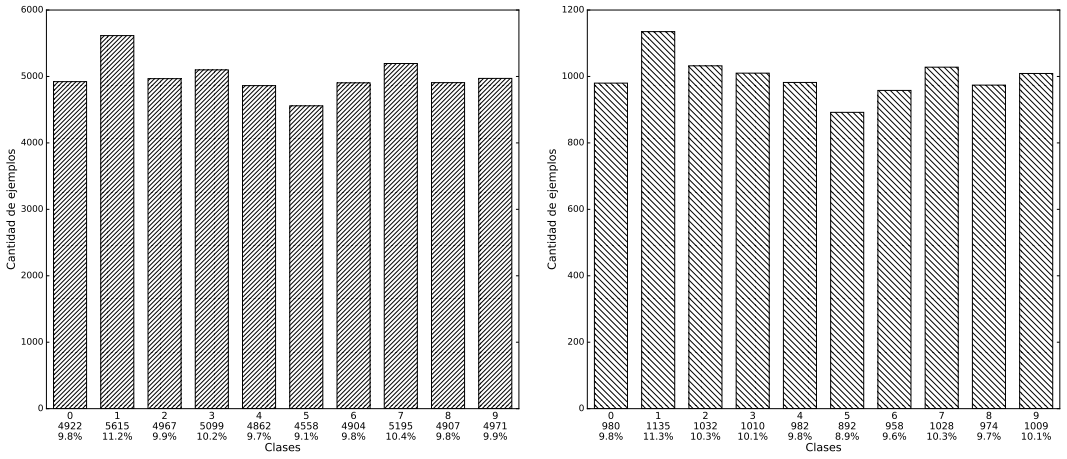
La importancia de esta base de datos radica en los numerosos experimentos que se han realizado con distintos algoritmos a través de los años, lo que permite la comparación entre los modelos neuronales diseñados con respecto a las distintas tendencias actuales del aprendizaje maquinal [27].

4.2.1. Preprocesamiento

Fue necesario realizar los subconjuntos de entrenamiento, validación, y pruebas requeridos para las técnicas de validación cruzada. Cada conjunto se encuentra balanceado con respecto a las clases, donde datos se están repartidos en entre un 9.0 % y 11.0 % aproximadamente para cada clase, tal como puede apreciarse en los histogramas figura 4.2.

Para la ejecución de las pruebas, se genero una serie de scripts desarrollados en lenguaje Python y Bash utilizados para cargar y manipular los datos, crear las redes con su estructura y parámetros, realizar el entrenamiento y ajuste de las mismas, almacenar los resultados y por último registrar en un archivo de texto la salida del proceso. A continuación se detalla de forma sucinta cada uno de ellos,

- `dbn_prueba.py`: script principal para la creación, entrenamiento y ajuste fino de las DBNs. Provee de las funciones `entrenar(...)` y `ajustar(...)`. Se crea la DBN con sus respectivos parámetros (directorio de almacenamiento, cantidad de unidades, tasa de aprendizaje para el entrenamiento y para el ajuste fino, etc).
- `dbn_MNIST.py`: invoca la rutina `dbn_prueba.py` realizando las operaciones necesarias para la carga de la base de datos MNIST (si el recurso se encuentra en disco o se obtiene desde su pagina



(**a**) (**b**)

Fig. 4.2 Cantidades de ejemplos por conjunto, **(a)** Conjunto de entrenamiento, **(b)** Conjunto de validación. En el eje de las abscisas se detalla la etiqueta al cual pertenece, cantidad y el porcentaje de la clases en su respectivo conjunto.

oficial), gestiona las particiones de conjuntos y adapta los datos en caso de que se requiera. Además en el caso que se desee se implementa la funcionalidad de búsqueda por grilla en el espacio de parámetros con `BusquedaEnGrilla(...)` y también se encuentra disponible la opción de ejecutar pruebas con técnicas de validación cruzada.

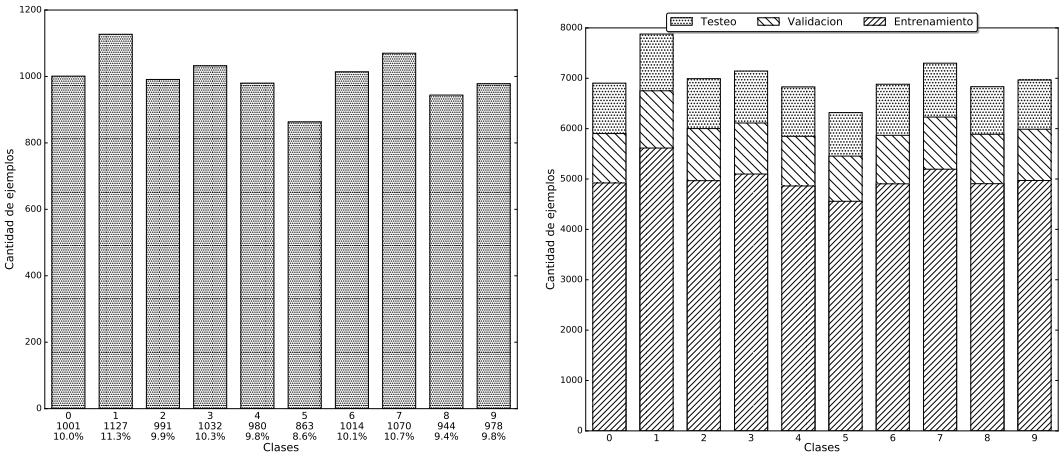
- `corredor_mnist.sh`: ejecuta el programa principal para las pruebas `dbn_MNIST.py`, imprime por consola la salida de la ejecución actual además de guardala en un archivo de texto simple de forma paralela. El mismo calcula los tiempos de ejecución y los imprime al final del archivo.

4.2.2. Experimentos

El objetivo propuesto para los experimentos sobre esta base de datos es analizar el comportamiento de las DBNs en pruebas estándar y, obtenerse así resultados esperados o bien patrones de comportamiento conocido para los modelos. De este modo se deben contemplar diferentes estructuras de redes, es por ello que se definieron varias arquitecturas neuronales, tasas de aprendizaje para el entrenamiento y ajuste fino, tasa de momento, parámetros de regularización, etc.

Se definió la búsqueda de parámetros para las configuraciones de los modelos en un espacio subóptimo por medio de una función implementada en la biblioteca Cupydle, *búsqueda por grilla aleatoria*. De este modo se definieron distintos parámetros manualmente y, luego por una búsqueda aleatoria fueron seleccionados un conjunto reducido el cual fue utilizado para realizar las pruebas. Fijando los siguientes rangos de parámetros:

- Función de activación sigmoidea para cada capa.
- La estructura o topología de los modelos se fijaron base a los experimentos de Hinton y con la premisa de la simplicidad. Las estructuras optadas son (784, 500, 500, 2000, 10) y (784, 500, 10), donde la primer capa corresponde a las entrada (784), luego las capas ocultas y por último la capa de salida (10).
- Se utilizaron RBM Binarias y RBM Gaussianas.



(**c**) (**d**)

Fig. 4.2 *(cont.)* **(c)** Conjunto de pruebas, **(d)** los totales de los tres conjuntos. En el eje de las abscisas se detalla la etiqueta al cual pertenece, cantidad y el porcentaje de la clases en su respectivo conjunto.

- Algoritmo de optimización Divergencia Contrastiva (CD) y Divergencia Contrastiva Persistente (PCD).
- Pasos de Gibbs para los algoritmos de CD y PCD en el rango $[1, 3]$.
- Tamaño del batch $\{10; 100\}$.
- Tasa de aprendizaje para el entrenamiento $\eta = \{0.05; 0.01; 0.10\}$, para el ajuste fino $\eta = \{0.01; 0.05; 0.10\}$.
- Termino de momento para el entrenamiento $\mu = \{0.00; 0.1\}$ y para el ajuste fino $\mu = \{0.00, 0.01; 0.1\}$.
- Tasa de dropout para el entrenamiento y ajuste fino $p = \{0.00; 0.05; 0.50\}$.
- Otros elementos regularizadores para el ajuste como la norma l_1 y l_2 se han fijado a valores nulos.
- Cantidad de épocas máximas para el entrenamiento $e = \{100; 150; 200\}$ y para el ajuste $e = \{1000; 1500; 2000\}$. No se fijo ningún criterio de corte temprano (como ser error mínimo alcanzado) lo que puede inducir a sobreentrenamiento.

Los mejores quince modelos (ordenados por menor tasa de error en conjunto de pruebas) pueden encontrase en la tabla 4.3. Cabe aclarar que se realizaron más de cien pruebas entre versiones CPU y GP-CPU, aunque debido al tiempo necesario de preparación y ejecución de las mismas fueron requeridos más de 13 días continuos de ejecución en la PC aproximadamente.

Se extrajo el mejor resultado para un modelo DBN (#1) y sus los parámetros de ajuste fino fueron utilizados en un clasificador estándar multicapa (#16) con su matriz de pesos idéntica al de redes de creencia profunda al comienzo del entrenamiento. Con dichos modelos se lograron tasas de aciertos en clasificación del **98.27 %** y 97.48 % respectivamente (ver tabla 4.3).

En ambos, el tamaño del batch fue de 100 ejemplos y la estructura correspondiente de red es de 784 unidades de entrada (visibles), 500 unidades ocultas para la segunda y tercer capa, 2000 unidades ocultas para la 4 capa y por último 10 unidades para la capa de salida. Para el caso de la DBN, se entrenó mediante el algoritmo de divergencia contrastiva persistente a un paso de Gibbs. El resto de las configuraciones se encuentra en la tabla 4.4.

			CD	Pasos	Tamaño	Entrenamiento				Ajuste						Tiempos [min]		Menor Error [%]	
#	Tipo	Capas	PCD	Gibbs	Batch	e	η	μ	p	e	η	μ	l_1	l_2	p	GP-GPU	CPU	Val	Tst
1	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	1	100	100	0.05	0.00	0.00	1000	0.10	0.01	0.00	0.00	0.00	**127.01**	-	**1.75**	**1.73**
2	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	1	100	200	0.01	0.00	0.00	2000	0.10	0.00	0.00	0.00	0.00	216.9	-	1.86	2.02
3	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	1	100	200	0.05	0.00	0.00	1000	0.10	0.01	0.00	0.00	0.00	139.8	-	1.95	2.05
4	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	1	10	100	0.01	0.00	0.00	1500	0.10	0.00	0.00	0.00	0.00	682.55	2823.33	2.01	2.18
5	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	3	100	150	0.05	0.00	0.00	2000	0.10	0.01	0.00	0.00	0.00	303.43	-	1.94	2.20
6	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	3	100	100	0.01	0.00	0.00	1500	0.01	0.00	0.00	0.00	0.00	252.04	-	2.33	2.44
7	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	PCD	1	100	150	0.01	0.00	0.00	2000	0.01	0.00	0.00	0.00	0.00	165.08	-	2.23	2.50
8	binaria	784_i 500_{h_1} 10_s	PCD	1	100	200	0.01	0.00	0.00	1500	0.10	0.01	0.00	0.00	0.00	40.87	-	2.41	2.53
9	binaria	784_i 500_{h_1} 10_s	PCD	1	10	100	0.01	0.00	0.00	2000	0.10	0.01	0.00	0.00	0.00	302.50	2405.87	2.53	2.53
10	binaria	784_i 500_{h_1} 10_s	PCD	1	100	150	0.05	0.00	0.00	1000	0.10	0.10	0.00	0.00	0.00	28.87	-	2.42	2.66
11	binaria	784_i 500_{h_1} 10_s	PCD	1	100	150	0.01	0.00	0.00	2000	0.01	0.01	0.00	0.00	0.00	44.26	-	2.66	2.68
12	binaria	784_i 500_{h_1} 10_s	PCD	1	100	150	0.01	0.00	0.00	1000	0.10	0.01	0.00	0.00	0.00	28.58	-	2.51	2.75
13	binaria	784_i 500_{h_1} 10_s	PCD	1	10	200	0.05	0.00	0.00	2000	0.10	0.00	0.00	0.00	0.00	369.66	2433.40	2.66	3.02
14	binaria	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	CD	1	100	100	0.05	0.00	0.00	2000	0.10	0.00	0.00	0.00	0.00	196.51	-	3.35	3.08
15	binaria	784_i 500_{h_1} 10_s	CD	3	10	150	0.01	0.00	0.00	1500	0.10	0.00	0.00	0.00	0.00	288.06	-	2.93	3.31
16	MLP	784_i 500_{h_1} 500_{h_2} 2000_{h_3} 10_s	-	-	100	1000	0.01	0.00	0.00	-	-	-	-	-	-	75.50	-	2.49	2.52

Tabla 4.3 Experimentos en MNIST handwritten database. Distintas pruebas, cada una con sus respectivos parámetros y resultados.
Capas: entrada$_i$ primer capa oculta$_{h_1}$ ... última capa$_s$
Épocas: e; Tasa de aprendizaje: η; Tasa momento: μ; Regularizadores: L_1, L_2.
Modelos no ejecutados en modo CPU solo (-)

Tipo	Entrenamiento				Ajuste						**Acierto [%]**	
	e	η	μ	p	e	η	μ	l_1	l_2	p	**Val.**	**Prueba**
DBN	100	0.05	0.0	0.0	1000	0.1	0.1	0.0	0.0	0.0	98.25	**98.27**
MLP	1000	0.1	0.1	0.0	-	-	-	-	-	-	97.51	97.48

Tabla 4.4 DBN versus MLP MNIST. Síntesis derivada de la tabla 4.3. Mejor resultado obtenido con una DBN y en contraposición, con los mismos parámetros un clasificador estándar.
Épocas: e; Tasa de aprendizaje: η; Tasa momento: μ; Regularizadores: L_1, L_2; Tasa de dropout: p.

Filtros

La figura 4.3 muestra los pesos de una RBM (correspondiente a la primer capa de la DBN en la tabla 4.4) con 784 unidades visibles y sus 500 ocultas entrenada a lo largo de 100 épocas. Cada cuadrado o *patche* se corresponde a los 784 pesos de 1 unidad oculta para las 784 neuronas visibles de un ejemplo. En la primer imagen se visualizan los filtros al inicio del proceso, seguidamente después de 25 épocas, luego de 75 épocas y al finalizar el proceso en la iteración 100.

Solo los cien primeros filtros de las primeras unidades ocultas son visualizadas. Estos son complejos en detalles y dado a la representación de los datos de entrada (dígitos) es posible reconocer patrones característicos que se dibujan sobre estos. Cuando es mayor la cantidad de neuronas utilizadas, los campos receptivos se vuelven más localizados y muestran características bien marcadas. Se observa que a medida que la red "aprende" la distribución de los datos, cada filtro se parece más forma de los datos. Este comportamiento es compartido con otros trabajos sobre la misma temática [20, 3, 21]

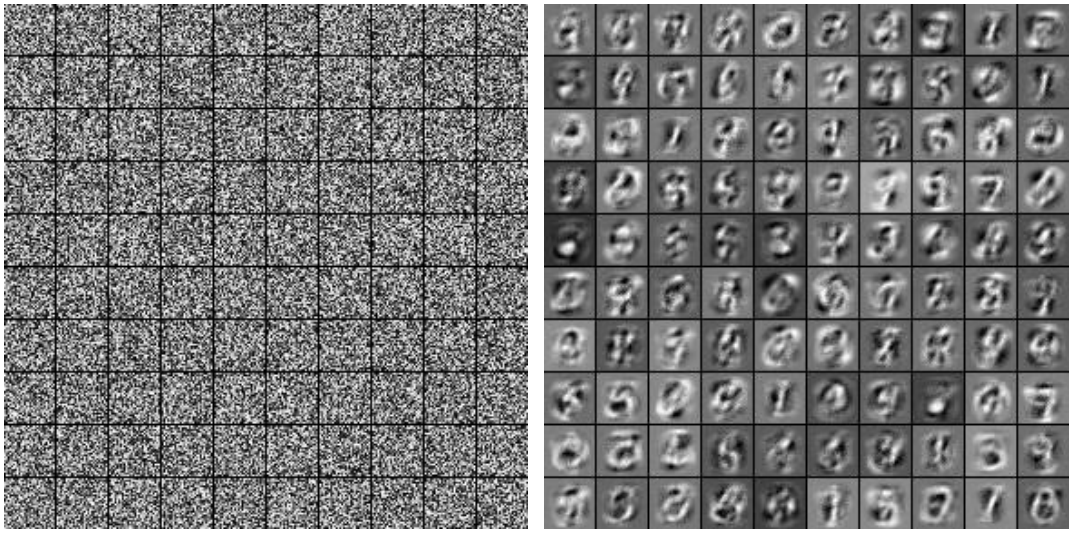
Matriz de confusión

Al final del entrenamiento y posterior ajuste fino de los pesos de la DBN, una herramienta útil para determinar el desempeño en la tarea de clasificación del modelo es la llamada *matriz de confusión*. En ella, sus filas representan las clases reales del conjunto de pruebas, y por columna las predicciones que el modelo realiza sobre cada ejemplo del mismo conjunto. La diagonal denota la precisión en la clasificación (etiqueta real versus predicción del sistema) y valores alejados de la diagonal donde el modelo fallo. Es de esperarse en un clasificador que fue bien entrenado su matriz de confusión sea lo más diagonal posible. La matriz de confusión de modelo DBN y MLP de la tabla 4.4 puede encontrarse en la figura 4.4. La figura de la izquierda corresponde a la DBN y a su derecha la de clasificador estándar.

La matriz asociada a la DBN determina la performance que se obtuvo con el correspondiente modelo al clasificar el conjunto de pruebas, con una tasa de error del 1.73 % el clasificador de la DBN asocia de forma correcta 9827/10000 ejemplos. Se puede inferir que el sistema se equivoca con mayor frecuencia al clasificar el dígito tres, un 3 % del total aunque acierta con una tasa del 99.31 % al clasificar el dígito uno.

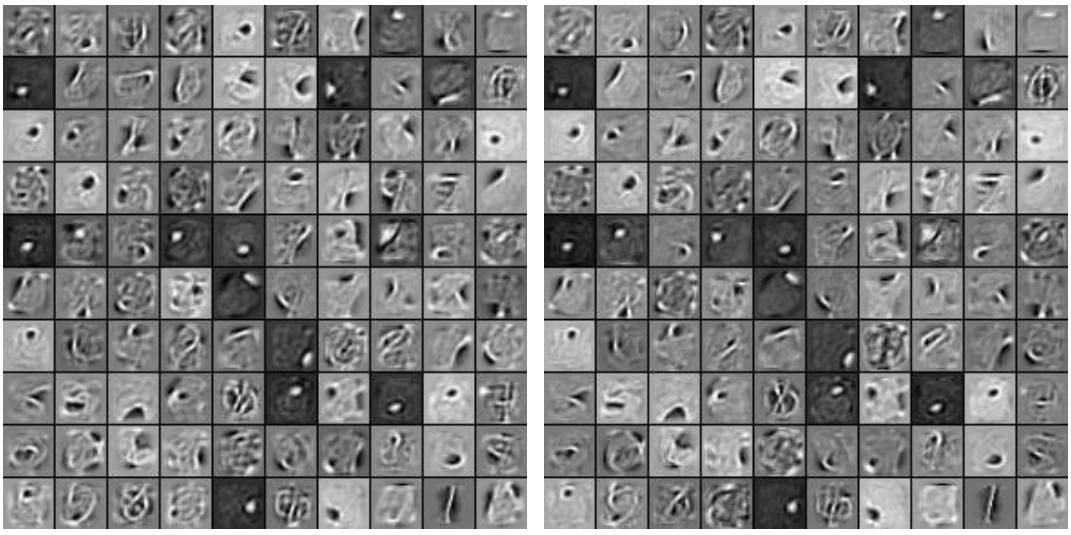
4.2.3. Resultados: DBN versus MLP

Para comparar el rendimiento de las DBNs se analizaron los resultados obtenidos en la tabla 4.4 con un clasificador estándar multicapa. El clasificador fue configurado con los parámetros correspondientes al ajuste de la DBN. Las pruebas realizadas demostraron que la performance de las DBN es mejor con



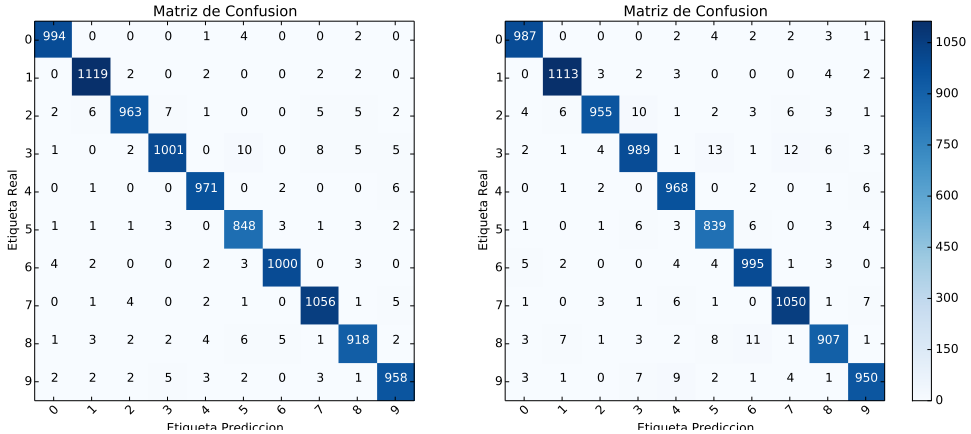
(**a**) (**b**)

Fig. 4.3 Filtros de los primeros 100 pesos de la RBM para el modelo DBN de la tabla 4.4. **(a)** Al inicio del proceso, época 0; **(b)** a la iteración 25, 25 % del entrenamiento.



(**c**) (**d**)

Fig. 4.3 *(cont.)* Filtros de los primeros 100 pesos de la RBM para el modelo DBN de la tabla 4.4. **(c)** en la iteración 75, **(d)** Iteración final 100. Notar que se pueden se pueden divisar formas, bordes en los pesos a medida que la red realiza el proceso de aprendizaje.



(**a**) (**b**)

Fig. 4.4 Matriz de confusión: Las filas contabilizan las etiquetas reales. Las columnas contabilizan las predicciones. Notar que la diagonal son los aciertos. (**a**) Matriz de confusión de la ejecución del modelo DBN de la tabla 4.4. La red falló al clasificar 173 ejemplos, analizando la misma, se puede afirmar que la red se equivoca con mayor frecuencia el dígito 3 con un 5 (10 veces en total) o por un 7. El error es mínimo al predecir el dígito 0, $\approx 99.31\,\%$ de acierto.(**b**) Matriz de confusión del clasificador estándar de la tabla 4.4. Con resultados similares a la matriz (a) aunque se contabilizan 253 errores al clasificar los dígitos.

respecto a los clasificadores multicapa, experimentos dieron como resultado que el error de validación y pruebas del ajuste fino en la DBN es siempre menor al ajuste del MLP. Además se tiene que el costo de entrenamiento en la DBN (1.20785) es superior en comparación al clasificador (0.51503) en lo cual la DBN puede ajustar aun más el sistema; mientras que el MLP una vez llegado al costo nulo no realizará ningún cambio en el modelo. En consecuencia los modelos DBNs generarán mejores sistemas de clasificación para las tareas de reconocimiento de dígitos manuscritos. En la figura 4.5, con escala logaritmica en el eje-y puede observar el comportamiento anterior.

4.3. RML Emotion Database: clasificación de emociones humanas

La base de datos RML fue sugerida por los directores como objeto de aplicación para este proyecto. Si bien existen varias bases de datos relacionadas a emociones humanas, los criterios de aceptación y justificación de su elección puede hallarse en [2]. Algunos de estos criterios son: base de datos de emociones prototípicas con frecuencia de muestreo de 30 fps, con suficiente cantidad de actores y muestras, citada con frecuencia en trabajos actuales de la comunidad científica; además los actores no presentan ningún tipo de marcador facial lo que hace a esta base de datos conveniente para probar las redes de creencia profunda.

RML es un conjunto de 720 archivos multimedia (video: imagen+audio). Puede adquirirse en su sitio oficial[1]. Además el análisis de los criterios de recopilación para la misma puede encontrarse en [52].

[1]Ryerson Multimedia Research Lab: http://www.rml.ryerson.ca/rml-emotion-database.html

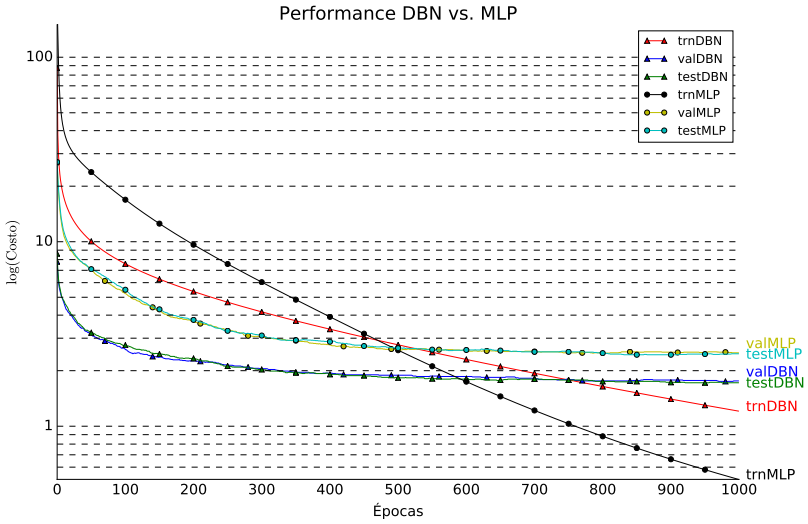


Fig. 4.5 DBN vs. MLP: Comparación en la performance de los sistemas. Evolución del costo de entrenamiento, error de validación y prueba obtenido en el proceso de ajuste fino del modelo DBN y entrenamiento-ajuste de clasificador MLP en la tabla 4.4. El eje vertical se encuentra en escala logarítmica los costos del entrenamiento de los modelos. Notar como el error sobre el conjunto de validación y pruebas del la DBN **siempre** es menor al correspondiente error del MLP.

La base de datos cuenta con las siguientes características:

- 720 expresiones de emociones audiovisuales posadas.
- 6 emociones humanas expresadas: enojo, asco, miedo, felicidad, tristeza y sorpresa; neutral (ausencia de emoción) al comienzo de cada secuencia de video. 120 ejemplos por clase.
- entorno controlado, iluminado y fondo simple.
- 10 sentencias de audio diferentes por cada emoción.
- independiente de lenguaje y cultura (8 sujetos hablando 6 lenguajes diferentes: ingles, mandarín, urdú, punyabí, persa e italiano).
- velocidad de muestreo: 22050 Hz, canal simple de 16-bit, 30 fps, formato AVI.
- origen canadiense.
- extensa cantidad de trabajos científicos que la utilizan (mayor a 23) [2].

4.3.1. Preprocesamiento

La base de datos en su estado puro, contiene 720 archivos de video. Cada uno de éstos cuenta con una resolución de 320 píxeles de ancho por 240 píxeles de alto, una tasa de muestreo de 30 fotogramas por segundo, además la pista de audio cuenta con una tasa de muestreo equivalente a 44100 Hz a razón de 16 bits por muestra de un único canal. Cada archivo multimedia tiene una duración de 4 a 6 segundos aproximadamente. Si se toma como promedio unos 5 segundos de duración para cada uno, se tienen unos 12 millones de valores aprox. en estado puro (video: $320 \times 240 \times 30 \times 5$; audio: 44100×5). En Theano la precisión para valores de punto flotante es de 32 bits, equivalentes a 4 bytes (menor precisión aún se encuentra en fase experimental). En consecuencia, si tomáramos dichos datos como entrada a la red y con tan solo una capa oculta de 1000 unidades además de la capa final (6

etiquetas) se requeriría un total de 47 Gbs aproximadamente en memoria para alojar los vectores de entrada y salida, las matrices de pesos y respectivos biases y demás arreglos en la GP-GPU.

Por lo tanto, resulta no practico hasta físicamente imposible con los recursos disponibles aplicar los datos puros a la entrada de una DBN. Es por ello que se procede reducir la dimensionalidad de los datos por medio de extracción de características del audio y video, tal como se indica en [10, 2].

4.3.2. Extracción de características en video

La extracción de características en video se realizó mediante programas y scripts provistos por los directores del proyecto y utilizados en [2]. La idea principal es encontrar las regiones de interés por frame de video, éstas corresponden a la zona de la boca y a la zona de los ojos del sujeto. Se optó por seleccionar dichas regiones dado que son las que suponen mayor contenido emocional en las expresiones faciales [28].

Los pasos a seguir son, en primer lugar seleccionar los frames en el video donde la persona realiza o expresa algún gesto. Seguidamente detectar el rostro de la persona, donde se deben determinar los puntos faciales de interés (*facial landmarks*). Luego se segmentan las zonas de interés (ojos y boca). Por último se normalizan los datos.

Para el primer paso, se observó que en cada uno de los ejemplos los sujetos parten del reposo con ausencia de emoción alguna, expresan una frase verbal con carga emocional y regresan al reposo. Debido a esto se debe cortar el video donde la persona comienza y termina de hablar. Realizando esta tarea para todos los ejemplos se calculó un promedio de 47 frames de los fotogramas de interés, se utiliza dicha cantidad (centrada) descartando los frames extremos.

Para el segundo paso, con ayuda de la librería externa `flandmark`[2] se detectan los puntos faciales de interés. Estos permiten determinar la localización del centro de la cara (ϵ_0), ojos izquierdo y derecho (ϵ_5, ϵ_1 y ϵ_2, ϵ_6 respectivamente), la boca (ϵ_6 y ϵ_4) y por último la nariz (ϵ_7). Un total de 8 puntos tal como se muestra en la figura 4.6. Una vez determinados los puntos de interés, se procede a segmentar las zonas de interés. Considerando ciertos márgenes para capturar información relevante como cejas y pequeñas muecas alrededor de la boca como se observa en la figura 4.6. Cada región de interés (boca y ojos) queda fijada a un tamaño de 70×35 píxeles, lo que da un total de 4900 valores por frame de video. Se tiene al final un total de 230300 valores por archivo de video. Se normalizan los resultados obtenidos en el rango [0,255].

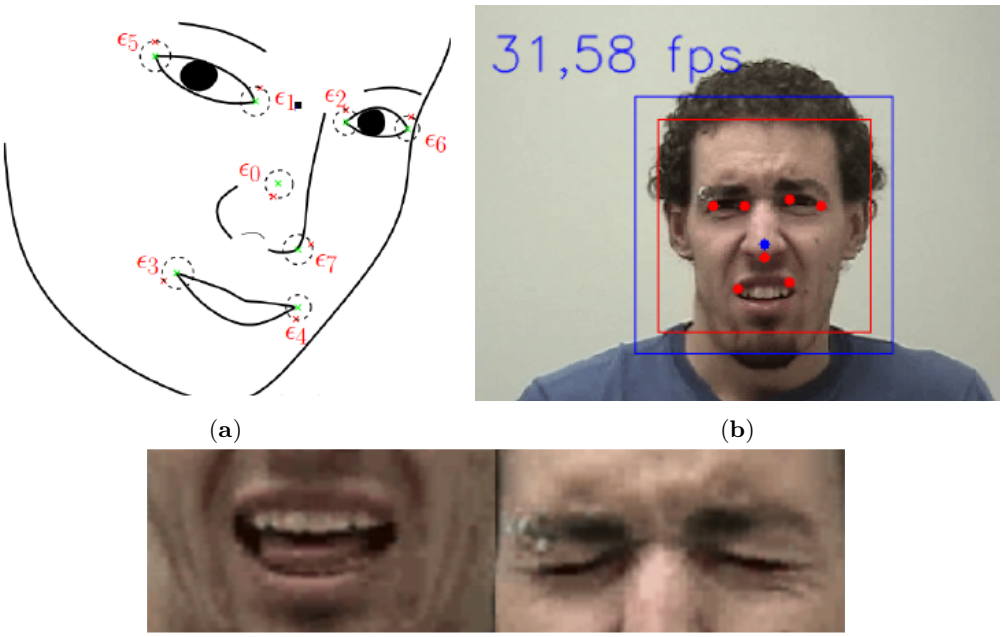
Para generar los nuevos espacios, se aplico también técnicas de reducción de dimensionalidad como ser PCA[3]. Se optó por retener al menos el 90 % de la varianza para cada una de las emociones, lo que significó fijar 85 componentes principales por espacio.

Con el procedimiento anterior se logró una reducción significativa en la dimensión de los datos, solo se retiene el $\frac{230300}{11520000} \approx 0.02\,\%$ del conjunto original. La memoria requerida es unas 50 veces menor que si se utiliza los datos puros.

4.3.3. Extracción de características en audio

Con el objetivo de reducir la dimensión de las pistas de audio y conservar propiedades que representen a los datos, se extraen de las mismas tres tipos de características: *prosódicas* (energía y frecuencia fundamental), *espectrales* (coeficientes Media del Espectro Logarítmico –MLS–) y las *cepstrales*

[2]Pagina oficial: http://cmp.felk.cvut.cz/~uricamic/flandmark/
[3]Para más información recurra a [17].



(**a**) (**b**)

(**c**)

Fig. 4.6 Extracción de características de videos. (**a**) los ocho puntos de interés determinados por la librería `flandmark`. (**b**) Aplica la funcionalidad de `flandmark` a la base de datos KML. (**c**) Zonas de interés, boca y ojos. Las mismas están conformadas por 70×35 píxeles.

(coeficientes cepstrales en escala de mel –MFCC–). Mas información sobre la elección y justificación de estas características puede encontrase en los trabajos [1, 2].

El vector de características de audio, queda conformado entonces por: 12 medias de los MFCCs, calculadas a través de ventanas en el dominio del tiempo del tipo Hamming con un tamaño de 1024 frames de audio. 30 coeficientes MLS. La media y el desvío estándar de la energía de corta duración, con ventanas de 10 ms. La media y el desvío de la frecuencia fundamental con un tamaño de ventana de 10 ms. Lo que hace un total de 46 componentes.

Gracias a este procedimiento se reducen las pistas de audio de un promedio de 5 segundos de duración, unos 220500 valores a tan solo 46 valores representativos. Esto es en definitiva una reducción de la dimensión a un 0.02 % de la original.

4.3.4. Experimentos

Las pruebas se definieron en función de evaluar el desempeño de diferentes arreglos de características sobre el sistema y modelos de DBNs, con el fin de determinar que contribución aporta cada uno en la tarea de clasificación. Definir la estructura de la red no es una tarea sencilla, de hecho no existe reglas a seguir y es por ello que se determina de manera empírica. Se desgloso el conjunto de pruebas en dos partes, la primera explora en un espacio reducido de estructuras de redes cual será la que mejor se adapta al problema en cuestión. La segunda, analiza el desempeño de la mejor estructura con respecto a los arreglos de características realizando un ajuste de la red un mayor número de épocas.

Los arreglos de características contienen los datos de audio y video de forma conjunta o por separado. En primer instancia se generó el arreglo de características $\mathbf{a}_{46}$, que corresponden a los 46

Simplificación	Arreglo/os	Contenido
$\mathbf{\Lambda}_1$	$\mathbf{a}_{46}$	Coeficientes (46) de audio.
$\mathbf{\Lambda}_2$	$\mathbf{v}_{230300}$	Píxeles de la zona de interés (video).
$\mathbf{\Lambda}_3$	$\mathbf{v}_{230300} + \mathbf{a}_{46}$	Combinación del imagen+audio.
$\mathbf{\Lambda}_4$	PCA($\mathbf{a}_{230300}$)	PCA sobre $\mathbf{A}_3$ de dimensión 85.

Tabla 4.5 Arreglo de características: Diferentes conjuntos de datos organizados en arreglos de características del audio, video y/o mezcla de las anteriores.

coeficientes extraídos del audio tal como se detalla en la sección 4.3.3. De manera similar se conforma el arreglo $\mathbf{v}_{230300}$ donde se alojan los píxeles de la zona de interés del contenido visual, 340300 valores como se explicó en sección 4.3.2. También se creó un arreglo reducido de $\mathbf{v}_{230300}$, resultado de aplicar técnicas de PCA sobre el conjunto. El mismo contiene un total de 85 características el cual que asegura retener al menos el 90 % de la varianza por conjunto al aplicar la transformación. También se define un arreglo combinación de los dos primeros, todo se detalla en la tabla 4.5.

Para la primer etapa de experimentos, búsqueda de la estructura neuronal que mejor se adapte al problema utiliza el módulo de `gridRandomSearch`. Se definieron un conjunto de hiperparámetros a explorar de forma similar a la sección 4.2.2 como sigue,

- Función de activación: sigmoidea. Para todos los modelos.
- Se utilizaron RBM Binarias y RBM Gaussianas.
- Algoritmo de optimización Divergencia Contrastiva (CD) y Divergencia Contrastiva Persistente (PCD).
- Pasos de Gibbs para los algoritmos de CD y PCD $\in \{1; 5; 10\}$.
- Tamaño del batch $\in \{6; 10; 50; 100\}$.
- Tasa de aprendizaje para el entrenamiento $\eta = \{0.0001; 0.001; 0.01; 0.05; 0.1\}$, para el ajuste fino $\eta = \{0.01; 0.05; 0.10; 0.20\}$.
- Termino de momento para el entrenamiento $\mu = \{0.00; 0.001; 0.01; 0.1\}$ y para el ajuste fino $\mu = \{0.00, 0.01; 0.1; 0.2\}$.
- Tasa de dropout para el entrenamiento y ajuste fino $p = \{0.00; 0.30; 0.40; 0.50\}$.
- Regularizadores $l_1 = \{0.00; 0.001; 0.010; 0.100\}$ y $l_2 = \{0.00; 0.0001; 0.001; 0.100\}$.
- Cantidad de épocas máximas para el entrenamiento fijada en $e = \{50\}$ y para el ajuste $e = \{500\}$. No se fijo ningún criterio de corte temprano.
- La estructura de los modelos se fijaron según los arreglos de características con varias capas ocultas. Para el arreglo Λ_1 se definieron combinaciones de capas ocultas de $\{50; 60\}$. Para el arreglo $\Lambda_2 \Rightarrow \{85; 100; 500; 1000; 2000; 3000\}$. En el arreglo $\Lambda_3 \Rightarrow \{100; 1000; 1200; 2000; 3000; 4000\}$. Por último, el arreglo $\Lambda_4 \Rightarrow \{50; 60\}$. Para todos los casos se fijaron también estructuras sin capa oculta (unidades visibles $\rightarrow$ unidades salida).

Los mejores diez modelos (ordenados por arreglo de características y menor tasa de error en conjunto de pruebas) pueden encontrase en la tabla 4.6. Los mejores para cada arreglo $\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4$ (posiciones en la tabla #1, #11, #21, #31) alcanzaron tasas de error en el conjunto de pruebas 41.66 %, 28.33 %, 24.16 %, 31.66 % respectivamente. Se realizaron más de un centenar de pruebas en forma de lotes, lo que requirió más de 20 días continuos de ejecución en la PC aproximadamente.

#	Arreglo	Capas	Tipo	CD	Pasos	Tamaño	Entrenamiento				Ajuste						Tiempo	Error [%]	
				PCD	Gibbs	Batch	e	η	μ	p	e	η	μ	l_1	l_2	p	[min:seg]	Val.	Prueba
1 ◄	Λ_1	46_i 60_{h_1} 6_s	Binaria	CD	5	6	50	0.0010	0.001	0.00	500	0.200	0.100	0.010	0.0000	0.00	01:51	44.79	41.66
2	Λ_1	46_i 60_{h_1} 6_s	Binaria	CD	5	6	50	0.0100	0.100	0.00	500	0.200	0.200	0.100	0.0000	0.00	02:15	47.91	41.66
3	Λ_1	46_i 6_s	Binaria	PCD	5	6	50	0.0100	0.100	0.00	500	0.050	0.200	0.010	0.0001	0.50	01:05	63.54	42.50
4	Λ_1	46_i 6_s	Binaria	PCD	5	10	50	0.0500	0.001	0.50	500	0.100	0.100	0.100	0.0010	0.00	00:42	56.00	44.16
5	Λ_1	46_i 50_{h_1} 6_s	Binaria	PCD	1	6	50	0.0010	0.001	0.00	500	0.200	0.100	0.100	0.0010	0.50	01:17	45.83	45.00
6	Λ_1	46_i 50_{h_1} 6_s	Binaria	PCD	10	6	50	0.0100	0.001	0.00	500	0.100	0.200	0.100	0.0000	0.00	02:15	60.41	45.83
7	Λ_1	46_i 60_{h_1} 6_s	Binaria	PCD	1	6	50	0.0500	0.001	0.50	500	0.100	0.000	0.100	0.0001	0.00	01:10	44.79	46.66
8	Λ_1	46_i 6_s	Gaussiana	PCD	10	10	50	0.0500	0.100	0.00	500	0.100	0.100	0.000	0.0000	0.00	00:48	46.00	46.66
9	Λ_1	46_i 50_{h_1} 6_s	Binaria	CD	5	6	50	0.0500	0.001	0.00	500	0.200	0.100	0.100	0.0001	0.40	01:54	56.25	46.66
10	Λ_1	46_i 6_s	Gaussiana	PCD	5	10	50	0.0001	0.001	0.00	500	0.100	0.100	0.000	0.0000	0.00	00:41	64.00	46.66
11 ◄	Λ_2	230300_i 6_s	Gaussiana	PCD	1	10	50	0.0100	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.00	139:04	27.00	28.33
12	Λ_2	230300_i 1200_{h_1} 1000_{h_1} 6_s	Binaria	PCD	10	10	50	0.0100	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.00	238:39	65.00	65.00
13	Λ_2	230300_i 1000_{h_1} 4000_{h_2} 100_{h_3} 6_s	Gaussiana	PCD	5	6	50	0.0001	0.000	0.00	500	0.100	0.000	0.000	0.0000	0.00	254:22	83.33	77.50
14	Λ_2	230300_i 1000_{h_1} 1000_{h_2} 6_s	Gaussiana	PCD	5	6	50	0.0010	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.00	319:36	78.12	82.50
15	Λ_2	230300_i 1000_{h_1} 3000_{h_2} 3000 $_{h_3}$ 100_{h_4} 6_s	Binaria	PCD	5	50	50	0.0100	0.000	0.00	500	0.100	0.100	0.001	0.0010	0.00	137:24	86.00	84.00
16	Λ_2	230300_i 85_{h_1} 100_{h_2} 6_s	Gaussiana	PCD	10	6	50	0.0010	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.00	211:09	89.58	84.16
17	Λ_2	230300_i 85_{h_1} 500_{h_2} 6_s	Gaussiana	PCD	10	6	50	0.0010	0.100	0.50	500	0.100	0.100	0.010	0.0010	0.00	260:12	81.25	85.00
18	Λ_2	230300_i 1000_{h_1} 1000_{h_2} 6_s	Gaussiana	PCD	5	6	50	0.0010	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.00	319:36	78.12	82.50
19	Λ_2	230300_i 85_{h_1} 6_s	Binaria	CD	5	6	50	0.1000	0.000	0.20	500	0.100	0.000	0.000	0.0000	0.40	162:07	80.20	85.83
20	Λ_2	230300_i 85_{h_1} 6_s	Binaria	PCD	5	6	50	0.1000	0.000	0.00	500	0.100	0.000	0.000	0.0000	0.00	162:05	82.29	87.50
21 ◄	Λ_3	230346_i 6_s	Gaussiana	PCD	1	6	50	0.0001	0.000	0.00	500	0.100	0.000	0.000	0.0000	0.00	120:37	16.66	24.16
22	Λ_3	230346_i 6_s	Binaria	PCD	5	6	50	0.0100	0.000	0.00	500	0.100	0.100	0.001	0.0010	0.00	156.32	14.58	25.83
23	Λ_3	230346_i 1200_{h_1} 1000_{h_2} 6_s	Binaria	PCD	3	6	50	0.0010	0.100	0.00	500	0.100	0.100	0.000	0.0000	0.00	340:31	52.08	56.66
24	Λ_3	230346_i 2000_{h_1} 1000_{h_2} 6_s	Gaussiana	PCD	5	6	50	0.0010	0.000	0.00	500	0.100	0.000	0.000	0.0000	0.30	312:04	78.12	81.66
25	Λ_3	230346_i 1000_{h_1} 4000_{h_2} 100_{h_3} 6_s	Gaussiana	PCD	5	6	50	0.0010	0.000	0.40	500	0.100	0.000	0.000	0.0000	0.00	239:34	90.62	82.50
26	Λ_3	230346_i 1000_{h_1} 4000_{h_2} 6_s	Binaria	PCD	5	6	50	0.0100	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.00	325:30	71.87	85.00
27	Λ_3	230346_i 1000_{h_1} 3000_{h_2} 3000_{h_3} 1000_{h_4} 6	Binaria	PCD	3	6	50	0.1000	0.100	0.00	500	0.100	0.100	0.000	0.0001	0.40	636:36	80.20	85.00
28	Λ_3	230346_i 1000_{h_1} 3000_{h_2} 3000_{h_3} 1000_{h_4} 6_s	Gaussiana	PCD	5	6	50	0.0001	0.000	0.00	500	0.100	0.000	0.000	0.0000	0.00	199:38	76.04	86.66
29	Λ_3	230346_i 1000_{h_1} 1000_{h_2} 6_s	Gaussiana	PCD	5	6	50	0.0010	0.000	0.00	500	0.100	0.100	0.000	0.0000	0.40	324:33	79.16	87.50
30	Λ_3	230346_i 1000_{h_1} 2000_{h_2} 6_s	Binaria	CD	5	6	50	0.1000	0.000	0.00	500	0.200	0.000	0.000	0.0000	0.00	314:33	83.23	88.30
31 ◄	Λ_4	85_i 6_s	Binaria	PCD	10	6	50	0.0001	0.001	0.00	500	0.200	0.000	0.000	0.0001	0.00	00:46	38.00	31.66
32	Λ_4	85_i 6_s	Gaussiana	CD	10	10	50	0.0010	0.100	0.00	500	0.050	0.000	0.100	0.0000	0.00	00:29	48.00	32.50
33	Λ_4	85_i 6_s	Gaussiana	PCD	1	10	50	0.0500	0.001	0.00	500	0.200	0.100	0.100	0.0000	0.00	00:31	34.00	33.33
34	Λ_4	85_i 6_s	Gaussiana	CD	5	10	50	0.0500	0.100	0.00	500	0.100	0.000	0.000	0.0000	0.00	00:40	35.00	33.33
35	Λ_4	85_i 6_s	Binaria	PCD	5	10	50	0.0010	0.000	0.00	500	0.100	0.200	0.000	0.0000	0.00	00:40	42.00	33.33
36	Λ_4	85_i 6_s	Binaria	PCD	5	50	50	0.0100	0.000	0.00	500	0.200	0.100	0.010	0.0001	0.00	00:14	53.00	34.00
37	Λ_4	85_i 6_s	Gaussiana	CD	1	10	50	0.0500	0.100	0.50	500	0.200	0.000	0.000	0.0001	0.00	00:29	34.00	34.16
38	Λ_4	85_i 6_s	Gaussiana	PCD	1	6	50	0.0010	0.000	0.00	500	0.200	0.100	0.010	0.0000	0.40	00:47	27.08	35.00
39	Λ_4	85_i 50_{h_1} 6_s	Binaria	PCD	1	10	50	0.0100	0.100	0.00	500	0.100	0.000	0.100	0.0000	0.00	00:22	34.00	36.66
40	Λ_4	85_i 6_s	Binaria	CD	10	6	50	0.0001	0.001	0.00	500	0.100	0.100	0.010	0.0000	0.30	01:15	37.50	37.50

Tabla 4.6 Experimentos RML emotion Database para diversas *arquitecturas* neuronales. Distintos modelos de redes/arquitecturas para la exploración del subconjunto óptimo. Resultados ordenados por vector de características seguido por error en el conjunto de pruebas (menor mejor)
Capas: entrada$_i$ primer capa oculta$_{h_1}$ ... última capa$_s$
Épocas: e; Tasa de aprendizaje: η; Tasa momento: μ; Regularizadores: L_1, L_2.

Arreglo	**Estructura**	**% Acierto DBN**	**% Acierto MLP**
$\mathbf{\Lambda}_1$	$46_i\ 60_{h_1}\ 6_s$	73.48 %	59.03
$\mathbf{\Lambda}_2$	$230300_i\ 6_s$	74.17 %	70.00
$\mathbf{\Lambda}_3$	$230346_i\ 6_s$	70.70 %	69.87
$\mathbf{\Lambda}_4$	$85_i\ 6_s$	**74.87 %**	72.51

Tabla 4.7 Perfomance de los mejores modelos para los distintos arreglos de la tabla 4.6 marcados con ◄. Para todos los casos las pruebas se hicieron en base a validación cruzada. Se comparan modelos de DBN vs. clasificador estándar MLP. El mejor resultado obtenido fue el arreglo producto del PCA sobre el arreglo de zonas de interés con un desempeño del 74.87 % de acierto.

Para la segunda etapa, se evalúa la perfomance que tienen los mejores modelos encontrados sobre cada arreglo de características, luego se realiza un "ajuste" con mayor número de épocas. Los modelos elegidos son los que han logrado la menor tasa de error en clasificar las emociones, denotados con ◄ de la tabla 4.6. Se compara cada resultado con un clasificador estándar MLP con parámetros de configuración iguales al ajuste fino de las DBN y pesos iniciales iguales al comienzo del proceso. Las DBNs fueron entrenadas 1000 épocas para los arreglos y con un máximo de 30000 épocas para ajuste fino.

Para el arreglo $\mathbf{\Lambda}_3$, se probó con una estructura $230346_i\ 6_s$ alzando una tasa de acierto de 70.70 % para el modelo de DBN. En cambio el modelo de MLP logró una performance apenas por debajo con un 69.87 %.

Seguidamente se puso a prueba el arreglo $\mathbf{\Lambda}_1$, con una estructura $46_i\ 60_{h_1}\ 6_s$, en el modelo de DBN se alcanzó una tasa de acierto de 73.48 %. Por otro lado en el modelo de MLP se observó un desempeño mucho menor, con 59.03 % de eficacia al clasificar.

El siguiente experimento con el arreglo $\mathbf{\Lambda}_2$, la estructura analizada fue $230300_i\ 6_s$, en el modelo de DBN se alcanzó una tasa de 74.17 % y para el clasificador estándar un 70.00 % en la tarea.

El mejor experimento logrado fue gracias al modelo $85_i\ 6_s$ para el arreglo $\mathbf{\Lambda}_4$ con una tasa de acierto del 74.87 %. Al compararse el mismo modelo con un MLP se obtuvo una performance de 72.51 %. El modelo DBN fue dos puntos porcentuales superior al clasificador estándar. El resultado obtenido para las redes de creencia profunda tiene un rendimiento superior en 58 % en comparación a la elección al azar con las seis clases. Los resultados obtenidos se sintetizan en la tabla 4.7.

El desempeño del modelo queda claro con la matriz de confusión 4.7, donde se observa la performance sobre cada clase del conjunto de pruebas. En este caso, el modelo obtuvo mejores resultados al clasificar la clase 1, *asco* con una tasa de acierto del 92.6 %. Por otro lado, su peor desempeño fue con una tasa del 46.6 % en la clase 2, miedo.

4.3.5. Evaluación del tiempo

Como característica crítica e indispensable de la biblioteca es realizar el entrenamiento de las redes en GP-GPU de forma veloz en comparación a pruebas secuenciales en CPU. En casos reales gestionar el análisis comparativo entre dos arquitecturas heterogéneas no es una tarea sencilla, solo se tendrá en cuenta el tiempo de ejecución, omitiéndose otras variables.

Se realizaron seis pruebas con diferentes estructuras de red con y sin ayuda de la GP-GPU. Fijando todos los parámetros a excepción de la estructura (la cual es la única que influye en el proceso); se eligió entrenar la DBN por cinco épocas y ajustarla con cincuenta, los demás parámetros tales como

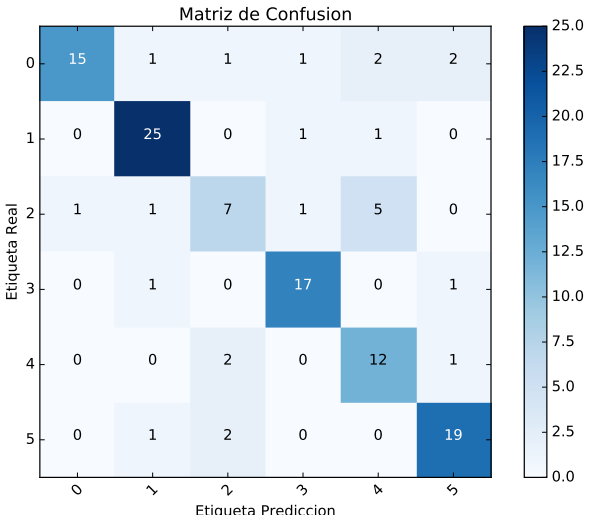


Fig. 4.7 Matriz de confusión mejor resultado sobre RML emotion database. Corresponde al modelo 85_i 6_s del vector de características PCA (tabla 4.7). Las clases numeradas del 0 al 5 se asocian con 0-enojo, 1-asco, 2-miedo, 3-felicidad, 4-tristeza y 5-sorpresa. Con esta herramienta se determina que la emoción que el sistema mejor clasifica es *1-asco* con una tasa de acierto del 92.6 %. Por otro lado, la clase que menos acertó fue 2-miedo con una tasa de acierto del 46.6 %.

las tasas de aprendizaje, regularizadores y tasa de momento y dropout se subestimaron o igualaron a valores nulos. Las estructuras elegidas fueron las siguientes:

- E_1: $230346_i \rightarrow 6_s$
- E_2: $230346_i \rightarrow 1000_{h_1} \rightarrow 6_s$
- E_3: $230346_i \rightarrow 1000_{h_1} \rightarrow 100_{h_2} \rightarrow 6_s$
- E_4: $230346_i \rightarrow 1000_{h_1} \rightarrow 1000_{h_2} \rightarrow 6_s$
- E_5: $230346_i \rightarrow 1000_{h_1} \rightarrow 1000_{h_2} \rightarrow 100_{h_4} \rightarrow 6_s$
- E_6: $230346_i \rightarrow 1000_{h_1} \rightarrow 1000_{h_2} \rightarrow 1000_{h_3} \rightarrow 1000_{h_4} \rightarrow 6_s$

Los resultados fueron los siguientes: para el experimento de la estructura E_1 requirió 11.54 min en el huésped y 15.39 min con ayuda de la GPU. La estructura E_2 en CPU necesitó 406.38 minutos mientras que en el dispositivo fue necesario apenas 33.47 minutos. Se observó un comportamiento similar en E_3 con 441.38 minutos para la ejecución en el huésped y 33.46 minutos en el dispositivo. En los experimentos E_4, E_5 y E_6 los tiempos registrados fueron 501.39, 586.24 y 609.42 minutos para la versión en CPU y 34.12, 33.59, 34.15 minutos en GPU respectivamente. Con un promedio de **13 veces** más rápida y picos de 17 veces la ejecución en GP-GPU fue notoriamente superior a la versión CPU en cuestiones de tiempos requeridos.

Las pruebas de los modelos $E_1, \ldots, E_6$ en la CPU se observo que los tiempos requeridos fueron superiores en comparación a los de la GP-GPU. Esto se debe al diseño del dispositivo de cálculo masivo, el acceso a los registros de memoria local es casi inmediato. Por otro lado la CPU debe lidiar con los cambios de contextos y los procesos del usuario, además del propio sistema operativo. Para comprender esto simplemente se debe tener en cuenta el volumen de datos a procesar, por ejemplo para la estructura E_6 se requiere aproximadamente unos 3Gb de memoria para alojar las cinco matrices de pesos, los cinco vectores de bias, las matrices para el termino de momento, los datos

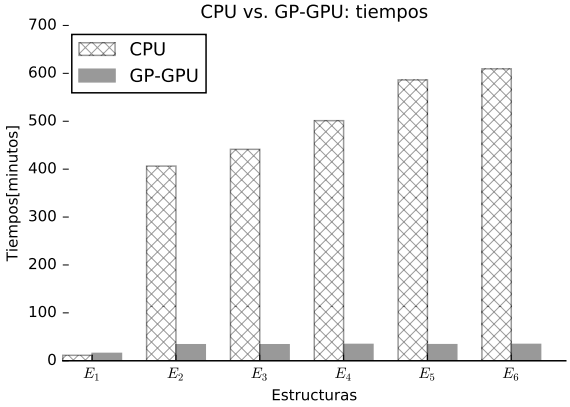


Fig. 4.8 Tiempos de ejecución de diversas DBNs sobre RML Data Base. Comparación de entre las dos arquitecturas, CPU vs GP-GPU. Los tiempos están expresados en minutos totales. Con el dispositivo se alcanzaron tasas de 17 veces más rápido con un promedio de 13 veces.

de entrenamiento y una insignificante cantidad de variables escalares. Luego se debe adicionar la "sobrecarga" que produce las librerías de apoyo como Numpy, el interprete de Python entre otros para manipular los datos y operaciones. A fin de cuentas de forma empírica se observó un incremento de 0.7Gb adicional. De este modo para dicha prueba se requiere un total de 3.7Gb aprox. y por otra parte lo propio del sistema operativo y demás procesos que se estén ejecutando en la PC. El recurso para este proyecto cuenta con un banco de memoria RAM de 4Gb, el sistema Linux instalado requiere 2Gb para su correcto funcionamiento. Al superarse el limite de memoria física disponible se habilita la memoria de intercambio o también llamada *swap*, la cual habilita al sistema a alojar en disco físico la memoria excedente necesaria. Esta última es extremadamente lenta a comparación a la memoria RAM por lo que el impacto en el tiempo de ejecución es notorio a medida que las matrices y datos son cada vez mayores.

Cuando se realizan los experimentos en GP-GPU, todo el conjunto de datos es transferido a la memoria del dispositivo de cómputo, se reserva memoria necesaria para alojar los arreglos, se realizan las operaciones y se transfiere una vez más hacia la CPU los resultados obtenidos. En caso de ser necesario los datos de entrenamiento pueden ser transferidos en porciones si no caben en la memoria de la GP-GPU. Gracias a que este dispositivo está diseñado para el acceso y cómputo masivo, las operaciones se realizan con mayor velocidad. En resumen, si se minimizan las transferencias entre memorias de los dispositivos de hardware, utilizar la GP-GPU minimiza el tiempo requerido. En todos los casos, se observó un rendimiento constante en las pruebas debido a que la cantidad de épocas se mantuvo fijo y sin cambios a lo largo de todas ellas.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Capítulo 5

Conclusiones finales y desarrollos futuros

En este trabajo se diseñó e implementó una biblioteca de algoritmos del campo del aprendizaje automático: *aprendizaje profundo*. La misma cuenta con ciertas funcionalidades, las cuales se pueden resumir en: creación y entrenamiento de redes de creencia profunda; ajuste fino del modelo por medio de algoritmo gradiente descendiente estocástico; creación y entrenamiento de maquinas de Boltzmann restringidas; visualización de filtros y evolución de los modelos; manipulación y procesamiento de datos; algoritmos de búsqueda de hiperparámetros. El despliegue fue realizado por medio de contenedores de virtualización docker, facilitando su instalación y pruebas en cualquier PC con pocos pasos de forma genérica. Tanto la documentación de la biblioteca como el código fuente se encuentran disponibles en la web para el público en general (https.//www.http://cupydle.readthedocs.io y https://github.com/lerker/cupydle respectivamente). Lo que facilita su continuidad en el desarrollo.

Por otro lado, también se evaluó el desempeño de las DBNs en dos tareas de reconocimiento. En primer instancia, con el objetivo de corroborar el correcto funcionamiento de la biblioteca y de verificar el proceso de aprendizaje de las DBNs se realizaron experimentos en la base de datos estándar dentro del campo del aprendizaje automático, reconocimiento de dígitos manuscritos MNIST. En esta etapa se observó el comportamiento predecible en el entrenamiento de los modelos, además de exponer las mejoras en las tasas de acierto alcanzadas de la DBN con respecto a un clasificador estándar. Luego se experimentó con un caso de aplicación real, que pone a prueba los límites del sistema evaluado con la base de datos emoción RML. Fue necesario la manipulación de los datos, aunque tuvo en cuenta la utilización de los mismos casi en su estado puro y así aprovechar las características de las DBNs. En todos los casos se obtuvieron resultados satisfactorios reduciendo tanto el costo computacional como los tiempos de ejecución necesarios en las pruebas.

Como trabajo futuro, en la biblioteca se podría incorporar funcionalidades para ejecutarse en clusters de computadoras que dispongan de GP-GPU. De este modo, es interesante incorporar una capa superior de software que involucre tecnologías relacionadas como ser Apache Spark y NVIDIA GPUDirect. Así lograr maximizar el poder de cómputo y a su vez disminuir los tiempos requeridos. También sería conveniente que se extienda la biblioteca a otros tipos de redes artificiales como ser redes convolucionales o redes recurrentes totalmente conectadas, las cuales también pueden aprovechar los beneficios de la GP-GPU.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Apéndice A

Elección de los recursos a utilizar

A continuación se realiza un breve análisis sobre algunos lenguajes de programación y librerías de comunicación con la GP-GPU. Luego se efectuá la decisión sobre cual seleccionar para el desarrollo del proyecto.

A.1. Lenguaje de programación

La elección del lenguaje de programación no es sencilla, ya que de la misma depende luego la implementación en GP-GPU. Son muchos los lenguajes relacionados a las redes neuronales, y específicamente a aprendizaje profundo encontramos los más populares C++, Python, Matlab, Java entre otros [29, 15].

El lenguaje de programación debe cumplir con ciertos requisitos, especificaciones para que el proyecto se lleve a cabo. La determinación de los mismos están dados por los objetivos del proyecto así como también la justificación del mismo. Algunos de ellos son:

- Debe ser portable, independiente del sistema operativo.
- Ser libre, sin restricciones para su utilización.
- Debe requerir pocas o bien casi ninguna dependencia externa.
- Preferiblemente orientado a objetos (modularización, desagregación).
- Deben contar comunidad de desarrolladores que lo respalde, asegurando el mantenimiento y mejoras futuras.
- Debe contar con una una interfaz de comunicación con la GP-GPU en lo posible.

A continuación se efectuá el análisis sobre los lenguajes C++, Matlab, Python y Java.

- Matlab
  - ***Descripción:*** Es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux.
    Entre sus prestaciones básicas se encuentran: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos interactiva, la creación de interfaces de usuario, la comunicación con otros dispositivos de hardware [1].
  - ***Características:***

[1] Para mas información sobre Matlab visite: http://www.mathworks.com

- Es multiplataforma.
- Matlab es un software propietario de MathWorks, por lo que se requiere una licencia para la utilización de entorno.
- El lenguaje M es interpretado.
- El paradigma que utiliza es orientados a objetos.

- C++
  - ***Descripción:*** Es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C agregando la capacidad de programación orientada a objetos. Es de libre utilización.
    Es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, genérica y orientada a objetos.
    Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos [2].
  - ***Características:***
    - C++ posee el mismo estándar para todas las plataformas o compiladores, lo que lo hace portable.
    - Posee características que permiten la manipulación de todo el entorno, punteros a los objetos en memoria, manipulación bit a bit.
    - Es un lenguaje compilado por lo que la ejecución del binario producido es veloz a comparación a un lenguaje interpretado.
    - Es multi-paradigma, lo que permite mezclar varios tipos de paradigmas en el diseño, entre ellos se encuentran "procedural", "orientado a objetos" y "funcional".
    - Es estrictamente estático, la lógica del programa puede ser optimizada en tiempo de compilación, por lo que lo hace más eficiente.
    - Posee un manejo de memoria determinístico, por lo que la "vida" de un objeto es conocida en todo momento.
    - El lenguaje surgió hace más de tres décadas, es de amplia utilización por lo que lo cuenta con una comunidad extensa.
    - La interfaz de comunicación con la GP-GPU es intuitiva, gracias a que CUDA, del kit de desarrollo de CUDA Nvidia para sus placas gracias posee una sintaxis similar.

- Java
  - ***Descripción:*** Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos. Desarrollado por Sun Microsystems en 1991. Tiene como objetivo ser independiente de la plataforma en la que se ejecuta, ello lo logra a través de una máquina virtual que se instala en la máquina donde reside el programa java y la misma tendrá como propósito interpretar las órdenes del software desarrollado.
    Se inspiró en el lenguaje C++, quitando funcionalidades a bajo nivel (control de puertos I/O, manejo de memoria) dejando un modelo más simple [3].
  - ***Características:***
    - Es multiplataforma, portable.

[2] Información, manuales y detalles de C++ acceda al sitio web: http://www.cprogramming.com/
[3] Sitio oficial de Java: https://www.java.com/es/

- Híbrido, es un lenguaje semi-compilado (bytecode) e interpretado.
- Es necesario que la máquina virtual java se encuentre instalada y en ejecución en la máquina host.
- El paradigma que utiliza es orientado a objetos.
- Abstrae al usuario de la utilización de memoria, delegando al mismo del manejo de la misma.
- La sintaxis deriva de C++, lo que los hace similar aunque menos funcionalidades para el manejo a bajo nivel.
- Es de uso libre, requiere instalar Java "Development Kit".
- Se dispone de una interfaz de comunicación con CUDA, por medio de Jcuda. Se encuentra en desarrollo aun y de escasa documentación [4]

- Python

  - ***Descripción:*** Se trata de un lenguaje de programación multiparadigma, ya que soporta diseño orientado a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.
    Es administrado por la Python Software Foundation . Posee licencia de código abierto. Fue desarrollado en 1991. Tiene entre sus objetivos o directrices que el código desarrollado sea legible y de fácil comprensión, esto lo logra gracias a una directivas llamadas "Index of Python Enhancement Proposals" (PEPs). Las PEPs indican al programador cuando el código es ofuscado o poco legible.
  - ***Características:***
    - Es multiparadigma.
    - Es multiplataforma, lo que lo hace portable.
    - Es un lenguaje interpretado.
    - Es de libre utilización.
    - Diseñado para facilitar la comprensión (PEPs).
    - No posee manejo de memoria por parte del usuario, lo realiza de forma automática Python.
    - Es ampliable mediante otros lenguajes, por ejemplo ejecutar código C++ internamente en Python.
    - La interfaz de comunicación con la GP-GPU es a través de pyCuda, disponible en los repositorios de Python y la misma realiza operaciones similares de invocación que en C++ en CUDA.

A.2. Librería de apoyo

Se debe optar por una librería entre un conjunto de herramientas y bibliotecas, donde las mismas deben tener como objetivo facilitar la programación de los algoritmos para el entrenamiento de redes profundas en la GP-GPU. Dicha elección debe contemplar algunas de las siguientes características,

[4]Sitio oficial de Jcuda: http://www.jcuda.org/
[4]Sitio oficial de Python: https://www.python.org/

- Permitir la interacción entre la CPU y la GP-GPU de forma transparente al usuario. Lo que se busca es que el conocimiento extra sobre cómo opera la GP-GPU sea mínimo para el usuario, y que dicha labor se encargue la librería seleccionada.
- La integración de la misma con el lenguaje de programación debe ser lo más simple posible, esto es, preferentemente desarrollada en el mismo lenguaje y además poseer el menor número de dependencias externas.

Actualmente, existen diversas librerías que implementan algoritmos para redes profundas, algunas de las más populares son pyLearn2, Caffe, Torch, OpenDeep, entre otras. En la tabla A.1 puede encontrarse una síntesis sobre las características de cada librería priorizando los objetivos del proyecto (para una lista más exhaustiva recurrir a [31]). Aún si bien son de libre utilización, presentan un inconveniente, cualquier modificación ya sea en carácter de mejora, expansión de características o bien corrección de errores, queda en completa responsabilidad por parte del usuario y es este último el que debe poseer conocimientos exclusivos del lenguaje de programación en que las librerías fueron desarrolladas, de conocer su diseño y la estructura interna de las mismas. Estas particularidades son intrínsecas de cualquier librería y, si bien pueden pasarse por alto, queda en evidencia cuando el software no cuenta con cierta funcionalidad, tal como lo es para el cumplimiento de los objetivos de este proyecto sobre dichas librerías. El usuario debe codificarlo, lo cual se vuelve una tarea costosa en tiempo y esfuerzo cuando la dimensión del código fuente es alta o la documentación escasa. En general, como estas librerías se encuentran en constante desarrollo, la documentación de las mismas suele ser limitada (también sucede con las que su desarrollo fue abandonado).

A continuación, se realiza análisis sobre las librerías más destacadas en cuanto a su popularidad y su volumen de utilización, además de ser los recursos más generales disponibles hasta el momento [24]. Para una síntesis de la misma, dirigase a la tabla A.1.

Pylern2 Es una librería para redes neuronales desarrollada por el laboratorio LISA de la universidad de Montreal, de uso libre. Con un enfoque a la utilización en el campo de visión computacional, encontrándose relacionado tanto al ámbito académico como investigación. La librería es suficientemente genérica para resolver una amplia cantidad de problemas, se apoya en otra llamada Theano [33] y la cual le brinda funcionalidades para operar eficientemente en GP-GPU. A pesar de todo ello, Pyleran2 posee una estructura compleja, con muchas interfaces que aprender por medio de la experimentación para la utilización. Crear nuevos modelos requieren una sobrecarga mayor de la necesaria. La misma con cuenta con la implementación estándar de RBMs, como así tampoco de las DBNs.

Torch7 Es una librería de computación científica desarrollada en el lenguaje LuaJit. Muchos de los desarrolladores son del Laboratorio del aprendizaje Biologico y Computacional de Nueva York. Torch provee un entorno de scripting, el cual el desarrollo puede ser realizado de forma veloz. Posee características para operar con la GP-GPU. Posee implementaciones de modelos generales, pero no cuenta con desarrollo para las RBMs.

Caffe Es otra librería muy popular para el desarrollo de redes neuronales de aprendizaje profundo. Desarrollada por el Laboratorio de visión computacional de Berkeley, posee funcionalidades que la habilitan fácilmente utilizar la GP-GPU. Aún el núcleo del mismo está escrito en C++, se disponen de muchas interfaces para acceder por medio de otros lenguajes, como Python o Matlab.

A pesar de poseer varias ventajas, Caffe aún se encuentra es intenso desarrollo y no cuenta con implementación de las redes de interés en este proyecto, las redes de creencia profunda.

PyCuda Es una interfaz que habilita la utilización de la API de cómputo paralelo Nvidia CUDA sobre el lenguaje Python. Todo código escrito en Python puede ser traducido a CUDA para luego aprovechas las funcionalidades de la GP-GPU.

Theano Es una librería desarrollada en Python, la cual permite definir, optimizar y evaluar expresiones matemáticas que involucren arreglos multi-dimensionales de manera eficiente. Theano se integra perfectamente a NumPy, es trasparente para la utilización de la GP-GPU, contiene muchas optimizaciones de operaciones, es eficiente en la derivación simbólica, genera código C de forma dinámica entre otras. Es una librería que se encuentra en constante desarrollo y es ampliamente utilizada junto a otras librerías enfocadas a redes neuronales.

A.3. Determinación de las tecnologías a utilizar

La elección del lenguaje de programación y la librería de apoyo para las operaciones en la GP-GPU se realizó de forma conjunta, esto indica que hay una dependencia entre ambas tecnologías. Si bien se determinó el lenguaje de programación en primer instancia, la elección del del mismo indujo la selección de la librería y vice-versa. Como resultado se opto por la dupla Python-Theano.

A continuación se deriva el análisis realizado para arribar a tal conclusión

El lenguaje de programación debe ser de libre utilización, es por ello que el entorno Matlab con su lenguaje M quedó eliminado de la lista final debido a que es un software privativo y requiere licencia. Luego entre los otros candidatos, C++, Java y Python los tres cubren los requerimientos propuestos desde un principio. Continuando Java, el mismo requiere de una máquina virtual para ejecutarse, además por otro lado no se dispone de una interfaz de conexión del lenguaje con la GP-GPU en la actualidad que se encuentre en desarrollo estable, como si disponen de otros lenguajes de forma nativa (CUDA, para C++, PyCuda para Python). Por lo que Java queda descartado.

Por último se analizaron los otros dos candidatos, C++ y Python, como se mencionó con anterioridad ambos cumplen con los requisitos preestablecidos Para la elección se tuvieron en cuenta dos factores, el primero es que C++ requiere mayor manipulación de la memoria (solicitando espacio, liberando espacio, manejos de hilos, etc), si bien esto otorga mayor control sobre el software, al mismo tiempo agrega una complejidad mayor a la hora de codificar los algoritmos, lo que no es un requerimiento para el proyecto. Por otro lado, el siguiente factor, los plazos a cumplir, debido a que Python tiene una filosofía de tipificación de código legible y comprensible, lo que hace que los desarrollos con el lenguaje sean más factibles en períodos cortos de tiempo en contraposición a C++. Por lo que se optó de forma definitiva Python como lenguaje de programación para el proyecto.

Finalmente, una vez definido el lenguaje de programación es necesario establecer un protocolo de comunicación con la GP-GPU, como resultado del proceso de selección fue elegido Theano. Ningunas de las librerías que desarrollan redes profundas cumplen con todos los objetivos propuestos en el informe (ver tabla A.1). Por lo que el desarrollo de la red debe realizarse desde cero, se elige Theano por estar internamente ligado a Python, posee una extensa comunidad de desarrolladores, se encuentra disponible la documentación de forma online [33]. Las operaciones en GP-GPU se realizan de forma transparente para el usuario, agilizando el proceso de desarrollo.

Librería	**Lenguaje Núcleo Librería Auxiliar**	**GP-GPU**	**RBM/DBN**	**Documentación**	**Estado**
Pylearn2	Python/Theano	✓	✓	-	⊖
Torch7	LuaJit	✓	×	✓	⊙
DeepLearnToolbox	Matlab	×	×	×	⊗
Cuda-convenet	C++/CUDA	✓	×	-	⊗
Deepmat	Matlab	×	✓	×	⊗
Caffe	C++	✓	×	✓	⊙
Keras	Python/Theano	✓	×	✓	⊙
Decaf	Python	×	×	-	⊗
OverFeat	C++	×	×	×	⊙
Blocks	Python/Theano	✓	×	-	⊛
Lasagne	Python/Theano	✓	×	×	⊛
MXNet	Python	✓	×	✓	⊙
Deep Belief Networks	Matlab	×	✓	×	⊗
deeplearning4j	Java/Scala	×	×	✓	⊙
OpenDeep	Python/Theano	✓	×	-	⊖
CUPYDLE	Python/Theano	✓	✓	✓	⊛

Tabla A.1 Detalle de las librerías analizadas. En la segunda columna, el lenguaje de programación del núcleo y la librería o lenguaje que la misma utilice de forma auxiliar de la cual depende. En la columna "GP-GPU" se especifica si es que la librería proporciona funcionalidades de del dispositivo o no. En la columna "RBM/DBN" si la misma cuenta con las implementaciones RBM y DBN requeridas para el proyecto. En la quinta columna si cuenta con documentación de sus funciones. En la columna "Estado", denota la situación de desarrollo de la librería, *estable* se encuentra con la mayor parte de sus funcionalidades implementadas con pocos cambios a futuro, *abandonado* el proyecto no tiene desarrolladores asignados, *desarrollo* se continúan agregando funcionalidades nuevas de forma continua y por último *incierto* si el desarrollo estaba abandonado pero muchas de sus funcionalidades estables. CUPYDLE es la biblioteca desarrollada en este proyecto.

SI: ✓ NO:× N/A: - Estable: ⊙ Desarrollo: ⊛ Abandonado: ⊗ Incierto: ⊖

Apéndice B

Anteproyecto

Índice general

1 Justificación

Las redes neuronales artificiales (RNA) constituyen modelos computacionales bioinspirados en las neuronas y funcionamiento del cerebro humano. El objetivo de las redes artificiales es construir sistemas que sean capaces de aprender, generalizar y resolver problemas a los que hoy en día no pueden dar solución los algoritmos convencionales. Las RNA presentan un amplio campo de aplicación, como ser en la economía, medicina, ingeniería, estadística, entre otras. Frecuentemente utilizadas en temas de clasificación, regresión, etc. de grandes volúmenes de datos, las aplicaciones son muy variadas, y van desde por ejemplo la predicción del mercado bursátil a la detección de rostros, optimización de control de vuelo en aeronaves, entre muchos otros [1–3].

En la última década ha surgido una serie de redes neuronales que presentan una nueva arquitectura: redes basadas en el aprendizaje profundo o también llamadas redes profundas. El interés por estas redes se incrementó al observar que al aplicarse a tareas de alta complejidad, como por ejemplo al reconocimiento de la voz [4], se obtienen mejores resultados que con redes tradicionales. Dentro de las redes profundas, podemos encontrar un tipo especial, la denominadas máquinas restringidas de Boltzmann (RBM) y las redes de creencia profunda (DBN), en las cuales se cree que tienen la capacidad de aprender características de los datos de entrada de forma abstracta mediante una representación interna [5], debido a ello la complejidad del problema para decidir cuáles y cómo serán los datos de entrada a la red disminuye para el usuario y, así este ultimo puede abocarse al diseño de la red e interpretación de resultados.

Para el entrenamiento de las redes anteriores, se requiere utilizar técnicas especiales y diferentes a las tradicionales, debido a esto, las mismas presentan un mayor costo computacional tanto en tiempo como de computo. Esto es una característica desalentadora y ha relegado al aprendizaje profundo por mucho tiempo [6, 7].

Por otro lado, con el surgimiento de las Unidades Gráficas de Procesamiento para el Cómputo de Propósito General, o sus siglas en inglés GP-GPU, sus bajos costos y requerimientos de hardware han hecho que las redes profundas puedan ser entrenadas en tiempos razonables [8]. Esto motivó a investigadores a programar sus algoritmos en dichas placas y así poder realizar experimentos con ellas. Actualmente, existen diversas librerías que implementan algoritmos para redes profundas, algunas de ellas son pyLearn2/Theano, Caffe, Torch, entre otras [9–13]. Aun si bien son de libre utilización, presentan un problema, cualquier modificación ya sea en carácter de mejora, expansión de características o bien corrección de errores, queda en completa responsabilidad por parte del usuario y es este último el que debe poseer conocimientos exclusivos del lenguaje de programación en que las librerías fueron desarrolladas, de conocer el diseño y la estructura interna de las mismas. Este problema puede encontrar solución, si se implementa una capa superior, que utilice métodos abstractos para el diseño y entrenamiento de redes neuronales. Con esta idea, el usuario sólo necesita conocer métodos abstractos para el diseño de arquitecturas profundas, para ser transparente la implementación y así poder enfocarse a otros aspectos más relevantes e inherentes al problema y su solución. Actualmente no se cuenta con dicho software que realice tal tarea.

Es por ello que en el presente proyecto se desea generar una librería, útil para implementar redes neuronales de máquinas restringidas de Boltzmann y redes de creencia profunda y, que utilicen la GP-GPU para acelerar su entrenamiento.

Se pondrá a prueba dicha librería en el problema de clasificación de emociones en contenido multimedia [14], realizando el análisis según su desempeño.

El sistema desarrollado será de gran utilidad para la comunidad académica de la Facultad de Ingeniería y Ciencias Hídricas, UNL, investigadores y expertos en el área de la inteligencia computacional podrán realizar sus trabajos valiéndose de esta herramienta para aumentar la productividad, reduciendo los tiempos necesarios y descubrir nuevos horizontes del conocimiento. Todo ello desde un único paquete de software funcional.

2 Objetivos

Generales

- Desarrollar una biblioteca para la creación, entrenamiento y prueba Redes de Creencia Profunda y Máquinas Restringidas de Boltzmann que utilice la GP-GPU.

Específicos

- Relevar características de las librerías pyCUDA, Theano, PyLearn2 y Torch.
- Implementar algoritmos para el entrenamiento y prueba de Máquinas Restringidas de Boltzmann y Redes de Creencia Profunda.
- Implementar en GP-GPU los algoritmos para el entrenamiento de RBM y DBN.
- Evaluar el software aplicado al reconocimiento de emociones.
- Analizar el sistema con respecto a otras arquitecturas de redes tradicionales.
- Realizar el informe final y documentación del sistema.

3 Alcances

En este proyecto se desarrolla un conjunto de algoritmos valiéndose de técnicas de aprendizaje profundo utilizando el cómputo paralelo de las unidades gráficas de propósito general. Los principales usuarios al cual están dirigidos pertenecen a la comunidad académica de la Universidad Nacional del Litoral.

Requerimientos no funcionales

- Los algoritmos deben ser robustos y veloces.
- El software final debe ser portable.

Limitaciones y Restricciones

- Utilizar la placa de procesamiento gráfico de propósito general.
- Utilizar librerías Cuda/pyCuda.
- Sólo se implementara una estrategia de entrenamiento de arquitecturas profundas.
- Sólo se implementan los algoritmos de entrenamiento y prueba para las RBM y DBN.

- No se proporciona interfaz gráfica.

Supuestos

- Placa Gráfica funcional, instalada en una PC. La misma debe ser capaz de procesar lenguaje C o Python, provisto por el controlador de dicha placa.
- El sistema operativo no debe tener limitaciones para instalar el conjunto de librerías necesarias, tales como pyCuda, Theano y pyLearn2, Torch, etc. y sus respectivas dependencias.

Requisitos

- Disponer del corpus de datos para experimentación, entrenamiento y prueba, así como también resultados sobre de los mismos utilizando otras estrategias de inteligencia artificial.

4 Metodología

Se comenzará con una etapa inicial en la cual se realizará una investigación bibliográfica, enfocada en el estudio de la problemática en cuestión y en técnicas de aprendizaje profundo en general, luego se profundizará con temas relacionados a Maquinas Restringidas de Boltzmann, Redes de Creencia Profunda y técnicas para implementar los algoritmos en GP-GPU. Lo siguiente será realizar un análisis de las librerías y paquetes de software disponibles que apliquen técnicas de aprendizaje profundo, evaluando la posibilidad de aplicarlos al proyecto a este proyecto.

Luego se diseñará e implementará el conjunto de algoritmos para el entrenamiento y prueba de las redes, se diseñarán tanto para la ejecución en CPU y para la ejecución utilizando técnicas paralelas en GP-GPU. Cada conjunto de algoritmos serán evaluados con casos de pruebas de laboratorio, para verificar su correcto funcionamiento.

El siguiente paso será analizar el caso de aplicación, reconocimiento de emociones, sobre el sistema desarrollado, evaluando así los resultados obtenidos frente a otras estrategias no paralelas. Finalmente, se procederá a la documentación del sistema y desarrollo del informe Final.

El desarrollo del software resultante se encuadra dentro del modelo en cascada. Cada proceso es secuencial y la finalización de uno de ellos es el punto de inicio del siguiente.

La metodología, por lo tanto está compuesta por las etapas de trabajo que se detallan a continuación. Al finalizar cada etapa se presentará un hito, que será resultado de la etapa, el cual estará sujeto a determinados criterios de aceptación. Para cada una de las etapas los directores del proyecto serán quienes juzguen si los resultados obtenidos satisfacen dichos criterios.

5 Plan de Tareas

El proyecto contará con una dedicación diaria de tres horas para la primera etapa y de cuatro horas diarias para la segunda etapa, de lunes a viernes. La duración total del mismo

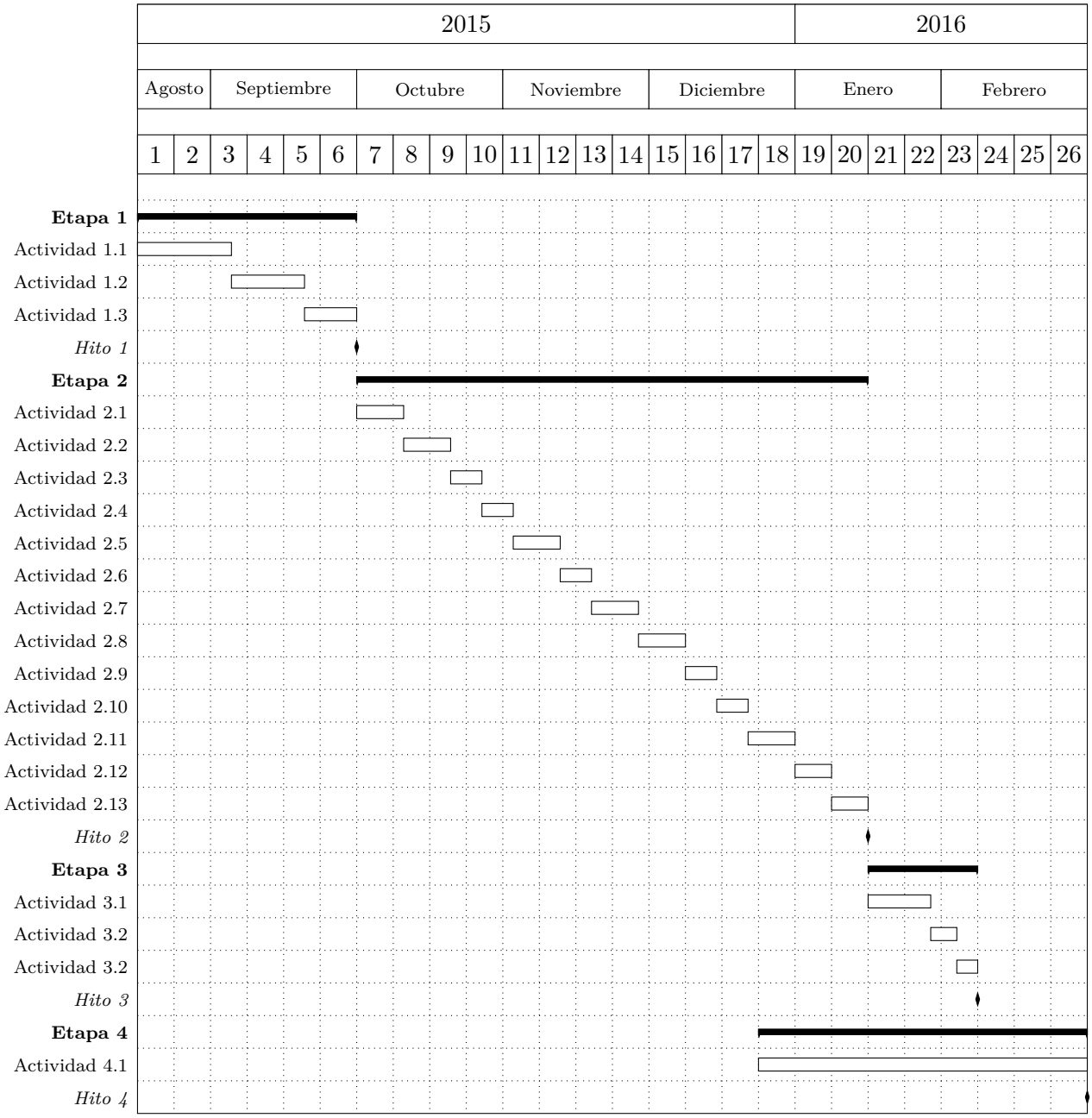
será de veintiséis semanas. Se dará inicio con las actividades el día 17 de Agosto del corriente año, y finalizará tentativamente, el día 29 de Febrero del 2016.

El proyecto cuenta con un solo recurso humano, la ejecución de las etapas se harán de forma secuenciales, a excepción de la cuarta etapa, *Documentación*, que se realizará de forma simultánea junto a la actividad 2.11, al inicio de la semana décimo octava de ejecución del proyecto, debido a ello que se adicionará una hora diaria de trabajo a partir de dicha semana, siendo así cinco horas diarias de trabajo para completar las tareas hasta la finalización del mismo.

1. *Búsqueda bibliográfica. [80 horas]*
   - 1.1. Estudio bibliográfico sobre las RBM y DBN. [40 horas]
   - 1.2. Estudio bibliográfico sobre algoritmos en GP-GPU. [30 horas]
   - 1.3. Redacción del informe de avance 1. [10 horas]
2. *Diseño, implementación y evaluación del sistema profundo. [320 horas]*
   - 2.1. Diseño de los algoritmos RBM iterativo.[30 horas]
   - 2.2. Diseño de los algoritmos DBN iterativo.[30 horas]
   - 2.3. Implementación de los algoritmos de la actividad 2.1.[20 horas]
   - 2.4. Implementación de los algoritmos de la actividad 2.2.[20 horas]
   - 2.5. Prueba de los algoritmos de las actividades 2.3 y 2.4.[30 horas]
   - 2.6. Correcciones de los algoritmos de la actividad 2.3 y 2.4.[20 horas]
   - 2.7. Diseño de los algoritmos RBM en GP-GPU.[30 horas]
   - 2.8. Diseño de los algoritmos DBN en GP-GPU.[30 horas]
   - 2.9. Implementación de los algoritmos de la actividad 2.7.[20 horas]
   - 2.10. Implementación de los algoritmos de la actividad 2.8.[20 horas]
   - 2.11. Prueba de los algoritmos de las actividades 2.9 y 2.10.[30 horas]
   - 2.12. Correcciones de los algoritmos de la actividad 2.9 y 2.10.[20 horas]
   - 2.13. Redacción del informe de avance 2.[20 horas]
3. *Análisis al caso de reconocimiento de emociones. [60 horas]*
   - 3.1. Prueba del sistema con el caso reconocimiento de emociones.[30 horas]
   - 3.2. Análisis de los resultados obtenidos en la actividad 3.1.[20 horas]
   - 3.3. Redacción del informe de avance 3.[10 horas]
4. *Documentación. [90 horas]*
   - 4.1. Documentación del Sistema y redacción del Informe Final.[90 horas]

6 Cronograma

A continuación se presenta el diagrama de Gantt de las actividades involucradas en el proyecto.



7 Puntos de Control, Entregables y Criterios de aceptación

En el desarrollo del proyecto, se fijan un conjunto de hitos donde se presentarán informes de avances, cada *hito* será considerado como un punto de control individual. A continuación se detalla la lista de los mismos, junto a las etapas que abarca cada uno y la fecha aproximada de realización.

Etapa 1: Búsqueda bibliográfica – Semanas 1-6

En esta etapa se estudiará el estado del arte, es decir, la bibliografía existente sobre la temática. El estudio estará enfocado, en primer lugar a comprender el problema en cuestión, a saber, algoritmos de entrenamiento para las RBM, DBN y cómo implementarlos en GP-GPU. En segundo lugar, se realizará un estudio las librerías y paquetes de software existentes que en la práctica utilicen algunos de los algoritmos de entrenamiento, y se prestará atención a aquellos que estén codificados en python o C++ y/o utilicen CUDA/pyCuda.

Entregable 1:

Documento con toda la información recopilada durante la etapa de búsqueda. Se incluirá un resumen del tema en general, el estado del arte y las herramientas o librerías a utilizar para desarrollar el proyecto. Se propone realizar un an análisis comparativo sobre la elección de una u otra biblioteca para el desarrollo del software, exponiendo las razones correspondientes.

Hito 1:

Fecha: 30/09/15
Logro de la comprensión adecuada del problema, de los algoritmos de entrenamiento y elección de la librería.

Criterios de aceptación:

La etapa se considerará finalizada satisfactoriamente si la bibliografía consultada ha sido suficientemente abarcativa y seleccionada la librería.

Etapa 2: Diseño y evaluación del sistema profundo – Semanas 7-20

En esta etapa, por un lado, se diseñará el conjunto de algoritmos tanto para el entrenamiento como prueba de las redes profundas de forma iterativa. Luego se procederá a la implementación de los mismos. Por otro lado, se realizará el diseño e implementación de los algoritmos para ejecución en GP-GPU. Para la implementación, se utilizarán las librerías escogidas previamente de la Etapa 1. Se evaluará la correcta ejecución de cada paquete de software con casos de pruebas modelos.

Entregable 2:

Como salida de la etapa se desea obtener un conjunto de algoritmos, el diseño e implementación de los mismo tanto de forma iterativa como paralela en GP-GPU.

Hito 2:

Fecha: 15/01/16
Sistema de aprendizaje profundo y validación del mismo.

Criterios de aceptación:

La etapa se considerará finalizada satisfactoriamente si el sistema una vez implementado supera las evaluaciones con el caso de prueba con márgenes de errores aceptables.

Etapa 3: Análisis del caso reconocimiento de emociones – Semanas 21-23

En esta etapa se analizará el desempeño del sistema aplicado al reconocimiento de emociones. Se evaluará el software frente a otras estrategias que resuelven el problema de forma iterativa. Se probarán diferentes parámetros del sistema y recopilar información necesaria para su evaluación.

Entregable 3:

Al finalizar la etapa 3, se generá un informe de avance detallando características aspectos relevantes del sistema desarrollado frente a otras soluciones no paralelas para resolver el caso de aplicación.

Hito 3:

Fecha: 05/02/16
Análisis caso de aplicación frente a soluciones iterativas.

Criterios de aceptación:

La etapa será considerada como finalizada satisfactoriamente cuando el análisis del sistema frente al caso de aplicación finalice y, informando los resultados obtenidos.

Etapa 4: Documentación – Semanas 18-26

En esta etapa se procederá a la documentación del sistema, generando un documento detallando las funcionalidades del sistema. Además se elaborará el informe final del proyecto.

Entregable 4:

Se generá un documento que detalla las funcionalidades del sistema. Ademas se elabora el documento final del proyecto.

Hito 4:

Fecha: 29/02/16
Documentación del sistema e informe Final del proyecto.

Criterios de aceptación:

La etapa se considera satisfactoriamente finalizada si se ha logrado documentar el sistema y, concluir con la totalidad el informe final.

8 Informes de Avance

Con propósito de seguimiento y control del proyecto, se generan informes de avance. En los mismos se detallarán aspectos relevantes del mismo, así como también el progreso y estado del proyecto al momento de generar el informe, contrastando el cronograma de las fechas estimadas para las actividades y el estado real de las mismas.

Se planean realizar tres informes de avance, cada uno al finalizar las etapas uno, dos y tres del proyecto. A continuación se detallan los mismos con sus respectivas fechas estimadas de presentación:

- Primer informe de avance: 06/10/15
  Este documento contendrá la información recopilada durante la etapa de búsqueda. Se incluirá un resumen del tema en general y las herramientas o librerías a utilizar para desarrollar el proyecto.
- Segundo informe de avance: 09/02/16
  El informe detallará el conjunto de algoritmos, el diseño e implementación de los mismo tanto de forma iterativa como paralela en GP-GPU, los mismos obtenidos en la salida de la etapa dos del proyecto.
- Tercer informe de avance: 16/02/16
  El documento, expondrá las características y aspectos relevantes del sistema desarrollado frente a otras soluciones no paralelas para resolver el caso de aplicación. Al final del mismo se desarrollará una breve conclusión en forma de síntesis de los problemas abordados a lo largo del proyecto.

9 Riesgos

El análisis de riesgos implicados en el proyecto se llevó a cabo realizando un análisis sobre la probabilidad de ocurrencia y el impacto que el mismo ocasiona si el riesgo sucede durante la ejecución del proyecto. También se han determinado estrategias de acciones para abordar los riesgos.

Para determinar la severidad de los riegos y selección de estrategia a seguir, se realizó el producto entre impacto y la probabilidad estimada para los mismo según las siguientes tablas:

Probabilidad	
Bajo	0.00 - 0.30
Medio	0.30 - 0.60
Alto	0.60 - 0.90

Impacto	
Bajo	0.00 - 0.30
Medio	0.30 - 0.60
Alto	0.60 - 0.90

Estrategias	
Bajo	Aceptar
Medio	Mitigar
Alto	Evitar

La matriz de severidad a la que ya se le han asociado las estrategias queda definida entonces como:

Matriz de Severidad				
		Impacto		
		Bajo	Medio	Alto
Prob.	Bajo	Aceptar	Aceptar	Mitigar
	Medio	Aceptar	Mitigar	Mitigar
	Alto	Mitigar	Mitigar	Evitar

A continuación se detallan los riegos relevados, a los cuales se han clasificado a priori en la etapa del cronograma posible de ocurrencia de los mismos. Se determinó un nombre identificativo-titulo, indicador de la presencia del riesgo, su probabilidad e impacto, severidad, la estrategia adoptada (en el caso de que la misma sea mitigar, se enfocará a reducir la probabilidad de ocurrencia del riesgo en cuestión) y una estrategia de contingencia en caso de que el riesgo suceda, abocada a reducir el impacto del mismo.

Etapa 1:

Riesgo 001: Falta de fiabilidad del material bibliográfico.

- *Descripción:* a causa de la novedad del problema bajo estudio, puede que el material consultado de sitios web como parte de recursos bibliográficos sean inconsistentes o presenten errores.
- *Indicador:* Aseveraciones por el autor sin justificaciones. Desarrollos sin pruebas o poco sustento teórico.
- *Probabilidad:* media.
- *Impacto:* alto.
- *Severidad:* media.
- *Estrategia:* mitigar. Utilizar sitios web conocidos o respaldados por alguna entidad confiable. Recurrir al juicio del director o co-director si se presentan dudas respecto a la bibliografía consultada.
- *Estrategia de contingencia:* En el caso que el material bibliográfico inconsistente o poco fiable haya sido incorporado al cuerpo de conocimiento del proyecto, esto se detectará a partir de la etapa 2 del mismo. Se continuará el desarrollo de la etapa y consultará a los directores sobre como proseguir.

Etapa 2:

Riesgo 002: Dificultades en el diseño e implementación de los algoritmos.

- *Descripción:* los algoritmos paralelos para redes profundas son complejos de diseñar e implementar. Realizar pruebas y validaciones de los mismos requieren un gran esfuerzo y tiempo.
- *Indicador:* Se identificará la presencia del riesgo si en la etapa dos del proyecto no se cumplen con el cronograma estimado.
- *Probabilidad:* media.
- *Impacto:* alto.
- *Severidad:* media.
- *Estrategia:* mitigar. Se consultará con personas experimentadas de forma progresiva, disminuyendo posibles inconsistencias.
- *Estrategia de contingencia:* En el caso de que alguna de las tareas retrase en la ejecución de alguna otra de forma pronunciada, se avanzará con otra actividad disponible en el momento, además se consultará a los directores sobre el problema específico.

Riesgo 003: Falta de información para poder determinar el método de paralelización adecuado.

- *Descripción:* aunque se cuente con información sobre como paralelizar los algoritmos, puede que haya varias formas de implementarlos. Además ciertos métodos no tienen justificación o sustento teóricos que lo validen.
- *Indicador:* Se dispone de varias formas de implementar un algoritmo.
- *Probabilidad:* bajo.
- *Impacto:* medio.
- *Severidad:* media.
- *Estrategia:* Aceptar.
- *Estrategia de contingencia:* En caso de que suceda, se analizará las opciones o se recurrirá al juicio de los directores.

Riesgo 004: Carencia de funcionalidades de la librería seleccionada.

- *Descripción:* En el proceso de selección de la librería no se detecto la necesidad de alguna funcionalidad en la librería, la cual no se encuentra implementada.
- *Indicador:* No se dispone de dicha funcionalidad.
- *Probabilidad:* baja.
- *Impacto:* medio.
- *Severidad:* baja.
- *Estrategia:* Aceptar.
- *Estrategia de contingencia:* Diseñar e implementar la funcionalidad requerida.

Etapa 2-3:

Riesgo 005: Falla en el acceso o disponibilidad de la GP-GPU

- *Descripción:* podrían presentarse problemas de comunicación entre el sistema operativo y la GP-GPU debido a la mala configuración del controlador de la placa gráfica y/o actualización del mismo. La misma podría estar siendo utilizada por otro usuario.
- *Indicador:* No se puede ejecutar los algoritmos paralelos. Imposibilidad de establecer la comunicación.
- *Probabilidad:* bajo.
- *Impacto:* medio.
- *Severidad:* media.
- *Estrategia:* Aceptar.
- *Estrategia de contingencia:* En caso de poder acceder a la utilización de la GP-GPU esperar un lapso de tiempo a que la misma se desocupe. Actualizar el sistema para posible solución. Avanzar con otras actividades del cronograma si es posible.

10 Recursos necesarios y disponibles

Durante la ejecución del proyecto, se requerirán los recursos que se detallan a continuación.

- *Disponible:* Notebook, se dispone con una notebook Dell Xps L502 de cinco años de antigüedad propiedad del estudiante.
- *Disponible:* GPGPU: se cuenta con el acceso a una computadora con placa GP-GPU instalada, perteneciente al "*Research Institute for Signals, Systems and Computational Intelligence*" «sinc(i)».
- *Disponible:* Bibliografía: Se dispone para su utilización de la biblioteca "Dr. Enzio Emiliani". Desde el acceso a internet por medio de la red de la FICH se cuenta con la posibilidad de acceder a cientos de publicaciones, papers, revistas científicas, etc. alojadas en la Web de forma gratuita.
- *Pendiente:* Librerías: aún no se encuentra seleccionada, la misma se definirá en la primer etapa de ejecución del proyecto.
- *Disponible:* Servicios: como ser internet, electricidad, agua. Están disponibles.

11 Presupuesto

A continuación se detalla el presupuesto estimado necesario para la ejecución del proyecto. Para aquellos recursos disponibles se realizará la amortización corespondiente, según la siguiente formula:

Amortización $= \frac{VN-VR}{VU} \cdot HU$

donde:

- *VN:* Valor a nuevo del bien de uso.
- *VR:* Valor residual.
- *VU:* Vida útil estimada en horas.
- *HU:* Horas de uso requerido.

En la siguiente tabla se detallan los costos del proyecto, clasificados por *tipo*: directo o indirecto, *concepto*: gasto o amortización, *razón*: nombre y el *detalle*: del mismo.

El presupuesto total requerido por el proyecto es de : $99.262 (*pesos noventa y nueve mil docientos sesenta y dos*).

Razón	Detalle	Concepto	Tipo	**TOTAL**
Bienes de Capital				
Notebook	Laptop Dell XPS L502x Valor Nuevo: $17.000 Valor Residual: $800 Vida Util: 10000 horas. Amortización: $800	Amortización	Costo directo	$890
PC(GP-GPU)	PC con una GP-GPU instalada, sinc	Amortización	Costo indirecto	$930
Materiales e Insumos				
Impresiones	informes de avance preinforme e informe final fotocopias	Gasto	Costo directo	$1.230
Viajes y Viáticos				
Trasporte	Boleto colectivo urbano por dia de trabajo: $10	Gasto	Costo directo	$1.300
Almuerzo	Vianda por dia de trabajo: $25	Gasto	Costo directo	$1.375
Otros	Cafe, yerba, varios por mes de trabajo $150	Gasto	Costo directo	$975
Recursos Humanos				
Estudiante	Remuneración por hora $90	Gasto	Costo directo	$49.500
Director	Remuneracion por hora $290 Cant. de horas requeridas 80	Gasto	Costo directo	$23.200
Co-Director	Remuneración por hora $290 Cant. de horas requeridas 60	Gasto	Costo directo	$17.400
Otros Costos				
Servicio de Agua	Costo fijo mensual servicio de agua $100	Gasto	Costo indirecto	$300
Servicio de Electricidad	Consumo: Notebook 100W/h PC 1000W/h Energia estimada requerida: 415.30kW Costo kW/h: $0.73	Gasto	Costo indirecto	$412
Servicio de internet	Valor mensual $250	Gasto	Costo directo	$1.750
COSTO TOTAL				$99.262

Referencias

[1] F. Villada *et al.*, "Aplicación de las redes neuronales al pronóstico de precios en el mercado de valores," *Información Tecnológica*, vol. 23, no. 4, pp. 11–22.

[2] G. Wyeth, G. Buskey, and J. Roberts, "Flight control using an artificial neural network," in *Proceedings of the Australian Conference on Robotics and Automation (ACRA 2000), August*, 2000.

[3] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 1, pp. 23–38, 1998.

[4] G. Hinton *et al.*, "Speech recognition with deep recurrent neural networks," *IEEE International Conference on Acoustic Speech and Signal Processing*.

[5] S. Ruslan *et al.*, "Restricted boltzmann machines for collaborative filtering," *Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR*, 2007.

[6] G. Hinton *et al.*, "A fast learning algorithm for deep belief nets," *Neural Computation*, 2006.

[7] Nvidia, "Machine learning, link: http://www.nvidia.com/object/machine-learning.html, consultado el 11/06/2015."

[8] J. Bergstra *et al.*, "Theano: A cpu and gpu math compiler in python," *Proc. of the 9th Python in Science conffference*.

[9] Theano, "Software, link: http://deeplearning.net/software/theano/, consultado el 02/03/2015."

[10] pyLearn2, "Software, link: http://deeplearning.net/software/pylearn2/, consultado el 11/06/2015."

[11] Torch, "Software, link: http://torch.ch/, consultado el 02/03/2015."

[12] Caffe, "Software, link: http://caffe.berkeleyvision.org/, consultado el 11/06/2015."

[13] Software, "Software, link: http://deeplearning.net/softwarelinks/, consultado el 11/06/2015."

[14] R. Alvarez *et al.*, *Reconocimiento automático de emociones en contenido multimedia*. 2014.

[15] H. G. E., "Sitio web oficial, link: https://www.cs.toronto.edu/ñinton/, consultado el 11/062015."

[16] P. M. Institute, *Pmbok: Guía de los fundamentos de gestión de proyectos*. 2013.

[17] N. Matloff, *Programming on Parallel Machines: GPU, Multicore, Clusters and More*. 2009.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Apéndice C

Documentacion

Tabla de Contenidos

cupydle package

1.1 Subpackages

1.1.1 cupydle.dnn package

Submodules

cupydle.dnn.capas module

cupydle.dnn.dbn module

cupydle.dnn.funciones module

cupydle.dnn.graficos module

cupydle.dnn.gridSearch module

class cupydle.dnn.gridSearch.**ParameterGrid**(*param_grid*)

Clases base: object

Busqueda de parametros en una grilla de un numero discreto de los mismos. Puede ser utilizado para iterar sobre la combinacion de los parametros con funciones internas de Python.

Parámetros **param_grid** (*dict*) – Grilla de parametros a explorar.

Examples

```
>>> from cupydle.dnn.grid_search import ParameterGrid
>>> param_grid = {'a': [1, 2], 'b': [True, False]}
>>> list(ParameterGrid(param_grid)) == (
...     [{'a': 1, 'b': True}, {'a': 1, 'b': False},
...      {'a': 2, 'b': True}, {'a': 2, 'b': False}])
True
>>> grid = [{'kernel': ['linear']}, {'kernel': ['rbf'], 'gamma': [1,␣
↪10]}]
>>> list(ParameterGrid(grid)) == [{'kernel': 'linear'},
...                               {'kernel': 'rbf', 'gamma': 1},
...                               {'kernel': 'rbf', 'gamma': 10}]
True
```

```
>>> ParameterGrid(grid)[1] == {'kernel': 'rbf', 'gamma': 1}
True
```

__getitem__(*ind*)
Obtine un punto de la grilla :param ind: indice :type ind: int

Devuelve params – igual a una lista

Tipo del valor devuelto dict

__iter__()
Itera sobre puntos de la grilla

Devuelve params

Tipo del valor devuelto iterador

__len__()
Numero de puntos de la grilla

cupydle.dnn.loss module

cupydle.dnn.mlp module

cupydle.dnn.prueba module

cupydle.dnn.rbm module

cupydle.dnn.stops module

criterios de paradas para el entrenamiento..

class cupydle.dnn.stops.**Patience**(*initial*, *key='hits'*, *grow_factor=1.0*, *grow_offset=0.0*, *threshold=0.0001*)
Clases base: object

Criterio de parada inspirado en el metodo de Bengio. La idea es incrementar el numero de iteraciones por un factor multiplicativo mientras la red obtiene mejores candidatos. Caso opuesto no aumenta.

Parámetros

- **func_or_key** (*funcion*) – Funcion que itera sobre un objeto.
- **initial** (*int*) – entero
- **grow_factor** (*float*) – factor de crecimiento
- **grow_offset** (*float*) –
- **threshold** (*float[Opcional]*) –

class cupydle.dnn.stops.**controlDelTeclado**
Clases base: object

killer = controlDelTeclado() while True:

time.sleep(1) print("doing something in a loop ...") if killer():

print("matando") break

print("End of the program. I was killed gracefully :)")

matar (*signum*, *frame*)

class cupydle.dnn.stops.**iteracionesMaximas** (*maxIter*)
Clases base: object

class cupydle.dnn.stops.**noMejorQueAntesError**
Clases base: object

class cupydle.dnn.stops.**tiempoTranscurrido** (*tiempoMaximo*)
Clases base: object

class cupydle.dnn.stops.**toleranciaError** (*tolerancia*)
Clases base: object

cupydle.dnn.unidades module

cupydle.dnn.utils module

class cupydle.dnn.utils.**RestrictedDict** (*allowed_keys*, *seq=()*, ***kwargs*)
Clases base: dict

Es un diccionario, cualquier intento de almacenar una clave ya existente se arroja un error.

Se inicializa con una instancia, y las claves permitidas que no pueden ser modificadas.

Parámetros **allowed_keys** (*tuple*) – claves permidas

Examples

```
>>> p = RestrictedDict(('x','y'))
>>> print p
RestrictedDict(('x', 'y'), {})
>>> p['x'] = 1
>>> p['y'] = 'item'
>>> print p
RestrictedDict(('x', 'y'), {'y': 'item', 'x': 1})
>>> p.update({'x': 2, 'y': 5})
>>> print p
RestrictedDict(('x', 'y'), {'y': 5, 'x': 2})
>>> p['x']
2
>>> p['z'] = 0
Traceback (most recent call last):
...
KeyError: 'z is not allowed as key'
>>> q = RestrictedDict(('x', 'y'), x=2, y=5)
>>> p==q
True
>>> q = RestrictedDict(('x', 'y', 'z'), x=2, y=5)
>>> p==q
False
>>> len(q)
```

```
2
>>> q.keys()
['y', 'x']
>>> q._allowed_keys
('x', 'y', 'z')
>>> p._allowed_keys = ('x', 'y', 'z')
>>> p['z'] = 3
>>> print p
RestrictedDict(('x', 'y', 'z'), {'y': 5, 'x': 2, 'z': 3})
```

__eq__(*other*)
Dos diccionarios son equivalentes si sus claves lo son

__ne__(*y*) <==> *not x.__eq__(y)*

__repr__()
Representacion por consola

__setitem__(*key*, *value*)
Chequea si la clave es una valida con los valores seteados

update(*e=None*, ***kwargs*)
Actualiza el diccionario

cupydle.dnn.utils.**blanqueo**(*datos*, *eigenvalues=100*, *epison=1e-05*)

cupydle.dnn.utils.**cargarHDF5**(*nombreArchivo*, *clave*)
Recupera de disco el archivo almacenado con *guardarHDF5()*

La estructura debe almacenarse previamente.

Parámetros

- **nombreArchivo** (*str*) – ruta + nombre del archivo a cargar los datos.
- **clave** (*str, list(str)*) – clave del objeto a cargar. Debe existir previamente. Si es una lista re recuperan los objetos. Si es *None* se recuperan todos los objetos en un diccionario.

Devuelve Diccionario si clave=None o si clave=list(...). numpy.ndarray si clave=str

Tipo del valor devuelto dict, numpy.ndarray

Ver también:

La documentacion oficial de h5py.

Ademas el la funcion *guardarHDF5()*.

Examples

```
>>> import numpy as np
>>> datos = {'nombre': 'mlp', 'valor': 0.0, 'pesos': []}
>>> a = np.asarray([[7,8,9],[10,11,12]])
>>> guardarHDF5(nombreArchivo="prueba.cupdyle", valor=datos,␣
↪nuevo=True)
>>> guardarHDF5("prueba.cupdyle",{'pesos':a})
>>> guardarHDF5("prueba.cupdyle",{'valor':10.0, 'nombre':'mlp2'})
```

```
>>> cargarHDF5("prueba.cupydle",'pesos')
[array([7, 8, 9]), array([7, 8, 9]), array([ 8,  9, 10])]
>>> cargarHDF5("prueba.cupydle",['nombre', 'pesos'])
{'nombre': 'mlp',
'pesos': [array([7, 8, 9]), array([7, 8, 9]), array([ 8,  9, 10])]}
>>> cargarHDF5("prueba.cupydle",None)
{'nombre': 'mlp2',
'valor': 10.0,
'pesos': [array([7, 8, 9]), array([7, 8, 9]), array([ 8,  9, 10])]}
```

cupydle.dnn.utils.**cargarSHELVE**(*nombreArchivo*, *clave*)

Metodo para recuperar los datos almacenados por medio de *shelve*

Es menos eficiente que HDF5 con respecto a la memoria requerida, pero mas simple de posterior lectura.

Parámetros

- **nombreArchivo** (*str*) – ruta + nombre del archivo a cargar los datos.
- **clave** (*str, list(str)*) – clave del objeto a cargar. Debe existir previamente. Si es una lista re recuperan los objetos. Si es *None* se recuperan todos los objetos en un diccionario.

Devuelve Diccionario si clave=None o si clave=list(...). numpy.ndarray si clave=str

Tipo del valor devuelto dict, numpy.ndarray

Ver también:

La documentacion oficial de shelve.

Ademas el la funcion *guardarSHELVE()*.

Examples

```
>>> import numpy as np
>>> datos = {'nombre': 'mlp', 'valor': 0.0, 'pesos': []}
>>> a = np.asarray([[7,8,9],[10,11,12]])
>>> guardarSHELVE(nombreArchivo="prueba.cupdyle", valor=datos,␣
↪nuevo=True)
>>> datos
{'nombre': 'mlp',
'valor': 0.0,
'pesos': []}
>>> guardarSHELVE("prueba.cupdyle",{'pesos':a})
>>> guardarSHELVE("prueba.cupdyle",{'valor':10.0, 'nombre':'mlp2'})
>>> cargarSHELVE("prueba.cupdyle",None)
{'nombre': 'mlp2',
'valor': 10.0,
'pesos': [array([7, 8, 9]), array([8, 9, 10])]}
```

cupydle.dnn.utils.**check_consistent_length**(**arrays*)

Chequea que los array sean cosistentes en sus dimensiones (primera)

Parámetros ***arrays** (*list*) – list o tuplas de objetos

cupydle.dnn.utils.**check_random_state**(*seed*)

cupydle.dnn.utils.**downCast**(*objeto*)

cupydle.dnn.utils.**guardarHDF5**(*nombreArchivo*, *valor*, *nuevo=False*)

Metodo de almacenamiento por medio de HDF5 (h5py)

Es mas eficiente con respecto a pickle en cuanto memoria requerida.

Internamente el modulo almacena los datos como una estructura de ficheros Todo se puede almacenar como *datasets*, lo cual es semejante a *numpy.ndarrays* Pueden agruparse datasets en grupos. Para objetos pequenios, en vez de datasets pueden almacenarse atributos del dataset

Referencia:

- / : base
- /miDataset : dataset
- /miGrupo : grupo
- /miGrupo/miDataset : dataset dentro del grupo
- /miDataset.attrb['miAtributo']

Los dataset son para almacenar objetos grandes (ej: numpy.ndarray, listas) Los atributos estan asociados a un dataset dado, y son mas eficientes para objetos mas pequeños (escalares y strings)

Modos de apertura de los archivos:

- r Solo lectura, el archivo debe existir.
- r+ Lectura y escritura, el archivo debe existir.
- w Crea el archivo, sobre-escribe si existe.
- x Crea el archivo, no sobrescribe.
- a Lectura y Escritura, Crea y sobreescribe.

Nota: El argumento *nuevo* debe utilizarse por unica vez, persistiendo la estructura de los datos, flotantes, listas, etc.

Parámetros

- **nombreArchivo** (*str*) – ruta + nombre del archivo a persistir los datos.
- **valor** (*dict*) – datos a almacenar contenidos en un diccionario.
- **nuevo** (*Opcional[bool]*) – Si es True, crea el archivo (sobreescribe). Utilizarlo para crear la estructura inicial.

Ver también:

La documentacion oficial de h5py.

Ademas el la funcion *cargarHDF5()*.

Examples

```
>>> import numpy as np
>>> datos = {'nombre': 'mlp', 'valor': 0.0, 'pesos': []}
>>> a = np.asarray([[7,8,9],[10,11,12]])
>>> guardarHDF5(nombreArchivo="prueba.cupdyle", valor=datos,␣
↪nuevo=True)
>>> datos
{'nombre': 'mlp',
'valor': 0.0,
'pesos': []}
>>> guardarHDF5("prueba.cupdyle",{'pesos':a})
>>> guardarHDF5("prueba.cupdyle",{'valor':10.0, 'nombre':'mlp2'})
>>> cargarHDF5("prueba.cupdyle",None)
{'nombre': 'mlp2',
'valor': 10.0,
'pesos': [array([7, 8, 9]), array([8, 9, 10])]}
```

cupydle.dnn.utils.**guardarSHELVE**(*nombreArchivo*, *valor*, *nuevo=False*)

Metodo de almacenamiento por medio de SHELVE (pickles)

Es menos eficiente que HDF5 con respecto a la memoria requerida, pero mas simple de posterior lectura

Internamente almacena los datos como *pickles* individuales.

Modos de apertura de los archivos:

- r Solo lectura, el archivo debe existir.
- w Lectura y escritura, el archivo debe existir.
- c Lectura y escritura, crea si no existe.
- n Lectura y Escritura, Crea y sobreescribe.

Nota: El argumento *nuevo* debe utilizarse por unica vez, persistiendo la estructura de los datos, flotantes, listas, etc.

Parámetros

- **nombreArchivo** (*str*) – ruta + nombre del archivo a persistir los datos.
- **valor** (*dict*) – datos a almacenar contenidos en un diccionario.
- **nuevo** (*Opcional[bool]*) – Si es True, crea el archivo (sobreescribe). Utilizarlo para crear la estructura inicial.

Ver también:

La documentacion oficial de shelve.

Ademas el la funcion *cargarSHELVE()*.

Examples

```
>>> import numpy as np
>>> datos = {'nombre': 'mlp', 'valor': 0.0, 'pesos': []}
>>> a = np.asarray([[7,8,9],[10,11,12]])
>>> guardarSHELVE(nombreArchivo="prueba.cupdyle", valor=datos,␣
↪nuevo=True)
>>> datos
{'nombre': 'mlp',
'valor': 0.0,
'pesos': []}
>>> guardarSHELVE("prueba.cupdyle",{'pesos':a})
>>> guardarSHELVE("prueba.cupdyle",{'valor':10.0, 'nombre':'mlp2'})
>>> cargarSHELVE("prueba.cupydle",None)
{'nombre': 'mlp2',
'valor': 10.0,
'pesos': [array([7, 8, 9]), array([8, 9, 10])]}
```

cupydle.dnn.utils.**indexable**(**iterables*)

Genera un arreglo se es indexable para la validacion cruzaja Chequea la consistencia.

Parámetros ***iterables** (*list*) – lists, dataframes, arrays, sparse matrices

cupydle.dnn.utils.**load**(*filename=None*, *compression=None*)

cupydle.dnn.utils.**min_max_escalado**(*datos*, *min_obj=0.0*, *max_obj=1.0*)

cupydle.dnn.utils.**mostrar_tamaniode**(*x*, *level=0*)

Imprime el tamanio en memoria requerido para el objeto *x*

Parámetros

- **x** (*Object*) –
- **level** (*int*) – cantidad de subniveles a entrar.

Devuelve imprime por pantalla el tamanio del objeto pasado

Examples

```
>>> mostrar_tamaniode(int)
4
```

cupydle.dnn.utils.**safe_indexing**(*X*, *indices*)

Retorna los items o filas de X usando indices Permite el indexados de listas o arreglos.

Parámetros

- **X** (*array*) – array-like, sparse-matrix, list.
- **indices** (*array*) – array-like, list

cupydle.dnn.utils.**save**(*objeto*, *filename=None*, *compression=None*)

cupydle.dnn.utils.**split_data**(*data*, *fractions*, *seed=123*)

Divide los datos en fracciones o partes

Parámetros

- **data** (*list*) – np.array array
- **fractions** (*list*) – [f_train, f_valid, f_test]
- **seed** (*int*) –

Devuelve sets (list) (e.g. train, valid, test)

class cupydle.dnn.utils.**temporizador**

Clases base: object

tipo singleton https://es.wikipedia.org/wiki/Singleton#Python

fin = None

inicio = None

instance = None

tic()

retorna una marca del temporizador... se ejecuta dos veces, una al inicio y otra al final... asi se cuenta la diferencia

tiempo()

static time()

toc()

transcurrido(*start=None*, *end=None*, *string=True*)

cupydle.dnn.utils.**vectorize_label**(*label*, *n_classes*)

Dada una etiqueta genera un vector de dimension (n_classes, 1) de 0 y un 1 en la posicion de la label

e.g: si label es '5', n_classes es '8': return: v[0]=0 v[1]=0 v[2]=0 v[3]=0 v[4]=1 v[5]=0 v[6]=0 v[7]=0

Parámetros

- **label** (*list*) –
- **n_classes** (*int*) –

cupydle.dnn.utils.**z_score**(*datos*)

cupydle.dnn.utils_theano module

cupydle.dnn.validacion_cruzada module

class cupydle.dnn.validacion_cruzada.**KFold**(*n*, *n_folds=3*, *shuffle=False*, *random_state=None*)

Clases base: cupydle.dnn.validacion_cruzada._BaseKFold

K-Folds cross validation iterator.

Parámetros

- **n** (*int*) – numero total de elementos
- **n_folds** (*int*) – int, default=3. nuemro de particiones

- **shuffle** (*bolleano[opcional]*) –
- **random_state** (*int, None*) –

Examples

```
>>> from cupydle.cross_validation import KFold
>>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
>>> y = np.array([1, 2, 3, 4])
>>> kf = KFold(4, n_folds=2)
>>> len(kf)
2
>>> print(kf)
cupydle.cross_validation.KFold(n=4, n_folds=2, shuffle=False,
                               random_state=None)
>>> for train_index, test_index in kf:
...    print("TRAIN:", train_index, "TEST:", test_index)
...    X_train, X_test = X[train_index], X[test_index]
...    y_train, y_test = y[train_index], y[test_index]
TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1] TEST: [2 3]
```

class cupydle.dnn.validacion_cruzada.**LabelKFold**(*labels*, *n_folds=3*)

Clases base: cupydle.dnn.validacion_cruzada._BaseKFold

K-fold iterator variant with non-overlapping labels.

Parámetros

- **labels** (*array*) – etiquetas
- **n_folds** (*int*) – numero de conjuntos a particionar

Examples

```
>>> from cupydle.dnn..cross_validation import LabelKFold
>>> X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
>>> y = np.array([1, 2, 3, 4])
>>> labels = np.array([0, 0, 2, 2])
>>> label_kfold = LabelKFold(labels, n_folds=2)
>>> len(label_kfold)
2
>>> print(label_kfold)
cupydle.dnn.cross_validation.LabelKFold(n_labels=4, n_folds=2)
>>> for train_index, test_index in label_kfold:
...     print("TRAIN:", train_index, "TEST:", test_index)
...     X_train, X_test = X[train_index], X[test_index]
...     y_train, y_test = y[train_index], y[test_index]
...     print(X_train, X_test, y_train, y_test)
...
TRAIN: [0 1] TEST: [2 3]
[[1 2]
 [3 4]] [[5 6]
 [7 8]] [1 2] [3 4]
TRAIN: [2 3] TEST: [0 1]
```

```
[[5 6]
 [7 8]] [[1 2]
 [3 4]] [3 4] [1 2]
```

class cupydle.dnn.validacion_cruzada.**LeaveOneOut**(*n*)

Clases base: cupydle.dnn.validacion_cruzada._PartitionIterator

Iterador de Leave one out. Para la validacion cruzada Divide los conjuntos de entrenamiento y validacion Cada muestra es seprada en dos conjuntos

---

Nota: LeaveOneOut(n) es equivalente a KFold(n,n_folds=n) y LeavePOut(n,p=1).

Parámetros **n** (*int*) – Nuemro total de ejemplos

Examples

```
>>> from cupydle.dnn.validacion_cruzaja import cross_validation
>>> X = np.array([[1, 2], [3, 4]])
>>> y = np.array([1, 2])
>>> loo = cross_validation.LeaveOneOut(2)
>>> len(loo)
2
>>> print(loo)
cupydle.cross_validation.LeaveOneOut(n=2)
>>> for train_index, test_index in loo:
...     print("TRAIN:", train_index, "TEST:", test_index)
...     X_train, X_test = X[train_index], X[test_index]
...     y_train, y_test = y[train_index], y[test_index]
...     print(X_train, X_test, y_train, y_test)
TRAIN: [1] TEST: [0]
[[3 4]] [[1 2]] [2] [1]
TRAIN: [0] TEST: [1]
[[1 2]] [[3 4]] [1] [2]
```

class cupydle.dnn.validacion_cruzada.**StratifiedKFold**(*y*, *n_folds=3*, *shuffle=False*, *random_state=None*)

Clases base: cupydle.dnn.validacion_cruzada._BaseKFold

Stratified K-Folds cross validation iterator

Parámetros

- **y** (*array*) – ejemplos del conjuntos
- **n_folds** (*int*) –
- **shuffle** (*boleano*) –
- **random_state** (*None*) –

Examples

```
>>> from cupydle.cross_validation import StratifiedKFold
>>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
>>> y = np.array([0, 0, 1, 1])
>>> skf = StratifiedKFold(y, n_folds=2)
>>> len(skf)
2
>>> print(skf)
cupydle.cross_validation.StratifiedKFold(labels=[0 0 1 1], n_folds=2,
                                         shuffle=False, random_
 ↪state=None)
>>> for train_index, test_index in skf:
...     print("TRAIN:", train_index, "TEST:", test_index)
...     X_train, X_test = X[train_index], X[test_index]
...     y_train, y_test = y[train_index], y[test_index]
TRAIN: [1 3] TEST: [0 2]
TRAIN: [0 2] TEST: [1 3]
```

Module contents

1.1.2 cupydle.test package

Subpackages

cupydle.test.mnist package

Submodules

cupydle.test.mnist.imagenes_mnist module

cupydle.test.mnist.imagenes_mnist.**crear**()

cupydle.test.mnist.imagenes_mnist.**guardar**(*entrenamiento*, *validacion*, *testeo*)

cupydle.test.mnist.imagenes_mnist.**histo**(*data*, *hatch*, *tam=13*)

cupydle.test.mnist.imagenes_mnist.**histo_todos**(*data1*, *data2*, *data3*, *tam=13*)

cupydle.test.mnist.imagenes_mnist.**medias**(*imagenes*)

cupydle.test.mnist.imagenes_mnist.**mnist_10means**(*means*)

cupydle.test.mnist.imagenes_mnist.**mnist_medias**(*imagenes_muestra*, *means*)

cupydle.test.mnist.imagenes_mnist.**plot_10digits**(*imagenes_muestra*)

cupydle.test.mnist.imagenes_mnist.**plot_30random**(*imagenes*)

cupydle.test.mnist.imagenes_mnist.**plot_mejor_iteracion**()

cupydle.test.mnist.mnist module

mnist

Download and draw/plot images based on the MNIST data. asa Ponzoni Nelson

class cupydle.test.mnist.mnist.**MNIST**(*path='.'*)
Clases base: object

classmethod display(*img*, *width=28*, *threshold=200*)

get_all()
devuelve todos los datos en una lista de tuplas, [(img,lbl)..]

get_testing()

get_training()

get_validation()

info
imprime por pantalla informacion relativa a la cantidad de cada tipo de datos en la bd

classmethod load(*path_img*, *path_lbl*)

load_data()
carga los datos en los arrays

load_testing(*numpy_array=True*)
si numpy_array es False se devuelve una lista, sino un tipo numpy array

load_training(*numpy_array=True*)
si numpy_array es False se devuelve una lista, sino un tipo numpy array

static plot_one_digit(*image*, *label=None*, *save=None*)
Plot a single MNIST image.

static plot_ten_digits(*images*, *save=None*, *crop=0*)
Plot a single image containing all six MNIST images, one after the other. if crop is true, Note that we crop the sides of the images so that they appear reasonably close together.

static plot_ten_digits2(*images*, *save=None*, *crop=0*)
Plot a single image containing all six MNIST images, one after the other. if crop is true, Note that we crop the sides of the images so that they appear reasonably close together.

static prepare(*directorio*, *nombre='mnist'*, *compresion='bzip2'*)

Parámetros **path** – ruta donde se encuentran los archivos desde internet

cupydle.test.mnist.mnist.**find**(*pattern*, *path*)

cupydle.test.mnist.mnist.**open4disk**(*filename='mnist'*, *compression='gzip'*)

cupydle.test.mnist.mnist.**save2disk**(*mnist*, *filename='mnist'*, *compression='gzip'*)

cupydle.test.mnist.mnist_loader module

class cupydle.test.mnist.mnist_loader.**MNIST**(*path='.'*)

Clases base: object

classmethod display(*img*, *width=28*, *threshold=200*)

classmethod load(*path_img*, *path_lbl*)

load_testing()

load_training()

cupydle.test.mnist.mnist_prueba module

cupydle.test.mnist.preview module

Module contents

Submodules

cupydle.test.dbn_FACE_validacionCruzada module

cupydle.test.dbn_base module

cupydle.test.dbn_prueba module

cupydle.test.dbn_prueba_CV module

cupydle.test.dbn_prueba_GS module

cupydle.test.dbn_prueba_GS_1 module

cupydle.test.mlp_FACE_validacionCruzada module

cupydle.test.mlp_KML module

cupydle.test.mlp_MNIST module

cupydle.test.rbm_KML module

cupydle.test.rbm_MNIST module

Module contents

1.2 Module contents

CAPÍTULO 2

Búsqueda de contenidos

- search

C

Symbols

B

C

D

F

G

H

U

V

Z

Referencias

[1] Enrique M Albornoz, María B Crolla, y Diego H Milone. Recognition of emotions in speech. En *XXXIV Conferencia Latinoamericana de Informática*, páginas 1120–1129. 2008.

[2] Alvarez Andrés, Fadil Carim, Rufiner Leonardo, y Martínez Cesar. Reconocimiento automático de emociones en contenido multimedia. Reporte técnico, Facultad de Ingeniería y Ciencias Hídricas, UNL, 2014.

[3] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. doi:10.1561/2200000006.

[4] Yoshua Bengio y Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural computation*, 21(6):1601–1621, 2009.

[5] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, y Yoshua Bengio. Theano: A cpu and gpu math compiler in python. En *Proc. 9th Python in Science Conf*, páginas 1–7. 2010.

[6] Caffe. Web oficial de caffe. [online] visitado el 25/08/2016, http://caffe.berkeleyvision.org/.

[7] Tsuhan Chen y Ram R Rao. Audio-visual integration in multimodal communication. *Proceedings of the IEEE*, 86(5):837–852, 1998.

[8] John Cheng, Max Grossman, y Ty McKercher. *Professional Cuda C Programming*. John Wiley & Sons, 2014.

[9] M Bishop Christopher. *Pattern Recognition and Machine Learning*, volumen 4 de *Information Science and Statistics*. Springer-Verlag New York, Cambridge UK, 2006. ISBN 978-0387-31073-2.

[10] Neri E Cibau, Enrique M Albornoz, y Hugo L Rufiner. Speech emotion recognition using a deep autoencoder. *Proceedings of the XV Reunión de Trabajo en Procesamiento de la Información y Control (RPIC 2013), San Carlos de Bariloche*, 2013.

[11] Nvidia Corporation. *Tesla C1060 Computing Processor Board*. Nvidia Corporation, 2008.

[12] Nvidia Corporation. Nvidia machine learning web, 2016. [online] visitado el 25/08/2016, http://www.nvidia.com/object/machine-learning.html.

[13] Ali Mamdouh Elkahky, Yang Song, y Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. En *Proceedings of the 24th International Conference on World Wide Web*, páginas 278–288. ACM, 2015.

[14] Asja Fischer y Christian Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[15] Github. Listado de software relacionado a deep learning. [online] visitado el 25/08/2016, https://goo.gl/6NvDMq.

[16] Alan Graves, Abdel-rahman Mohamed, y Geoffrey Hinton. Speech recognition with deep recurrent neural networks. En *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, páginas 6645–6649. IEEE, 2013.

[17] Simon Haykin. *Neural networks and learning machines*, volumen 3. Pearson Education, Upper Saddle River, New Jersey, 2009. ISBN 978-0-13-147139-9.

[18] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[19] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[20] Geoffrey E Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[21] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[22] Geoffrey E Hinton, Simon Osindero, y Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[23] Geoffrey E. Hinton y Ruslan R Salakhutdinov. Replicated softmax: an undirected topic model. En Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, y A. Culotta, editores, *Advances in Neural Information Processing Systems 22*, páginas 1607–1614. Curran Associates, Inc., 2009.

[24] kdnuggets. 50 deep learning software tools and platforms, updated. [online] visitido el 25/08/2016, http://www.kdnuggets.com/2015/12/deep-learning-tools.html.

[25] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, y Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40, 2009.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, y Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Yann LeCun, Corinna Cortes, y Christopher JC Burges. The mnist database of handwritten digits, 1998. [online] visitado el 25/08/2016, http://yann.lecun.com/exdb/mnist/.

[28] Michael Lewis, Jeannette M Haviland-Jones, y Lisa Feldman Barrett. *Handbook of emotions*. The Guilford Press, 2008.

[29] LISA-lab. Coleccion de tutoriales de deep learning. [online] visitado el 25/08/2016, http://deeplearning.net/tutorial/.

[30] LISA-lab. Direccion de paginas sobre librerias de machine learning. [online] visitado el 25/08/2016, http://deeplearning.net/softwarelinks/.

[31] LISA-lab. Listado de software relacionado a deep learning. [online] visitido el 25/08/2016, http://deeplearning.net/software_links.

[32] LISA-lab. Web oficial de pylearn2. [online] visitado el 25/08/2016, http://deeplearning.net/software/pylearn2/.

[33] LISA-lab. Web oficial de theano. [online] visitado el 25/08/2016, http://deeplearning.net/software/theano/.

[34] Daniel L Ly, Volodymyr Paprotski, y Danny Yen. Neural networks on gpus: Restricted boltzmann machines. *ver http://www. eecg. toronto. edu/moshovos/CUDA08/doku. php*, 2008.

[35] Tim K Marks y Javier R Movellan. Diffusion networks, product of experts, and factor analysis. En *Proc. Int. Conf. on Independent Component Analysis*, páginas 481–485. 2001.

[36] Aleix Martinez y Shichuan Du. A model of the perception of facial expressions of emotion by humans: Research overview and perspectives. *Journal of Machine Learning Research*, 13(May):1589–1608, 2012.

[37] NVIDIA Corporation. *NVIDIA CUDA Programming Guide*. NVIDIA Corporation, 2012.

[38] Christina Regenbogen, Daniel A Schneider, Andreas Finkelmeyer, Nils Kohn, Birgit Derntl, Thilo Kellermann, Raquel E Gur, Frank Schneider, y Ute Habel. The differential contribution of facial expressions, prosody, and speech content to empathy. *Cognition & emotion*, 26(6):995–1014, 2012.

[39] Christina Regenbogen, Daniel A Schneider, Raquel E Gur, Frank Schneider, Ute Habel, y Thilo Kellermann. Multimodal human communication—targeting facial expressions, speech content and prosody. *Neuroimage*, 60(4):2346–2356, 2012.

[40] Henry A Rowley, Shumeet Baluja, y Takeo Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998. ISSN 0162-8828. doi:10.1109/34.655647.

[41] Ruslan Salakhutdinov y Hugo Larochelle. Efficient learning of deep boltzmann machines. En *International Conference on Artificial Intelligence and Statistics*, páginas 693–700. 2010.

[42] Ruslan Salakhutdinov, Andriy Mnih, y Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. En *Proceedings of the 24th international conference on Machine learning*, páginas 791–798. ACM, 2007.

[43] Ian Sommerville. *Software engineering*. Pearson Education, 9 edición, 2011.

[44] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, y Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[45] Joshua M Susskind, Adam K Anderson, Geoffrey E Hinton, y Javier R Movellan. *Generating facial expressions with deep belief nets*. INTECH Open Access Publisher, 2008.

[46] Ilya Sutskever y Tijmen Tieleman. On the convergence properties of contrastive divergence. En *International Conference on Artificial Intelligence and Statistics (AISTATS*, páginas 789–795. 2010.

[47] Yegor Tkachenko. Autonomous crm control via clv approximation with deep reinforcement learning in discrete and continuous action space. *arXiv preprint arXiv:1504.01840*, 2015.

[48] Torch. Web oficial de pylearn2. [online] visitado el 25/08/2016, http://torch.ch/.

[49] Aaron Van den Oord, Sander Dieleman, y Benjamin Schrauwen. Deep content-based music recommendation. En *Advances in Neural Information Processing Systems*, páginas 2643–2651. 2013.

[50] Fernando Villada, Nicolás Muñoz, y Edwin García. Aplicación de las redes neuronales al pronóstico de precios en el mercado de valores. *Información tecnológica*, 23(4):11–20, 2012.

[51] Oriol Vinyals, Alexander Toshev, Samy Bengio, y Dumitru Erhan. Show and tell: A neural image caption generator. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, páginas 3156–3164. 2015.

[52] Yongjin Wang y Ling Guan. Recognizing human emotion from audiovisual information. En *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volumen 2, páginas ii–1125. IEEE, 2005.

[53] Max Welling y Charles A Sutton. Learning in markov random fields with contrastive free energies. En *AISTATS*. 2005.

[54] Wucius Wong. *Principles of form and design*. John Wiley & Sons, 1993.

[55] Gordon Wyeth, Gregg Buskey, y Jonathan Roberts. Flight control using an artificial neural network. En *Proceedings of the Australian Conference on Robotics and Automation (ACRA 2000), August*. 2000.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
N. C. Ponzoni, H. L. Rufiner & C. E. Martínez; "Desarrollo de una biblioteca para sistemas basados en aprendizaje profundo mediante GP-GPU: aplicación al reconocimiento de emociones (Undergraduate project)" Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2017.

Índice alfabético