CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco
Departamento de Control Automático

El aprendizaje profundo para la identificación de sistemas no lineales

Tesis que presenta

Ing. Erick Dasaev de la Rosa Montero

Para obtener el grado de

Maestro en ciencias

En la especialidad de

Control Automático

Director de tesis

Dr. Wen Yu Liu

México, D.F. **Agosto, 2014**

"He sido un niño pequeño que, jugando en la playa, encontraba de tarde en tarde un guijarro más fino o una concha más bonita de lo normal, mientras que el océano de la verdad se extendía delante de mi, todo él por descubrir"

Isaac Newton

Agradecimientos

Agradezco a la vida por permitirme existir y darme la oportunidad de ser.

Agradezco a mis padres Minerva y Raúl quienes sobre todas las cosas me han brindado el soporte necesario para lograr mis objetivos y que han creído en mi a pesar de mis tropiezos y los amo con todo mi ser.

Gracias al Dr. Wen Yu Liu por su valioso consejo y guía sin los cuales hubiese sido imposible la realización de este trabajo científico.

Le doy las gracias al Centro de Investigación y Estudios Avanzados por proporcionarme los medios materiales e intelectuales que permitieron llegar a esta nueva etapa.

Finalmente agradezco al Consejo Nacional de Ciencia y Tecnología por la beca que me fue otorgada para la realización de mis estudios.

Abreviaturas

BM: Boltzmann Machines (Máquinas de Boltzmann)

DBN: Deep Belief Network (Red de Creencia Profunda)

DNN: Deep Neural Network (Red Neuronal Profunda)

ICA: Independent Component Analysis (Análisis de Componentes Independientes)

MLP: Multilayer Perceptron (Perceptrón Multicapa)

PCA: Principal Component Analysis (Análisis de Componentes Principales)

RBM: Restricted Boltzmann Machines (Máquinas Restringidas de Boltzmann)

RNA: Red Neuronal Artificial

SBN: Sigmoid Belief Network (Red de Creencia Sigmoide)

SDA: Stacked Denoising Autoencoders (Codificadores Ruidosos Apilados)

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El primer precedente en el uso exitoso del aprendizaje profundo se debe a Geoffrey Hinton [29, 30] quien introdujo las Redes de Creencia Profunda utilizando en cada capa de la red una Máquina de Boltzmann Restringida (RBM) para la asignación inicial de los pesos sinápticos .

El principio general del funcionamiento de una arquitectura profunda es guiar el entrenamiento de las capas intermedias de representación utilizando aprendizaje no supervisado específicamente para cada capa; para ello se utilizan distintas técnicas entre las que destacan los Autoencoders [32], las RBMs [31] y las DBNs [29] (Deep Belief Networks). Aunque estas técnicas constituyen en sí mismas una red neuronal, han sido utilizadas para inicializar los pesos de arquitecturas profundas de redes neuronales con conexiones hacía adelante supervisadas.

En el presente trabajo se explorar la aplicación de técnicas de preentrenamiento no supervisado en arquitecturas profundas en la tarea de identificación de sistemas no lineales comparando la eficiencia de estas técnicas con los resultados obtenidos por medio de sistemas enteramente supervisados.

1.1. Motivación

Una arquitectura profunda se define como una composición de varias etapas de procesamiento. En el caso de una red neuronal el principal problema del uso de una arquitectura profunda es el siguiente: Si una buena representación de la entrada ha sido encontrada en cada capa de la red, ésta puede ser utilizada para inicializar y entrenar las capas subsecuentes por medio de una optimización supervisada basada en gradiente descendente; sin embargo, la llamada "buena representación" es difícil de conseguir por medio de métodos habituales de asignación de pesos sinápticos [9, 21].

Se ha encontrado que la representación del conocimiento en el cerebro humano se lleva a a cabo de manera distribuida [12, 28], lo que significa que el aprendizaje se distribuye

en un número grande de neuronas en lugar de estar focalizado en una región particular; además se usa una representación esparcida, es decir, si bien el conocimiento se distribuye en muchas neuronas, también se esparse entre ellas de tal manera que solamente entre uno y cuatro por ciento de las neuronas humanas se encuentran activas en un instante dado.

1.2. Estructura de la tesis

En el presente capítulo se da una noción general de la motivación e importancia que tienen las arquitecturas profundas en la solución de problemas de clasificación y regresión.

En el capítulo 2 se mencionan algunos resultados teóricos y de simulación encontrados en la literatura, que sugieren ventajas en el desempeño de topologías profundas frente a topologías compuestas por pocas capas de representación.

Los algoritmos principales de ajuste inicial de pesos sinápticos y de ajuste supervisado para modelos de regresión son presentados en el capítulo 3, además se introducen algunas topologías de identificación neuronal.

El procedimiento de aprendizaje de Boltzmann es introducido en el capítulo 4, esto sirve como marco teórico para el entendimiento de algoritmos de preentrenamiento posteriormente presentados. A su vez, se describen técnicas estocásticas para la elección de los hiperparámetros del modelo.

Se expone el método de autocodificación en el capitulo 5, introduciendo a los autoencoders como bloques constructores de una arquitectura profunda.

En el capitulo 6 se describe el aprendizaje por medio de máquinas de Boltzmann restringidas detallando las consideraciones necesarias para su entrenamiento. Se presentan los resultados obtenidos resolviendo los problemas de clasificación de patrones MNIST e identificación de sistemas no lineales, se comparan la exactitud obtenida en ambas aplicaciones exponiendo las diferencias y similitudes obtenidas.

Finalmente en el capítulo 7 se detallan las conclusiones a las que se ha llegado en la investigación sugiriendo además algunas líneas de trabajo y áreas de oportunidad.

Capítulo 2

Aprendizaje profundo

Un problema resuelto por medio de la aplicación de diversos algoritmos pertenecientes a la Inteligencia Artificial consiste en la representación de una función por medio de una arquitectura de aprendizaje. Decimos que la representación de una función es compacta cuando tienes pocos elementos computacionales o grados de libertad que deben ser ajustados utilizando un procedimiento de aprendizaje [4, 28, 3].

En este contexto una función puede ser representada por arquitecturas computacionales muy diversas. Funciones que pueden ser representadas compactamente por una arquitectura de profundidad k pueden requerir un número exponencialmente grande de elementos para ser representadas por una arquitectura de profundidad $k-1$. Una característica fundamental es que entre mayor sea el número de elementos utilizados para representar una función, mayor será el número de patrones necesarios para el proceso de aprendizaje con el objetivo de obtener una buena generalización.

Se define a un conjunto de elementos computacionales como el conjunto de cálculos que deben ser hechos para resolver determinado problema. Por ejemplo, el conjunto de todos los cálculos que pueden ser hechos utilizando solamente las operaciones básicas de suma, resta, producto y división. Una función puede ser expresada por la composición de elementos computacionales de un conjunto dado. Esta composición se define generalmente mediante un grafo representando cada elemento computacional por un nodo del grafo.

Se ha hablado de la profundidad de una arquitectura de modelo pero no se ha dado una definición formal de la misma, el concepto de profundidad de una arquitectura de representación se refiere a la profundidad del grafo o dibujo correspondiente, es decir, el camino mas largo desde un nodo de entrada hasta un nodo de salida. Cuando el conjunto de elementos computacionales es el conjunto de cálculos que puede llevar a cabo una neurona artificial, la profundidad corresponde al número de capas de la red neuronal.

Ejemplo:

Considere la función $f(x) = a + x * \sin(a - b)$ si definimos el conjunto de elementos computacionales permitidos como $\{\sin, *, +, -\}$ y utilizamos como entradas los valores $\{x, a, b\}$ podemos obtener una representación de la función $f(x)$ como se muestra en la Figura 2.0.1:

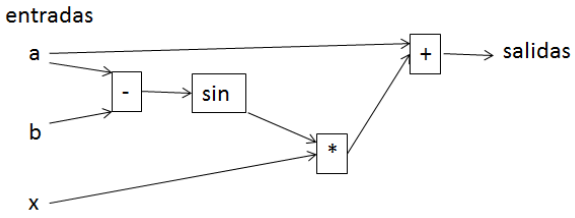


Figura 2.0.1: Arquitectura de representación con profundidad 4

La profundidad de un modelo es objeto de estudio no solamente en el ámbito computacional sino también constituye un elemento de valoración en el área de neurociencias. Algunas investigaciones sugieren que la corteza cerebral puede ser vista como una arquitectura profunda que involucra de 5 a 10 capas durante el procesamiento de la información visual [23].

Una neurona artificial puede ser vista como la combinación de una transformación afín (aplicación y suma de pesos sinápticos) seguida de una transformación no lineal (función de activación). Al colocar neuronas artificiales como elementos computacionales de una arquitectura profunda obtenemos redes neuronales artificiales multicapa. La forma más común de este tipo de redes consiste en la elección de solo una capa oculta relacionada con una capa de salida con lo que obtenemos una profundidad de 2.

Considérese un modelo multitarea, supóngase un sistema con múltiples salidas en donde cada una corresponde a un objetivo que debe de ser alcanzado, si muchas de estas características deseadas son compartidas y tienen correlación unas con otras se consigue robustez del sistema.

2.1. Objetivos del aprendizaje utilizando inteligencia artificial

Para cualquier sistema de entrenamiento o aprendizaje basado en IA (inteligencia artificial) se busca que cumpla con las siguientes características [18, 19]:

1. Habilidad para aprender funciones complejas altamente variables con relativamente pocos ejemplos de entrenamiento.

2. Habilidad para aprender de manera no supervisada (sin interacción humana).

3. Habilidad para aprender de una vasta cantidad de ejemplos.

4. Habilidad para aprovechar la sinergia del fenómeno, es decir, tener un aprendizaje multiobjetivo.

5. Habilidad de conseguir una buena generalización en el caso de aprendizaje no supervisado.

2.2. Complejidad computacional

Se ha visto con anterioridad que cuando una función puede ser representada por una arquitectura profunda compactamente, ésta suele necesitar un número grande de elementos computacionales para ser representada por una arquitectura poco profunda.

El tamaño de una arquitectura se define como el número de elementos computacionales que la constituyen.

Considérese el siguiente caso: Supóngase un circuito de umbral monotónicamente ponderado con profundidad $k-1$ calculando una función $f_k \in \mathcal{F}_{K,N}$ tiene al menos un tamaño 2^{cN} para alguna constante $c > 0$ y $N > N_0$

La clase de funciones $\mathcal{F}_{K,N}$ contiene funciones con N^{2K-2} entradas definidas para un circuito de profundidad K. El circuito puede verse como un árbol booleano en cuyas hojas se encuentran las variables de entrada (no negadas) y el valor final de la función desciende hasta la raíz [14]. El $i-ésimo$ nivel o capa desde la base consiste en compuertas AND cuando i es par y compuertas OR si i es impar. El $fan-in$ (número de entradas soportadas por un elemento computacional) en la capa de la copa y en la base del árbol es de N y en las capas restantes es de N^2 [15].

El ejemplo anterior es un caso concreto del aumento exponencial en el número de elementos computacionales en una arquitectura de representación si se utiliza una profundidad menor a la apropiada para una función. Esto no comprueba que todas las funciones requieran de una representación utilizando una arquitectura profunda pero si sugiere que existe una profundidad adecuada para cada función.

Si una función presenta muchas variaciones y estas variaciones no están relacionadas entre sí mediante una regularidad subyaciente, entonces ningún algoritmo de aprendizaje no local funcionará mucho mejor que los estimadores en el dominio local. Los árboles de decisión son una de las herramientas mas utilizadas en el diseño de sistemas de aprendizaje. Una de sus características más sobresalientes es que necesitan al menos tantos ejemplos de entrenamiento como variaciones de la función objetivo.

Si se emplean conjuntos de árboles trabajando cooperativamente, esto es equivalente a añadir una tercer capa en la arquitectura que permita discriminar entre un elevado número de regiones dentro del espacio de parámetros. Estas regiones forman implícitamente una representación distribuida utilizando la salida de todos los árboles en el

bosque. Por ejemplo, considérese un conjunto de tres árboles binarios, cada árbol es capaz de dividir el espacio de parámetros en dos regiones, por lo que los tres árboles por separado dividirían el espacio en 6 zonas; suponga que los árboles se encuentran colaborando en un bosque de decisión, con esta arquitectura los árboles dividen al espacio en 7 regiones en lugar de 6 lo cual sugiere que entre mayor sea la cantidad de árboles, la cantidad de regiones en las cuales el espacio es dividido aumenta de manera exponencial (ver Figura 2.2.1).

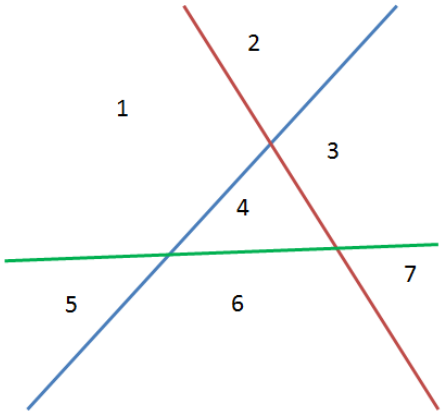


Figura 2.2.1: Bosque de decisión compuesto por 3 árboles, se forman 7 regiones de clasificación

2.3. Aprendizaje de representaciones distribuidas

Como se ha detallado antes, una representación distribuida puede ser mucho mas compacta que una local debido a que la distributividad puede alcanzarse aumentando la profundidad de la arquitectura que se este usando y por lo tanto se reduce exponencialmente el número de elementos computacionales utilizados.

En una representación distribuida el patrón de entrada es representado por un conjunto de características que no son mutuamente exclusivas pero si pueden ser estadísticamente independientes. En una representación local el proceso de clusterización se caracteriza porque los cluster no tienen regiones comunes y por lo tanto su intersección es nula.

A diferencia de la clusterización local, en la clusterización distribuida, el número de clases posibles es comparativamente inmenso y su comportamiento es similar al *overlapping* descrito en los árboles de decisión [15]. En el contexto de las redes neuronales artificiales, las redes multicapa y las máquinas de Boltzmann tienen como objetivo el aprendizaje de una representación distribuida del vector de entrada por medio de las capas de neuronas ocultas.

Se dice que una función es altamente variante cuando una aproximación “a pedazos” de la función requiere el uso de muchas piezas. Una función de este estilo puede ser representada por una arquitectura profunda compactamente mientras que generalmente

se necesitaría una gran cantidad de elementos computacionales para ser representada por una arquitectura de profundidad 2 (por ejemplo una Perceptrón con una sola capa oculta).

Algunos psicólogos cognitivos han estudiado la idea de computación neuronal proponiendo en el cerebro una jerarquía de niveles de representación correspondiente a diferentes niveles de abstracción con una representación distribuida en cada nivel.

2.4. Generalización local vs generalización no local

Un estimador local en el espacio de entrada obtiene una buena generalización para un vector de entrada x explotando ejemplos de entrenamiento cerca de la vecindad de x. Los estimadores locales "parten" el espacio de entrada en regiones y requieren muchos parámetros para modelar la función objetivo en cada región. Cuando muchas regiones son necesarias debido a la gran variabilidad de la función objetivo, el número de parámetros crecerá, generando la necesidad de un número alto de ejemplos para producir una buena aproximación.

El problema de la generalización está estrechamente relacionado con la dimensionalidad de la función; sin embargo, el factor mas importante en el proceso de generalización es la cantidad de variaciones que presente la función objetivo que deseamos sean aprendidas por nuestro estimador.

Existen arquitecturas basadas en el emparejamiento de plantillas locales formando dos niveles de abstracción. En el primer nivel, una plantilla obtendrá una salida cuyo valor indica el grado de emparejamiento con cada ejemplo de entrenamiento mientras que la segunda capa utiliza estos valores para estimar una salida conveniente.

Un ejemplo de esta conducta de dos capas es la *Kernel Machine* [4] denotada por la ecuación 2.4.1.

$$f(x) = b + \sum_i \alpha_i K(x,\ x_i) \tag{2.4.1}$$

donde b y α_i forman el segundo nivel de decisión, mientras que la función kernel $K(x,\ x_i)$ relaciona la entrada x con el ejemplo de entrenamiento x_i. En este caso también existen funciones de kernel locales y distribuidas. Un kernel es local cuando la relación $K(x,\ x_i) > \rho$ con ρ positivo, es verdad solo para $x's$ en cierta región conectada alrededor de x_i. Un ejemplo de kernel local es la función Gaussiana denotada por la ecuación 2.4.2.

$$K(x,\ x_i) = e^{-(\|x-x_i\|/\sigma)^2} \tag{2.4.2}$$

donde σ controla el tamaño de la región alrededor de x_i.

Las *Kernel Machines* explotan la llamada prioridad de suavidad, es decir, si en la red se presenta como entrada el patrón x_i es inteligente esperar que el predictor otorgue una salida cercana a y_i dado que esta es salida para la cual fue entrenado. Esta técnica es buena en una gran cantidad de aplicaciones pero resulta insuficiente cuando se trata de modelar funciones altamente variables debido a la falta de un número suficiente de patrones de entrenamiento. Para ello se recurre a otra técnica llamada Deep Belief Networks.

Definimos el espacio característico de una arquitectura de representación como una representación interna de los datos de entrenamiento en la cual se utilizan sus características intrínsecas y que además es más útil que los datos originales en procesos de reconocimiento y estimación.

Utilizando una red neuronal multicapa que calcule el espacio característico de la función, la arquitectura puede ser ajustada para optimizar el error obtenido por la *Kernel Machine* utilizando el método del gradiente descendente.

2.5. Aprendizaje profundo y redes neuronales multicapa

En una red neuronal multicapa, la capa k calcula un vector de salida h^k usando la salida h^{k-1} obtenida de la capa anterior comenzando este proceso con la entrada $x = h^0$. Un ejemplo clásico de la función de activación de una red neuronal es la tangente hiperbólica en cuyo caso la salida de la capa analizada estaría dada por la ecuación 2.5.1:

$$h^k = \tanh\left(b^k + W^k h^{k-1}\right) \tag{2.5.1}$$

donde b^k es el vector de offsets y W^k es la matriz de pesos, la función de activación puede ser cambiada dependiendo el problema que se este atacando, otra función muy utilizada es la función sigmoide dada por la ecuación 2.5.2.

$$\text{sigm}(u) = \frac{1}{1+e^{-u}} = \frac{1}{2}\left(\tanh(u) + 1\right) \tag{2.5.2}$$

Si denotamos la salida total de la red por h^l, podemos calcular, utilizando además la información del objetivo supervisado y, la función de costo $L(h^l, y)$, la cual es típicamente convexa para $b^l + W^l h^{l-1}$. Una variación de la red neuronal es aquella en la que la capa de la salida tiene una función de activación diferente a la utilizada en las demás capas, un ejemplo de esto es la ecuación 2.5.3:

$$h_i^l = \frac{\exp\left(b_i^l + W_i^l h_i^{l-1}\right)}{\sum_j \exp\left(b_j^l + W_j^l h_j^{l-1}\right)} \tag{2.5.3}$$

donde $\sum_i h_i^l = 1$ y h_i^l es positiva. La salida h_i^l puede ser utilizada como estimador de $P(Y = y|x)$. En este caso, se utiliza usualmente una negativa de log-verosimilitud $L(h^l, y) = -\log P(Y = y|x) = -\log h_y^l$ cuyo valor esperado sobre (x, y) tiene que ser minimizado.

2.6. La dificultad de entrenar una red neuronal con arquitectura profunda

Hasta antes de 2006 [35, 29], las arquitecturas profundas no habían sido objeto de discusión dentro del ámbito científico debido a los pobres resultados que se habían tenido en su proceso de aprendizaje. Este comportamiento era generalmente obtenido utilizando una inicialización aleatoria de los pesos sinápticos de la red. Los únicos casos de éxito que habían sido reportados hasta ese momento habían sido las llamadas redes convolucionales [36].

Muchas observaciones sugieren que el entrenamiento basado en el método del gradiente descendente para redes neuronales profundas (deep neural network, DNN) supervisadas se estanca en lo que ha sido llamado mínimo local aparente. En general, a pesar de la mejor representación obtenida utilizando una arquitectura profunda se obtienen peores resultados que con soluciones obtenidas por medio de arquitecturas con una sola capa de neuronas ocultas.

El problema descrito anteriormente fue atacado desde diferentes puntos de vista. Se descubrió que se obtienen resultados notoriamente mejores si cada capa de la red es entrenada utilizando un algoritmo de preentrenamiento no supervisado sobre cada una de las capas comenzando con la capa de entrada a la red (se propone utilizar con este fin autoencoders o máquinas de Boltzmann). Después de tener todas las neuronas pre-entrenadas, la red puede ser ajustada utilizando un criterio supervisado como lo es el método del gradiente descendente.

Bajo las consideraciones anteriores surge la idea de que inyectando una señal de entrenamiento no supervisado a cada capa, se pueden guiar los parámetros de la capa de la red hacia mejores regiones dentro del espacio de parámetros [35, 28, 21].

Dentro de la arquitectura de las redes neuronales podemos considerar que la salida de cada capa de la red constituye una nueva representación del vector de entrada x, de esta manera considere a $h^k(x)$ como la representación nivel k de x en el modelo.

Inmersos en este contexto consideremos un criterio local definido en cada capa de la red que empuje las representaciones $h^k(x)$ y $h^k(\tilde{x})$ ya sea hacía una misma región o lejos el uno del otro dependiendo si los patrones de entrenamiento sugieren si deben ser vecinos o no.

El pre-entrenamiento no supervisado conduce a una restricción sobre la región del espacio de parámetros donde una solución es permitida, lo que le otorga una conducta

regulatoria.

2.6.1. Cualidad regulador-optimizador

Cuando la última capa oculta es restringida a un número pequeño de elementos computacionales, las DNN con inicialización de pesos aleatoria se comportan de manera poco eficiente tanto en tareas de identificación como de clasificación. Sin embargo; si se le permite a la capa crecer indefinidamente, existen casos donde el error acumulado durante la fase de entrenamiento resulta atractivamente pequeño; esto es debido a que la gran cantidad de elementos de la capa es suficiente para representar cualquier patrón de entrenamiento adecuadamente.

Otra técnica que ha sido utilizada durante el entrenamiento de redes neuronales con arquitecturas profundas es el entrenar de una manera general la red neuronal y solamente aplicar algoritmos de optimización a los procesos de entrenamiento de las 2 capas superiores de la red, esto es mucho más sencillo que el proceso de optimización de toda las red y además proporciona resultados mejores que cuando se implementa un algoritmo de entrenamiento global. Si existen suficientes unidades en la última capa oculta, el error de entrenamiento puede ser muy pequeño a pesar de que las capas inferiores no sean apropiadamente entrenadas. Cabe mencionar que aunque el error de entrenamiento sea bajo esto no implica que la red tenga una buena generalización por lo que el error de salida con un patrón de entrada diferente a los utilizados durante la fase de aprendizaje en general producirá un error considerablemente alto.

Al fenómeno de obtener un error de entrenamiento bajo y un error de generalización alto es conocido como $overfitting$ el cual puede ser comprendido como un sobreaprendizaje de una región particular del espacio de parámetros especializándose solamente en una región del mismo.

Aunque el error de entrenamiento puede ser reducido explotando solamente la habilidad de las capas superiores de ajustarse a los ejemplos de entrenamiento, una buena generalización es notoriamente difícil de alcanzar con lo que se debe de recurrir a métodos que ajusten todas las capas de la red apropiadamente minimizando alguna función de costo de reconstrucción. Tal ajuste como se ha dicho anteriormente puede ser llevado a acabo por algún pre-entrenamiento de las capas superiores que tiene como meta restringir a las capas inferiores a capturar patrones y regularidades de la distribución de los patrones de entrada.

Un caso particular de esta ideología es el siguiente: considere la pareja entrada-salida aleatoria (X, Y), en este ejemplo tal regularización es similar al efecto normativo de maximizar la verosimilitud de $P(X, Y)$ contra $P(Y|X)$. Si $P(X)$ y $P(Y|X)$ son funciones no relacionadas de X (es decir, escogidas independientemente tal que el aprendizaje de una no proporciona ninguna información de la otra), entonces el aprendizaje no supervisado de $P(X)$ no ayudará en el aprendizaje de $P(Y|X)$. Pero si están relacionados

y los mismos parámetros están involucrados en su estimación entonces cada vector de entrada-salida (X, Y) proporcionará información de $P(Y|X)$ a través de $P(X)$

Si se considera el problema de ajustar las capas inferiores mientras se mantiene un número pequeño de unidades computacionales en la última capa de neuronas ocultas y además se requiere que la magnitud de los pesos en las dos últimas capas no exceda cierto umbral, entonces estamos hablando de un problema de optimización dentro de la arquitectura de la red neuronal.

Si la hipótesis que se ha presentado es correcta, esperaríamos que el pre-entrenamiento supervisado tuviera beneficios sobre la inicialización de pesos aleatoria aun cuando se tuviera a disposición un conjunto infinito de patrones de entrenamiento. Si el efecto del pre-entrenamiento fuese solamente regulatorio, se esperaría que el error durante pruebas de generalización convergiera aproximadamente de la misma manera o al mismo rango de valores tanto con el pre-entrenamiento como sin el.

Cuando se entrena una arquitectura profunda sin el efecto de un pre-entrenamiento en las capas inferiores generalmente provoca que la dinámica del entrenamiento quede atrapada en algún mínimo local aparente y de este modo, la adición de mas ejemplos de entrenamiento no proveerá información suficiente para salir de esa región atractiva.

La información anterior sugiere que el método del gradiente descendente utilizado en conjunto con técnicas de *backpropagation* puede no ser efectivo para mover los parámetros de una capa oculta en particular hacía regiones correspondientes a soluciones deseadas [21].

Una topología diferente de RNA es la red neuronal recurrente, este tipo de redes se caracteriza por tener lazos de retroalimentación entre las capas superiores y las capas inferiores, de este modo, este tipo de redes puede ser "desdoblada en el tiempo" considerando la salida de cada neurona en distintos instantes como una variable diferente convirtiendo la red desdoblada en una arquitectura profunda.

2.6.2. Entrenamiento no supervisado para arquitecturas profundas

Dado el problema consistente en la existencia de mínimos locales durante la fase de entrenamiento, se propone un criterio de aprendizaje no supervisado definido al nivel de cada capa que puede ser usado para salir o evitar esos mínimos locales que el método del gradiente descendente no puede superar. Esto es de esperarse si el algoritmo de aprendizaje descubriera una representación que capture regularidades estadísticas del vector de entrada.

Algunas técnicas como son el PCA (Principal Component Analysis) o ICA (Independent Component Analysis) fallan en el proceso de obtención de estas características en el llamado caso sobredeterminado donde el número de salidas es mayor que el número de

entradas dado que las dos técnicas anteriormente citadas requieren una relación entrada-salida 1 a 1. En este caso existen variantes especialmente desarrolladas de ICA que resuelven este problema, así como algoritmos relacionados como son los autoencoders y las Máquinas Restringidas e Boltzmann (RBM) [30, 35].

Podemos concluir que los algoritmos de entrenamiento no supervisados pueden extraer información saliente de la distribución del vector de entrada. Esta información puede ser capturada en una representación distribuida, es decir, un conjunto de características en las cuales se codifican los factores sobresalientes de las variaciones en el vector de entrada.

Una arquitectura profunda con aprendizaje no supervisado puede entenderse de la siguiente manera: las características obtenidas por la primer capa son vistas como de primer nivel de abstracción mientras que las de las capas superiores representan un nivel de abstracción mas alto.

2.6.3. Arquitecturas degenerativas profundas

El entrenamiento de una red utilizando un algoritmo no supervisado es de interés para la obtención de una representación distribuida de la entrada y generar muestras de dicha distribución.

Los modelos generadores de la distribución pueden ser visualizados como grafos cuyos nodos representan variables aleatorias mientras que los lazos conectores proporcionan información acerca de la dependencia mutua entre las variables. Un ejemplo de arquitectura degenerativa profunda es la Red de Creencia Sigmoide (SBN por sus siglas en inglés), en esta red las unidades (típicamente variables aleatorias binarias) pertenecientes a cada capa son independientes de las unidades correspondientes a cualquier capa superior como se muestra en la Figura 2.6.1:

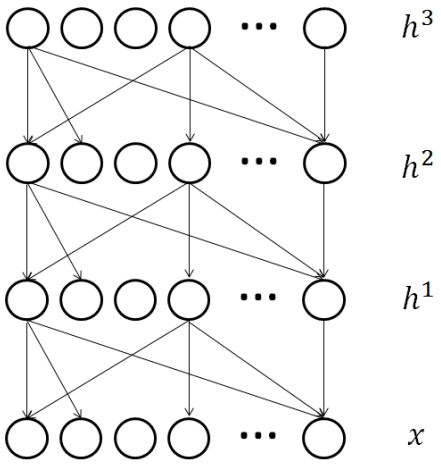


Figura 2.6.1: Topología de una red de creencia sigmoide, no existen dependencias hacía atrás

Una parametrización típica de este tipo de red es la ecuación de activación neuronal (ecuación 2.6.1):

$$P\left(h_i^k = 1|h^{k+1}\right) = \text{sigm}\left(b_i^k + \sum_j W_{i,j}^{k+1} h_j^{k+1}\right) \tag{2.6.1}$$

donde h_i^k es la activación binaria de nodo oculto i perteneciente a la capa k, mientras h^k es el vector de salidas de representación $\left(h_1^k, h_2^k, ...\right)$. Denotamos como el vector de entrada x a la capa h^0. Si hacemos que $\hat{P}$ represente la distribución de entrenamiento, es decir, la distribución generadora de ejemplos para el proceso de aprendizaje) podemos utilizar dicha distribución para la predicción de las estimaciones. La capa inferior genera un vector x en el espacio de entrada y nuestro objetivo es obtener un buen modelo de representación que otorgue una probabilidad alta de pertenencia a los datos de entrada. Considerando varias capas de la red, el modelo generador es descompuesto de la forma:

$$P\left(x, h^1, ..., h^l\right) = P\left(h^l\right)\left(\prod_{k=1}^{l-1} P\left(h^k|h^{k+1}\right)\right) P\left(x|h^1\right) \tag{2.6.2}$$

En una red de creencia sigmoide $P\left(h^l\right)$ es generalmente elegida para poder ser factorizada de la forma $P\left(h^l\right) = \prod_i P\left(h_i^l\right)$ y solo un parámetro es requerido para cada caso en el que $P\left(h_i^l = 1\right)$ en el entorno de que se cuenten con unidades binarias.

Por otra parte, el gran avance en los procedimientos de entrenamiento de una red profunda vino dado por la presentación de las llamadas Redes de Creencia Profunda (DBN por sus siglas en inglés) que son de arquitectura similar a las SBNs (ver Figura

2.6.2) con una diferencia en la parametrización de las dos capas superiores obteniéndose la probabilidad como en la ecuación 2.6.3:

$$P\left(x, h^1, ..., h^l\right) = P\left(h^{l-1}, h^l\right)\left(\prod_{k=1}^{l-2} P\left(h^k|h^{k+1}\right)\right) P\left(x|h^1\right) \tag{2.6.3}$$

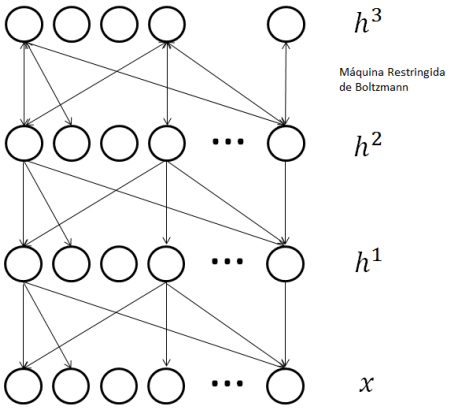


Figura 2.6.2: Arquitectura de una Red de Creencia Profunda, las dos capas superiores constituyen una Máquina Restringida de Boltzmann teniendo dependencias en ambas direcciones

La acción conjunta de las 2 capas superiores $P\left(h^{l-1}, h^l\right)$ es una Máquina de Boltzmann Restringida (RBM) cuyo comportamiento es el denotado por la ecuación 2.6.4:

$$P\left(h^{l-1}, h^l\right) \propto e^{b'h^{l-1} + c'h^l + h^{l'}Wh^{l-1}} \tag{2.6.4}$$

Este cambio explota y desarrolla la noción de un entrenamiento egoísta aplicándose a una capa de la red a la vez; esta cualidad es necesaria para el pre-entrenamiento de las capas ocultas de la red.

2.6.4. Redes neuronales convolucionales

Las redes neuronales convolucionales fueron inspiradas en la estructura del sistema visual humano. En el Neocognitron de Fukushima [23], el aprendizaje está basado en que cuando la salida de neuronas similares es aplicada o incide en lugares específicos de la capa superior de neuronas, una forma de varianza traslacional es obtenida. Actualmente, los sistemas de reconocimiento de patrones basados en redes neuronales convolucionales son los que tienen mejor desempeño (Ejemplo: reconocimiento de caracteres a mano alzada).

Las redes convolucionales están organizadas en 2 tipos de capas: las capas convolucionales y las capas submuestreadas. Cada capa tiene una estructura topográfica y está

asociada con una posición bidimensional respecto a un campo receptivo. En toda localidad de cada capa, existe un número diferente de neuronas con cada neurona relacionada con un conjunto de pesos sinápticos de entrada asociados con neuronas pertenecientes a una zona de la capa anterior. El mismo conjunto de pesos pero con una nueva zona de forma rectangular está asociado con neuronas distribuidas en diferentes lugares.

Un fan-in pequeño (cada neurona acepta solo un pequeño número de señales de entrada) en estas neuronas ayuda al método del gradiente descendente a actuar y propagarse a través de varias capas sin que su acción se desvanezca tanto como para no actuar, La estructura de conectividad jerárquica configura los parámetros de la red neuronal completa en una región favorable en la cual la optimización basada en el gradiente descendente funciona bien.

Un punto importante es que este tipo de redes aún utilizándose con pesos aleatorios en las primeras capas sin entrenamiento tienen un mejor desempeño que una arquitectura profunda tradicional completamente entrenada utilizando un aprendizaje supervisado; sin embargo, su desempeño mejora considerablemente comparada con una red convolucional completamente entrenada. Para finalizar debemos recordar que estas redes constituyeron el único caso de entrenamiento exitoso de una arquitectura profunda hasta antes de la llegada de la Redes de Creencia Profunda [4, 29].

2.6.5. Autoencoders

Los autoencoders han sido utilizados como bloques constructores de entrenamiento de arquitecturas profundas, donde cada nivel está asociado con un autoencoder que entrena inicialmente la capa independientemente de las demás [32, 31].

Un autoencoder es entrenado para codificar la entrada x en alguna representación correspondiente $c(x)$ tal que la entrada pueda ser reconstruida a partir de esta representación. La formulación utilizada generaliza el criterio del error cuadrático medio proponiendo la minimización de la verosimilitud logarítmica de la reconstrucción, entonces dada la representación $c(x)$ tenemos:

$$RE = -\log P(x|c(x))$$

Si la distribución de $x|c(x)$ es de tipo Gaussiano recuperamos el criterio del error cuadrático medio. Las entradas x_i son binarias o consideradas como probabilidades de tipo binomial, así, la función de costo está dada por la ecuación 2.6.5:

$$-\log P(x|c(x)) = -\sum_i x_i \log f_i\left(c(x)\right) + (1 - x_i)\log(1 - f_i(c(x))) \tag{2.6.5}$$

donde $f(\centerdot)$ es llamado decodificador y $f(c(x))$ es la reconstrucción producida por la red, en este caso un vector de números en el rango $(0, 1)$ obtenidos, por ejemplo, utilizando una función de tipo sigmoide.

Un autoencoder con una entrada n-dimensional y una dimensión de codificado de al menos n puede potencialmente aprender exclusivamente la función identidad dado que no existe una condición para evitar este tipo de codificación; sin embargo, este caso trata de evitarse. Algunas técnicas para el entrenamiento de autoencoders son:

1. Entrenamiento con gradiente descendente estocástico (Regularización de tipo l_2 de los parámetros)
2. Agregar ruido en la autocodificación
3. Establecer una condición de escasez (sparsity) en el código
4. Maximizar la verosimilitud del modelo generativo (denoising autoencoder)

Capítulo 3

Técnicas clásicas de identificación de sistemas no lineales

El aprendizaje profundo tiene su principal aplicación en la solución del problema de clasificación de patrones, a continuación se describen algunas de las técnicas que sirven de base para la constitución de una arquitectura profunda que funcione como clasificador; estos algoritmos constituyen el pilar fundamental para atacar el problema de regresión e identificación de la dinámica de sistemas.

3.1. Regresión logística

La regresión logística es un modelo de clasificación lineal de tipo probabilístico. La estructura del algoritmo está parametrizada por una matriz de pesos W y un vector de bias b. La clasificación se realiza proyectando cada conjunto de datos sobre un conjunto de hiperplanos, cuya distancia a cada uno de ellos refleja la probabilidad de membresía a cada clase que está representada y corresponde a un hiperplano específico.

Esta descripción puede representarse matemáticamente como:

$$P\left(Y = i|x, W, b\right) = \text{softmax}_i\left(Wx + b\right) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}} \tag{3.1.1}$$

La salida del clasificador o su predicción es hecha simplemente tomando el argumento máximo del vector cuyo i-ésimo elemento es $P\left(Y = i|x\right)$.

$$y_{pred} = \text{argmax}_i P\left(Y = i|x, W, b\right) \tag{3.1.2}$$

En el caso de que la tarea sea la identificación de un sistema, la salida de la red simplemente será la combinación lineal de los pesos sinápticos W y el vector de bias b.

$$y_{pred} = Wx + b \tag{3.1.3}$$

Un problema fundamental para la optimización de cualquier modelo es la elección de una función de pérdida, el aprendizaje de los parámetros óptimos del modelo de clasificación se realiza con el propósito de minimizar esta función de costo. En el caso de la regresión logística multi-clase, es muy común usar la log-verosimilitud negativa como función minimizadora. Esto es equivalente a maximizar la verosimilitud del conjunto de datos D bajo el modelo parametrizado por el conjunto Θ. De esta manera se define la verosimilitud L y la función de pérdida l.

$$L\left(\Theta = \{W, b\}, D\right) = \sum_{i=0}^{|D|} \log\left(P\left(Y = y^{(i)} | x^{(i)}, W, b\right)\right) \tag{3.1.4}$$

$$l\left(\Theta = \{W, b\}, D\right) = -L\left(\Theta = \{W, b\}, D\right) \tag{3.1.5}$$

Existen muchas publicaciones relacionadas con el tópico de minimización ([33, 42]) y aunque se han propuesto distintos métodos para lograr este objetivo, el algoritmo de gradiente descendente, es por mucho, el método mas simple para minimizar funciones no lineales arbitrarias. Aquí se describirá el método de gradiente descendente estocástico con mini-batches.

Para fines de identificación dinámica de un sistema, no es posible definir una función de verosimiltud discreta como la presentada por la Ecuación 3.1.4. En su lugar se utiliza la función del error cuadrático medio definida por la Ecuación 3.1.6.

$$l\left(\Theta = \{W, b\}, D\right) = \frac{1}{N} \sum_{i=0}^{|D|} \left(Y^{(i)} - y^{(i)}\right)^2 = \tag{3.1.6}$$

3.2. Perceptrón multicapa

Una Perceptrón multicapa (MLP) puede ser interpretada como una extensión del algoritmo de regresión logística donde primero la entrada es transformada utilizando una transformación no lineal Φ. El propósito de esta transformación es proyectar los datos de entrada a un espacio donde sean linealmente separables. Esta capa intermedia es conocida como capa oculta y es característica de una MLP poseer dos o mas de ellas. Una capa oculta única es suficiente para hacer de una MLP un aproximador universal [26]. Sin embargo, como se ha señalado anteriormente existen diversas ventajas al usar muchas capas ocultas, es decir, seguir la filosofía del aprendizaje profundo.

Formalmente, una MLP constituye una función $f : R^D \rightarrow R^L$,donde D es el tamaño del vector de entrada x y L es el tamaño del vector de salida $f(x)$. Si se consideran p capas ocultas entonces la salida queda determinada por la ecuación 3.2.1:

$$f(x) = \psi_p(b^{(p)} + W^{(p)}(\psi_{p-1}(b^{(p-1)} + W^{(p-1)}(\psi_{p-2}(...)))))) \tag{3.2.1}$$

con vectores de bias $b^{(0)}, ..., b^{(p)}$ y matrices de pesos sinápticos $W^{(0)}, ..., W^{(p)}$. Las funciones $\psi_{p-1}, ..., \psi_0$ son llamadas funciones de activación siendo las mas usuales la función $\tan h(a) = (e^a - e^{-a})/(e^a + e^{-a})$ y la función sigmoide $\text{sigm}(a) = 1/(1 + e^{-a})$. De esta manera el vector de entrada x ingresa al sistema formando la salida $\psi_0(b^{(0)} + W^{(0)}x)$, a este valor se le considera la salida de la primer capa oculta de la red, este valor se propaga a la siguiente capa sustituyendo al vector x por el vector ψ_i correspondiente.

Como en el caso de regresión logística, el valor esperado del sistema y_{pred} se realiza definiendo la función ψ_p como sigmoide para clasificación o la función identidad con fines de identificación de sistemas.

3.3. Método de gradiente descendente estocástico

Es un algoritmo simple en el cual repetidamente se realizan pequeños pasos en dirección decreciente sobre la superficie de error definida por una función de pérdida dada por algunos parámetros. Para este propósito se considera que los datos de entrenamiento se encuentran evaluados dentro de la función de pérdida. El pseudocódigo de este algoritmo puede escribirse como:

```
mientras True:
    perdida = f(parametros)
    gradiente = ...
    parametros = parametros − tasa_aprendizaje ∗ gradiente
    si <condicion_paro>
    regresar parametros
```

El algoritmo de gradiente descendente estocástico trabaja exactamente con los mismos principios que el método de gradiente descendente ordinario, pero trabaja mas rápidamente ya que estima el gradiente de la función de pérdida a partir de solamente algunas muestras de datos durante un instante de tiempo determinado en lugar de evaluar el conjunto de entrenamiento completo. En su forma mas simple, se estima el gradiente utilizando solamente un ejemplo por evento de estimación.

```
por cada pareja (x_i, y_i):
    perdida = f(parametros, x_i, y_i)
    gradiente = ...
    parametros = parametros − tasa_aprendizaje ∗ gradiente
```

`si` $<condicion\_paro>$

`regresar` $parametros$

La variante del método de gradiente estocástico más utilizada y recomendada para su utilización en técnicas de aprendizaje profundo, es el método de gradiente descendente estocástico por minibatches. En este caso se usa mas de un ejemplo de entrenamiento para cada estimación del gradiente, donde un factor crucial, será el número de ejemplos que contendrá cada conjunto actualizador de parámetros. Esta técnica reduce la varianza en la estimación del gradiente, y usualmente hace mejor uso de la organización jerárquica de la memoria de las computadoras modernas [6].

`por cada conjunto de datos (`x_{batch}, y_{batch}`) del conjunto de batches válidos:`

$perdida = f(parametros, x_{batch}, y_{batch})$

$gradiente = ...$

$parametros = parametros - tasa\_aprendizaje * gradiente$

`si` $<condicion\_paro>$

`regresar` $parametros$

Con este algoritmo existe un intercambio o balance entre el efecto reductivo de la varianza y la elección del tamaño del minibatch B. El factor de reducción de la varianza mejora notoriamente cuando se incrementa B de 1 a 2 , pero la mejora conseguida, rápidamente decae conforme se incrementa el tamaño de B. Con B grandes, el tiempo es desperdiciado reduciendo la varianza en conjunto de datos generados por estimador del gradiente, pero se cree que tiempo sería mas provechoso si se gastase en pasos adicionales de actualización paramétrica[9].

Un B óptimo es dependiente del modelo, del conjunto de datos y del hardware utilizado para la implementación del algoritmo y puede tener valores desde 1 o 2 hasta varios cientos, la elección del mismo se realiza de manera arbitraria.

Si se está entrenando un modelo utilizando un número fijo de épocas, el tamaño del minibatch se vuelve realmente importante porque controla el número de actualizaciones que se realizan a los parámetros. Entrenar el mismo modelo por 10 épocas usando un tamaño de batch de 1 tiene resultados completamente diferentes comparados a los obtenidos durante las mismas 10 épocas pero con un tamaño de batch de por ejemplo 20.

3.4. Método supervisado con restricción de paro temprano

El paro temprano tiene como objetivo evitar el sobreajuste monitoreando el desempeño del modelo sobre un conjunto de validación. Un conjunto de validación es un conjunto de

ejemplos que nunca son usados durante la etapa de entrenamiento, pero que tampoco forman parte del conjunto de prueba. Los ejemplos de validación son considerados como representativos de los ejemplos de prueba que serán utilizados en el futuro. Si el desempeño del modelo cesa de mejorar en el conjunto de validación, o incluso si presenta un desempeño menor a un umbral determinado durante el proceso de optimización, entonces se implementa una heurística que detiene el algoritmo de optimización.

La elección de la heurística que tome la decisión de cuando parar depende mucho de la aplicación, pero una estrategia muy común es la basada en el incremento geométrico de un parámetro llamado paciencia. El pseudocódigo de este algoritmo es el siguiente:

```
mientras (epoca < epoca_total) y (salir = falso):
    epoca = epoca + 1
    por cada minibatch en el conjunto de entrenamiento:
        perdida = f(parametros, x_batch, y_batch)
        gradiente = ...calculagradiente
        parametros = parametros − tasa_aprendizaje ∗ gradiente
        iteracion = (epoca − 1) ∗ numero_batches + indice_batch_actual
        si iteracion %
            costo_validacion = calcula_costo...
            si costo_validacion < mejor_costo
                si costo_validacion < mejor_costo ∗ 0,95
                    paciencia = paciencia ∗ 1,1
                mejor_costo = costo_validacion
                mejores_parametros = parametros_actuales
        si paciencia <= iteracion
            salir = verdadero
regresar parametros
```

El porcentaje de mejora permitida y el porcentaje que se aumenta en la paciencia son valores propuestos durante la implementación y deberán ser cambiados dependiendo de las necesidades de la aplicación.

3.5. Validación cruzada

La validación cruzada es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición hecha entre los datos

de validación y de prueba [10, 26]. Consiste en repetir y calcular la media aritmética obtenida de las funciones de costo entre diferentes particiones propuestas.

El método de retención o hold out method consiste en dividir en dos conjuntos los datos muestra, se realiza un análisis de un subconjunto llamado conjunto de entrenamiento y se valida el análisis en el otro subconjunto también llamado conjunto de prueba. Si tomamos una muestra independiente como datos de validación , normalmente el modelo no se ajustará con igual exactitud a los datos de prueba que a los de entrenamiento. Esta característica se conoce como sobreajuste y ocurre cuando el tamaño de los datos de entrenamiento es pequeño o cuando el número de parámetros del modelo es demasiado grande.

3.5.1. Validación cruzada de k iteraciones

Los datos muestra son divididos en k subconjuntos. Por cada iteración uno de los subconjunto se utiliza como datos de prueba y el resto como datos de entrenamiento. El proceso de validación se repite k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se calcula la media aritmética de los resultados de cada iteración para obtener un único resultado, ver Figura 3.5.1.

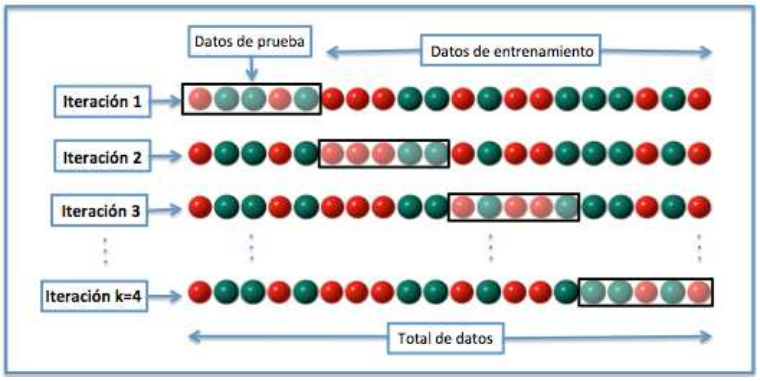


Figura 3.5.1: Validación cruzada de k iteraciones con $k = 4$

3.5.2. Validación cruzada aleatoria

Este método consiste en dividir aleatoriamente el conjunto de datos de entrenamiento y el conjunto de datos prueba (ver Figura 3.5.2). Para cada división la función de aproximación se ajusta a partir de los datos de entrenamiento y se calcula los valores de salida para el conjunto de datos de validación. El resultado final corresponde a la media aritmética de los valores obtenidos para cada una de las divisiones.Un inconveniente de este método es que hay algunas muestras que quedan sin evaluar y otras que se evalúan más de una vez.

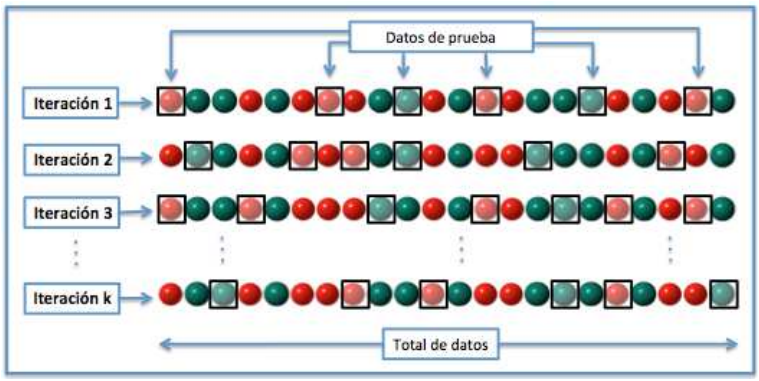


Figura 3.5.2: Validación cruzada aleatoria con k iteraciones

El objetivo de la validación cruzada consiste en estimar el nivel de ajuste de un modelo a un cierto conjunto de datos de prueba independientes de los utilizados para entrenar el modelo. Estas medidas obtenidas pueden ser utilizadas para estimar cualquier medida cuantitativa de ajuste apropiada para los datos y el modelo. Por ejemplo, en un modelo basado en clasificación binaria, cada muestra se etiqueta como correcta o incorrecta, de forma que en este caso, se puede usar "tasa de error de clasificación" para resumir el ajuste del modelo. Así mismo, se podrían utilizar otras medidas como el valor predictivo positivo.

3.6. Identificación de sistemas no supervisada

El problema de identificación de sistemas o problema de regresión tiene su principal exponente en el aprendizaje supervisado [42, 33], en el caso no supervisado, la idea básica es dividir el espacio muestra en varios subespacios, donde los datos con propiedades similares son reunidos. El procedimiento es expandido hacia el conjunto de datos completo, incluyendo los datos de salida.

La meta de la optimización no lineal es buscar el punto, en donde la evaluación de una función objetivo predefinida resulta en un valor máximo o mínimo considerando algunas restricciones las cuales son dadas mediante ecuaciones apropiadas. De acuerdo al valor de la función de costo J, es posible establecer el mejor candidato para la solución. Los métodos de aprendizaje exploran el espacio de soluciones y finalmente eligen el candidato que de la mejor solución.

En el caso de que se trate de obtener una solución basada en mediciones u observaciones, la función objetivo es siempre una función de observaciones. Por ejemplo considere que se busca una función f que minimize la función objetivo J:

$$J = E\left\{(f(u) - y)^2\right\}$$

En este caso, J está definida por pares de datos (u, y), que son generados por un proceso son distribución estadística D. En este caso es posible tratar el problema de minimización utilizando la ecuación:

$$\hat{J} = \frac{1}{n} \sum_{k=1}^{n} (f(u) - y)^2 \tag{3.6.1}$$

La función $\hat{J}$ está basada en un número finito de ejemplos N tomados de un conjunto distribuido de acuerdo con D. Los métodos para resolver este problema pueden ser divididos en 3 grupos:

Métodos de aprendizaje supervisados: Están basados en un conjunto de datos $u \in U$, $y \in Y$ que es usado para establecer una transformación, es decir, una función f de un conjunto dado de funciones la cual transforma los datos de entrada en funciones de salida. La función resultante es entonces evaluada de acuerdo a un criterio predefinido, el cual consiste de la diferencia entre los datos transformados y los datos de salida del conjunto original.

Una de las posibles técnicas utilizadas para la obtención de una solución es la optimización utilizando el método del gradiente descendente.

Métodos de aprendizaje no supervisados. Estos métodos refieren de los métodos de aprendizaje supervisado en que los datos de salida no son diferenciados a priori, por lo que todos los datos son tratados como entradas. El aprendizaje no supervisado representa un método donde el modelo es ajustado a los datos de una manera óptima, el resultado del aprendizaje modela la distribución de los datos en el espacio presentado en el problema.

Los métodos de aprendizaje no supervisados son utilizados durante el preprocesamiento de los datos antes de ser utilizados por algoritmos de característica supervisada. Estos algoritmos también son muy utilizados en la compresión de datos. Todos los métodos son implícita o explícitamente basados en la distribución de probabilidad en el espacio definido por las variables de entrada.

Métodos de aprendizaje reforzado: Estos métodos explotan la información de la calidad del modelo resultante, aunque la información de la salida de referencia basada en alguna entrada no este disponible. Los algoritmos son mayoritariamente usados en el desarrollo de estrategias a largo plazo, en donde una referencia o medida de la salida no se encuentra de manera explícita.

3.7. Topología de identificación

Además del problema de clasificación de patrones, la siguiente tarea que ha sido históricamente atacada por medio de modelo y algoritmos de aprendizaje donde convergen la teoría de sistemas y la inteligencia artificial es la identificación de sistemas dinámicos.

Para probar la tarea de identificación neuronal utilizando arquitecturas profundas sujetas a procesos de preentrenamiento, se considera una clase de sistemas representados por ecuaciones en diferencias de forma:

$$y(k) = f\left(y(k-1), ..., y(k-n), u(k-1)\right)$$

donde $y(k)$ representa la evolución de la salida del sistema que depende de n salidas anteriores y $u(k-1)$ es el valor de alguna señal que se encuentra interactuando activamente con el sistema.

En el presente trabajo se utiliza un modelo de tipo serie-paralelo para efectuar el proceso de identificación, esta configuración hace uso de la salida del sistema real para el calculo y actualización de los pesos sinápticos.

En la Figura 3.7.1 se ejemplifica el proceso de identificación considerando una planta de segundo orden, la cual depende directamente de los valores de su salida en dos instantes anteriores de tiempo, aquí, la salida del sistema real se denota por y_p y el valor de identificación del modelo se representa con y.

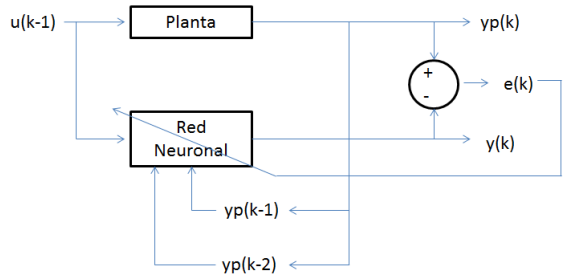


Figura 3.7.1: Topología del sistema de identificación utilizado

Capítulo 4

Elección de hiperparámetros en la identificación de sistemas no lineales

4.1. Aprendizaje de Boltzmann

El aprendizaje de Boltzmann es un algoritmo de aprendizaje estocástico derivado de ideas basadas en la mecánica estadística. Una red neuronal basada en este principio se le llama Máquina de Boltzmann (BM).

En una Máquina de Boltzmann las neuronas constituyen una estructura recurrente y operan de manera binaria, por ejemplo, pueden estar en un estado "encendido" denotado por $+1$ o un estado "apagado" descrito por un -1. Está máquina está caracterizada por una función de energía la cual está determinada por los estados ocupados por cada neurona individual, por ejemplo, una función de energía característica es la denotada por la ecuación 4.1.1:

$$E = -\frac{1}{2}\sum_{j}\sum_{k} w_{kj}x_k x_j \text{ con } j \neq k \tag{4.1.1}$$

donde x_j es el estado de la neurona j y w_{kj} es el peso sináptico que relaciona a la neurona j con la neurona k. En una Máquina de Boltzmann no se considera el caso de auto-retroalimentación, la máquina opera escogiendo una neurona al azar en algún punto del proceso de aprendizaje, entonces cambia el estado de la neurona k del estado x_k al estado -x_k con alguna temperatura T, este cambio en el estado de la neurona ocurre con una probabilidad dada por la ecuación 4.1.2:

$$P\left(x_k \to -x_k\right) = \frac{1}{1 - \exp\left(-\triangle E_k/T\right)} \tag{4.1.2}$$

donde $\triangle E_k$ es el cambio de energía en la red neuronal. La temperatura T no corresponde a una temperatura física real sino a una medida del desorden o nivel de entropía

existente en la red. Si el cambio de estado de las neuronas se elige como el de máxima probabilidad, el sistema alcanzará el equilibrio térmico.

Las neuronas se clasifican en dos grupos funcionales: visibles y ocultas. Las neuronas visibles proveen una interfase entre la red y el ambiente en el cual operan, mientras que las neuronas ocultas actúan libremente sin interactuar directamente con el entorno. La Máquina de Boltzmann tiene dos modos de operación que deben de considerarse:

1.- Condición de anclaje: Las neuronas visibles están ancladas en estados específicos determinados por los estímulos del medio ambiente.

2.- Condición de libertad: Tanto neuronas visibles como ocultas actúan libremente sin condiciones impuestas por el medio ambiente.

Sea p_{kj}^{+} la correlación entre los estados j y k con la configuración de la red en su condición de anclaje y sea p_{kj}^{-} la correlación pero en condición de libertad. Ambas correlaciones son promediadas sobre todos los posibles estados de la máquina cuando ésta se encuentra en equilibrio térmico. Entonces de acuerdo a la regla de aprendizaje de Boltzmann, el cambio de los pesos sinápticos durante la fase de aprendizaje es (ecuación 4.1.3):

$$\triangle w_{kj} = \eta \left(p_{kj}^{+} - p_{kj}^{-} \right) \text{ con } j \neq k \tag{4.1.3}$$

donde η es un parámetro conocido como tasa de aprendizaje. Note que ambas correlaciones p_{kj}^{+} y p_{kj}^{-} tienen valores en el rango de -1 a $+1$.

Una característica distintiva de la Máquina de Boltzmann es el uso de conexiones sinápticas simétricas entre las neuronas.

Durante la fase de entrenamiento las neuronas visibles son enclavadas en estados específicos determinados por el medio ambiente. Las neuronas ocultas siempre operan libremente y son usadas para encontrar características subyacentes contenidas dentro de los vectores de entrada, esto se observa cuando las neuronas ocultas presentan correlaciones de alto orden. Este método de aprendizaje constituye un procedimiento no supervisado y su objetivo principal es el modelado de una distribución de probabilidad especificada por los patrones aplicados en las entradas de las neuronas visibles.

Cuando en la entrada de la red neuronal no se cuenta con información completa del patrón de entrada, es decir, solamente un subconjunto incompleto de las neuronas visibles está siendo estimulado, la red logra determinar los valores de las entradas faltantes siempre y cuando haya sido entrenada apropiadamente dentro de la distribución de probabilidad.

Sea x el vector de estado de la máquina de Boltzmann con $x_i$denotando el estado de la neurona i. La conexión sináptica de la neurona i con la neurona j es w_{ij} cumpliendo la característica de simetría se cumple que $w_{ij} = w_{ji}$ para todo $\neq j$ y $w_{ii} = 0$ para todo i.

El uso de bias se permite en este proceso utilizando un peso w_{j0} con entrada igual a $+1$. La expresión para la energía de la red está dada por:

$$E(x) = -\frac{1}{2}\sum_i \sum_j w_{ji} x_i x_j \text{ con } i \neq j$$

La ecuación de Gibbs otorga una estimación de la probabilidad de que la red se encuentre en determinado estado x, esta probabilidad es calculada por:

$$P(X = x) = \frac{e^{-E(x)/T}}{Z}$$

donde Z es llamada función de partición y está determinada por:

$$Z = \sum_i e^{-E_i/T}$$

La probabilidad condicional de que una neurona j se encuentre configurada en un estado determinado x_j dado a priori un determinado estado particular de todas las demás neuronas es (ecuación 4.1.4):

$$P\left(X_j = x_j | \left\{X_i = x_i\right\}_{i=1}^{k},\ i \neq j\right) = \varphi\left(\frac{x}{T}\sum_{i \neq j} w_{ji} x_i\right) \tag{4.1.4}$$

donde:

$$\varphi(\nu) = \frac{1}{1 + e^{-\nu}}$$

cuyo comportamiento se aprecia en la Figura 4.1.1:

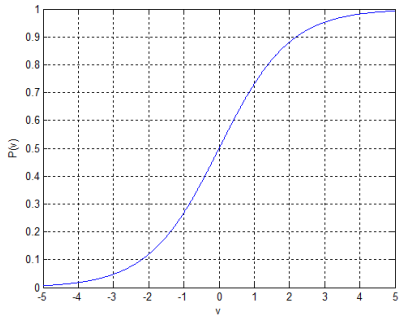


Figura 4.1.1: Probabilidad condicional de un estado particular de una neurona x_j dado un estado general de la red X

Dos suposiciones son hechas:

1. Cada vector de entrada persiste lo suficiente para permitir a la red alcanzar su equilibrio térmico.

2. No hay estructura en el orden en el que los patrones provenientes del medio ambiente son asignados a la entrada de las neuronas visibles de la red.

Un conjunto de pesos constituye un modelo perfecto de la estructura ambiental si da como salida la misma distribución de probabilidad en los estados de las neuronas visibles cuando estos operan en la fase de libertad que cuando funcionan durante la fase de anclaje. Esta característica generalmente es alcanzada utilizando un número lo suficientemente grande de neuronas ocultas [31].

4.1.1. Muestreo de Gibbs

Supongamos que tenemos una configuración arbitraria $\{x_1(0),\ x_2(0),\ ...\ ,\ x_k(0)\}$, se realiza el siguiente algoritmo de muestreo:

$x_1(1)$ es tomada de la distribución de x_1, dadas $x_2(0),\ x_3(0),\ ...\ ,\ x_k(0)$

$x_2(1)$ es tomada de la distribución de x_2, dadas $x_1(1),\ x_3(0),\ ...\ ,\ x_k(0)$

$x_l(1)$ es tomada de la distribución de x_l, dadas $x_1(1),\ x_2(1),\ ...\ ,\ x_{l-1}(1),\ x_{l+1}(0),\ ...\ ,\ x_k(0)$

$x_k(1)$ es tomada de la distribución de x_l, dadas $x_1(1),\ x_2(1),\ ...\ ,\ x_{k-1}(1)$

Este proceso constituye un esquema iterativo adaptativo. Después de n iteraciones se obtienen: $x_1(n),\ x_2(n),\ ...\ ,\ x_k(n)$

Existe una equivalencia entre los conceptos de la mecánica estadística y los utilizados en el estudio de redes neuronales de comportamiento estocástico:

Tabla 4.1: Equivalencia entre mecánica estadística y optimización combinatoria

Mecánica estadística	Optimización
Muestreo	Instancia del problema
Estado	Configuración
Energía	Función costo
Temperatura	Parámetro de control
Energía estado-referencia	Costo mínimo
Configuración estado-referencia	Configuración óptima

El proceso de muestreo de Gibbs da buenos resultados si el entorno en el cual es aplicado obedece los siguientes Teoremas:

1.- Teorema de convergencia: La variable aleatoria $x_k(n)$converge en las distribuciones verdaderas de probabilidad de x_k para $k = 1, 2, ..., K$ cuando n tiende a infinito, es decir:

$$\lim_{n\to\infty} P\left(x_k^{(n)} \leq x | x_k(0)\right) = F_{x_k}(x) \ \forall \ k = 1, 2, ..., K$$

donde $F_{x_k}(x)$ es la distribución marginal de probabilidad.

2.- Teorema de tasa de convergencia: La distribución de probabilidad conjunta de las variables aleatorias $x_1(n),\ x_2(n),\ ...\ ,\ x_k(n)$ converge a la verdadera distribución de probabilidad conjunta de $x_1,\ x_2,\ ...\ ,\ x_k$ con una tasa de acercamiento geométrica en n.

3.- Teorema de ergodicidad: Para cualquier función medible g de las variables aleatorias $x_1,\ x_2,\ ...\ ,\ x_k$ cuya esperanza matemática existe, se tiene:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} g\left(x_1(i),\ x_2(i),\ ...\ ,\ x_k(i)\right) \to E\left[g\left(x_1,\ x_2,\ ...\ ,\ x_k\right)\right] \text{ con probabilidad de 1}$$

Si el proceso de Gibbs es llevado a cabo por suficiente tiempo, la red alcanzará equilibrio térmico con temperatura T.

4.1.2. Proceso de enfriamiento simulado

1. Valor inicial: El valor de partida T_0 se escoge lo suficientemente alto para asegurar que todas las transiciones propuestas sean aceptadas y de esta manera evitar en lo posible los mínimos locales.

2. Decremento: Se realiza de manera general utilizando un descenso exponencial cuyos cambios son relativamente pequeños. Un ejemplo es $T_k = \alpha T_{k-1}$ para $k = 1, 2, ...$ donde α es un poco menor a 1. En cada temperatura se realizan suficientes iteraciones de enfriamiento por ejemplo 10.

3. Valor final: El proceso se detiene cuando el número deseado de elementos de entrada permitido no es alcanzado para tres temperaturas consecutivas.

En la Máquina de Boltzmann se realiza el proceso de templado para una secuencia finita de temperaturas $T_0, T_1, ..., T_{final}$. La temperatura inicial T_0 es alta para permitir que el equilibrio térmico sea alcanzado rápidamente. De esta manera obtenemos gradualmente la temperatura T_{final}, cuando esta temperatura es alcanzada las neuronas habrán llegado a sus distribuciones de probabilidad deseadas.

Regla de aprendizaje de Boltzmann

Un índice de desempeño muy utilizado es la función de verosimilitud. La meta del aprendizaje de Boltzmann es maximizar este parámetro.

Considere las siguientes definiciones:

1. x vector de estado
2. x_α vector de estado de las neuronas visibles
3. x_β vector de estado de las neuronas ocultas

Sea $\mathcal{F}$ el conjunto de ejemplos de entrenamiento de la distribución de probabilidad de interés. Se supone que todos los valores son bivaluados y pueden utilizarse mas de una vez. Existen dos fases de operación de la máquina de Boltzmann.

1. Fase positiva: La red opera en su condición de anclaje en las neuronas visibles.
2. Fase negativa: La red puede operar libremente sin la influencia de ninguna entrada derivada del medio ambiente.

Dado un vector de pesos w. La probabilidad de que las neuronas visibles tengan un estado x_α es $P\left(X_\alpha = x_\alpha\right)$. Si todos los posibles valores de $x_\alpha \in \mathcal{F}$ se suponen estadísticamente independientes, las distribución de probabilidad, la distribución de probabilidad es la distribución de tipo factorial $\prod_{x_\alpha \in \mathcal{F}} P\left(X_\alpha = x_\alpha\right)$. Para formular la función correspondiente a la log-verosimilitud $L(w)$ se toma esta distribución y se trata al peso w como un vector desconocido (ecuación 4.1.5).

$$L(w) = \log \prod_{x_\alpha \in \mathcal{F}} P\left(X_\alpha = x_\alpha\right) = \sum_{x_\alpha \in \mathcal{F}} \log P\left(X_\alpha = x_\alpha\right) \tag{4.1.5}$$

Cabe recordar que $P(X = x) = Z^{-1} e^{-E(x)/T}$

Por definición x es la combinación conjunta de x_α perteneciente a las neuronas visibles y x_β perteneciente a las neuronas ocultas. Así las probabilidad de encontrar las neuronas visibles en un estado x_α dado cualquier estado de las neuronas ocultas x_β está dado por:

$$P(X_\alpha = x_\alpha) = \frac{1}{Z} \sum_{x_\beta} e^{-E(x)/T}$$

donde Z está definida por:

$$Z = \sum_{x} e^{-E(x)/T}$$

Así sustituyendo en la expresión para la log-verosimilitud de los pesos sinápticos se obtiene:

$$L(w) = \sum_{x_\alpha \in \mathcal{F}} \log \frac{1}{Z} \sum_{x_\beta} e^{-E(x)/T} = \sum_{x_\alpha \in \mathcal{F}} \left(\log \sum_{x_\beta} e^{-E(x)/T} - \log \sum_{x} e^{-E(x)/T} \right)$$

Diferenciando $L(w)$ respecto a w_{ij} obtenemos el siguiente resultado:

$$\frac{\partial L(w)}{\partial w_{ji}} = \frac{1}{T} \sum_{x_\alpha \in \mathcal{F}} \left[\sum_{x_\beta} P\left(X_\beta = x_\beta | X_\alpha = x_\alpha\right) x_j x_i - \sum_{x} P\left(X = x\right) x_j x_i \right]$$

Para simplificar se definen dos conceptos de acuerdo a las dos etapas del proceso de aprendizaje que habían sido descritas anteriormente (ecuaciones 4.1.6 y 4.1.7):

$$p_{ji}^{+} = < x_j x_i >^{+} = \sum_{x_\alpha \in \mathcal{F}} \sum_{x_\beta} P\left(X_\beta = x_\beta | X_\alpha = x_\alpha\right) x_j x_i \tag{4.1.6}$$

$$p_{ji}^{-} = < x_j x_i >^{-} = \sum_{x_\alpha \in \mathcal{F}} \sum_{x} P\left(X = x\right) x_j x_i \tag{4.1.7}$$

donde p_{ji}^{+} es la correlación entre los estados i y j con la red operando en su fase positiva o de anclaje, mientras que p_{ji}^{-} es la correlación de los estados de las neuronas i y j en la fase negativa o de libertad.

Así podemos simplificar la derivada de la log-verosimilitud como:

$$\frac{\partial L(w)}{\partial w_{ji}} = \frac{1}{T} \left(p_{ji}^{+} - p_{ji}^{-} \right)$$

Recordemos que la meta es maximizar la función de $L(w)$. Podemos utilizar el método del gradiente descendente para lograr la meta deseada utilizando:

$$\triangle w_{ij} = \eta \left(p_{ji}^{+} - p_{ji}^{-} \right)$$

donde η es un parámetro llamado tasa o rapidez de aprendizaje. Esta ley de aprendizaje provoca que $\triangle L(w)$ sea siempre positivo otorgándole a este parámetro un crecimiento sostenido. A esta serie de ecuaciones es lo que se conoce como regla de aprendizaje de Boltzmann.

Características de la regla de aprendizaje

Los pesos sinápticos de una Máquina de Boltzmann son ajustados usando solamente observaciones disponibles sobre 2 condiciones diferentes: la primera una fase llamada positiva o de anclaje y una segunda fase de comportamiento libre o negativa. En este contexto se puede hacer la siguiente distinción:

p_{ji}^{+} representa una regla Hebbiana de aprendizaje

p_{ji}^{-} representa un término de desaprendizaje

Puede verse como una generalización del *repeated forgetting and relearning rule* para el caso de redes simétricas sin neuronas ocultas.

La razón de concebir una etapa negativa durante el proceso de aprendizaje es que la dirección del descenso máximo en el espacio energético no es la misma que la dirección del máximo ascenso en el espacio de probabilidad lo que ayuda a evitar y en último caso escapar de la presencia de mínimos locales durante la fase de entrenamiento alcanzando el resultado óptimo en muchos casos de estudio. Dado que la Máquina de Boltzmann requiere que las neurona ocultas conozcan la diferencia entre las dos fases de entrenamiento, existe una red externa de estímulos que les señala a las neuronas cuando están siendo estimuladas por estímulos del medio ambiente.

Las principales dificultades de la máquina de Boltzmann es el tiempo computacional notoriamente incrementado comparado con el tiempo asociado a topologías determinísticas y también la sensibilidad que presente frente a errores estadísticos [26, 31].

4.2. Técnicas de ajuste de hiperparámetros de una red neuronal con arquitectura profunda

En [37] se dice que el algoritmo de backpropagation es en realidad un método de optimización por medio de gradiente descendente y por lo tanto no hay garantía de que el mínimo absoluto sea ser alcanzado. A pesar de este hecho teóricamente aceptado, el método de propagación hacia atrás funciona adecuadamente en muchas aplicaciones convergiendo al verdadero mínimo global, o al menos hace posible conseguir alcanzar algún criterio de paro del algoritmo.

En diversos textos [37] se recomienda no definir redes neuronales con mas parámetros que el número de ejemplos de entrenamiento disponibles. La cualidad de la solución obtenido generalmente depende de varios factores entre los que destacan:

1. La complejidad de la aproximación requerida.
2. El tamaño de la red en relación con el tamaño requerido para una solución óptima.
3. El grado de ruido de los datos presentados

Un problema clásico es aquel que minimiza el tamaño de la red neuronal manteniendo un buen desempeño. Se puede acceder a esta solución empleando dos técnicas básicas.

- Crecimiento de la red: Comenzar con un una red de arquitectura pequeña y añadir una nueva neurona o una nueva capa solo cuando somos incapaces de cumplir con las especificaciones de diseño.

- Poda de la red: Comenzar con una red lo suficientemente grande que satisfaga los criterios de diseño y proceder a podarla eliminando pesos sinápticos de acuerdo a cierto criterio de desempeño.

4.2.1. Elección aleatoria de hiper-parámetros

En [11] se define la optimización de hiper-parámetros como el problema de optimizar una función de costo sobre un espacio de configuración estructurado por un grafo. En este trabajo a su vez, se estudian espacios de configuración de tipo árbol, entendiéndose como aquellos en los que algunas variables tipo hoja (e.g. el número de unidades en una DBN) solamente están bien definidas cuando se especifican determinísticamente las variables de tipo nodo (e.g el número de capas de la red).

Un algoritmo de optimización no solamente debe de optimizar variables que son discretas, ordinales o continuas, sino que además debe simultáneamente escoger que variables optimizar.

En [11], se menciona que el objetivo final de un algoritmo de aprendizaje típico A es encontrar una función f que minimice una pérdida esperada $L(x; f)$ sobre ejemplos i.i.d. (independent and identically distributed) x de una distribución natural G_x. El algoritmo de entrenamiento a veces depende de unas variables conocidas como hiper-parámetros λ, y el algoritmo real de aprendizaje es el que se obtiene después de escoger este vector λ, que puede ser denotado como A_λ, Y $f = A_\lambda\left(X^{(train)}\right)$ para algún vector de entrenamiento $X^{(train)}$.

Lo que es necesario en la práctica es una manera de escoger λ que minimice el error de generalización $E_{x\sim G_x}\left[L\left(x; A_\lambda\left(X^{(train)}\right)\right)\right]$. Así, la optimización de los hiper-parámetros se reduce a:

$$\lambda^{(*)} = \text{argmin}_{\lambda\in\Lambda} E_{x\sim G_x}\left[L\left(x; A_\lambda\left(X^{(train)}\right)\right)\right]$$

En general, no existen algoritmos eficientes para efectuar dicha optimización. Respecto a la esperanza sobre G_x, se emplea la técnica llamada validación cruzada para estimar una posible optimización. La validación cruzada es la técnica de reemplazar la esperanza matemática con un promedio sobre un conjunto de validación $X^{(valid)}$ cuyos elementos son tomados i.i.d. de G_x. La validación cruzada es imparcial siempre y cuando que $X^{(valid)}$ sea independiente que cualquier dato utilizado por A_λ. Entonces se puede reformular el problema de optimización de hiperparámetros como:

$$\lambda^{(*)} \approx \text{argmin}_{\lambda\in\Lambda}\text{mean}_{x\in X^{(valid)}}\left[L\left(x; A_\lambda\left(X^{(train)}\right)\right)\right]$$

$$\equiv \text{argmin}_{\lambda\in\Lambda}\Psi(\lambda) \tag{4.2.1}$$

$$\approx \text{argmin}_{\lambda \in \left\{\lambda^{(1)} \ldots \lambda^{(s)}\right\}} \Psi\left(\lambda\right) \equiv \hat{\lambda}$$

La ecuación 4.2.1 expresa el problema de optimización en términos de la función de respuesta de los hiper-parámetros Ψ. Conociendo en general muy poco acerca de la superficie de respuesta Ψ o del espacio de búsqueda Λ , la estrategia dominante para encontrar un buen conjunto λ es escoger algún número (S) de puntos prueba $\left\{\lambda^{(1)} \ldots \lambda^{(s)}\right\}$, evaluar $\Psi(\lambda)$ y regresar el $\lambda^{(i)}$ que mejor se comportó como $\hat{\lambda}$. El problema radica entonces en como escoger el conjunto de prueba $\left\{\lambda^{(1)} \ldots \lambda^{(s)}\right\}$ que generalmente se resuelve utilizando algoritmos de búsqueda por malla o búsqueda manual.

Si Λ es un conjunto indexado por K variables de configuración (tasa de aprendizaje, número de capas, número de neuronas en cada capa, etc.), entonces la búsqueda por malla requiere que se escojan un conjunto de valores para cada variable $\left(L^{(1)} \ldots L^{(K)}\right)$. En la búsqueda de malla el conjunto de prueba es formado ensamblando cada posible combinación de valores, así el número de pruebas en la malla es $S = \prod_{k=1}^{K} \left|L^{(k)}\right|$. Este producto sufre de la llamada maldición de dimensionalidad [30, 8] porque el número de valores aumenta exponencialmente con el número de hiper-parámetros.

Por otra parte, la búsqueda manual de hiper-parámetros es usada para identificar regiones en Λ que son prometedoras y se usa para desarrollar la intuición necesaria para escoger los conjuntos $L^{(k)}$. El principal inconveniente de este tipo de selección es la dificultad en producir resultados lo cual es muy importante para el progreso científico en el campo del aprendizaje de máquinas. A pesar de sus limitaciones existen varias razones por las cuales las búsquedas manual y por malla prevalecen:

1. La optimización manual le otorga a los investigadores algún grado de visión dentro de Ψ.

2. No existe barrera o limite para la optimización manual.

3. La búsqueda por malla es simple de implementar y la paralelización es trivial.

4. El mallado es confiable y pequeño para espacios dimensionalmente bajos.

Dado la gran complejidad que representa un mallado completo, se ha propuesto en [11] una búsqueda aleatoria de hiperparámetros teniendo selecciones independientes de un espacio uniforme del mismo espacio de configuración similar al proceso de malla con el objetivo de elegir $\left\{\lambda^{(1)} \ldots \lambda^{(s)}\right\}$. En general la selección aleatoria tiene prácticamente todas las ventajas de la búsqueda por malla y canjea una pequeña reducción en eficiencia en espacios de dimensión pequeña por una gran mejora en la eficiencia en espacios de dimensión alta.

En [11]se menciona además que en general las funciones Ψ tienen una dimensionalidad efectiva baja, esto es, son mas sensibles a cambios en algunas dimensiones que en otras.

Mallado contra aleatoriedad para la optimización de redes neuronales

Cuando se determina el desempeño de los algoritmos de aprendizaje, es útil tener en consideración la incertidumbre generada por el conjunto de valores escogidos para los hiper-parámetros. En [11] se distingue la diferencia entres los estimados de desempeño $\Psi^{(valid)} = \Psi$ y $\Psi^{(prueba)}$ basados en los conjuntos de prueba y validación.

$$\Psi^{(valid)} = \text{mean}_{x \in X^{(valid)}} \left[L\left(x; A_\lambda\left(X^{(train)}\right)\right)\right] \tag{4.2.2}$$

$$\Psi^{(prueba)} = \text{mean}_{x \in X^{(prueba)}} \left[L\left(x; A_\lambda\left(X^{(train)}\right)\right)\right] \tag{4.2.3}$$

Se debe definir la varianza de estos valores en función de los hiper-parámetros escogidos, utilizando la varianza de Bernoulli:

$$V^{(valid)}(\lambda) = \frac{\Psi^{(valid)}(\lambda)\left(1 - \Psi^{(valid)}(\lambda)\right)}{|X^{(valid)}| - 1} \tag{4.2.4}$$

$$V^{(prueba)}(\lambda) = \frac{\Psi^{(prueba)}(\lambda)\left(1 - \Psi^{(prueba)}(\lambda)\right)}{|X^{(prueba)}| - 1} \tag{4.2.5}$$

Para resolver la dificultad de escoger un conjunto de valores ganador, se reporta un promedio ponderado de todos los resultados del conjunto de prueba, cuya ponderación corresponde a la probabilidad de que ese $\lambda^{(s)}$ sea en realidad el mejor. En este punto de vista, la incertidumbre debida a que $X^{(valid)}$ sea una muestra finita de G_x hace que el puntaje del conjunto de prueba con el mejor modelo $\lambda^{(s)}$ sea una variable aleatoria z. Este puntaje z es modelado por un modelo Gaussiano cuyos S componentes tienen medias $\mu_s = \Psi^{(prueba)}\left(\lambda^{(s)}\right)$, varianzas $\sigma_s^2 = V^{(prueba)}\left(\lambda^{(s)}\right)$, y pesos w_s definidos por

$$w_s = P\left(Z^{(s)} < Z^{(s')}, \forall s' \neq s\right) \tag{4.2.6}$$

$$Z^{(i)} \sim N\left(\Psi^{(valid)}\left(\lambda^{(i)}\right), V^{(valid)}\left(\lambda^{(i)}\right)\right) \tag{4.2.7}$$

Para finalizar el desempeño z del mejor modelo en un experimento de S pruebas tiene media μ_z y error estándar σ_z^2.

$$\mu_z = \sum_{s=1}^{S} w_s \mu_s$$

$$\sigma_z^2 = \sum_{s=1}^{S} w_s\left(\mu_s + \sigma_s^2\right) - \mu_z^2 \tag{4.2.8}$$

El procedimiento para estimar pesos $w_s$es repetidamente seleccionar puntajes de validación hipotéticos $Z^{(s)}$ de una distribución Normal y se toma en cuenta que tan frecuente cada conjunto de prueba genera un puntaje ganador, entonces, w_s no necesita ser estimado muy precisamente. Esta técnica da un estimado más alto que la técnica tradicional de reportar el error de entrenamiento del mejor modelo en validación.

Hiper-parámetros mallados

La elección de hiper-parámetros que hacen en [11] para una Red Perceptron Multicapa es la siguiente considerando un aprendizaje por el método del gradiente descendente:

1. Número de capas ocultas: Se muestrea geométricamente de un espacio de 1 a 10.
2. Número de neuronas para cada capa: Se muestrea geométricamente de 5 a 1024 creándose un nuevo parámetro de este nivel por cada capa oculta.
3. Tasa de aprendizaje: Muestreada geométricamente de 0.001 a 1.
4. Tasa de momento: Muestreada geométricamente de 0.001 a 1.
5. Tipo de preprocesamiento: Ninguna, normalizada o utilizando la técnica de Principal Component Analysis (PCA).

En el caso requerido que es el Aprendizaje profundo se analiza el particular uso de DBNs (Deep Belief Networks).

- Número de capas de la red entre 1 y 10 con igual probabilidad.
- Para cada capa se escoge
  - número de unidades ocultas (uniformes logarítmicamente entre 128 y 4000)
  - número de iteraciones para divergencia contrastiva durante el pre-entrenamiento (distribución logarítmica uniforme entre 1 a 10000)
  - tasa de aprendizaje inicial para divergencia contrastiva (distribución logarítmica uniforme entre 0.0001 y 1)
- Tasa de aprendizaje para el ajuste fino de la red del aproximador final utilizando el método del gradiente descendente (distribución logarítmica uniforme entre 0.001 y 10)
- Tipo de regularización l_2 de las matrices de pesos por cada capa durante el ajuste fino que puede ser 0 con probabilidad de 0.5 o una distribución logarítmica uniforme de 10^{-7} a 10^{-4}

La búsqueda por malla resulta ineficaz debido a la gran cantidad de valores posibles para los hiperparámetros necesitando un enorme poder de cómputo para su cálculo.

4.2.2. Optimización no supervisada con técnicas de capa-egoísta

La técnica presentada en [40] responde a la pregunta que surge si se supone que el procedimiento de aprendizaje no supervisado en una Deep Belief Network puede ser extendido a la selección de los hiperparámetros de la red.

En [40] se usa como criterio de evaluación del modelo el llamado Error de Reconstrucción, el cual formalmente se obtiene estableciendo las unidades visibles v a algunas entradas muestra x, se calcula el valor de las unidades ocultas después de la consideración de $P(h|v)$, se propaga hacia atrás este resultado hacía las unidades visibles y se calcula la distancia Euclideana entre x y la configuración de las unidades visibles que ha sido obtenida.

Sea el vector W_θ^* los pesos de una Máquina Restringida de Boltzmann (RBM) entrenada con el conjunto de datos D utilizando los hiper-parámetros θ; entonces el Error de Reconstrucción está definido como:

$$L\left(D,\theta\right) =_{def} L\left(D,W_\theta^*\right) = \sum_{x\in D} \left\| x - g_{w_\theta^*} \circ f_{w_\theta^*}(x) \right\|^2 \tag{4.2.9}$$

El error de reconstrucción correspondiente a las l capas de la RBM puede ser directamente definido de la ecuación 4.2.9 como :

$$L\left(D,W_1,...,W_l\right) = \sum_{x\in D} \left\| x - G_l \circ F_l(x) \right\|^2 \tag{4.2.10}$$

El procedimiento para elegir el conjunto de hiper-parámetros utilizando la información proporcionada por el error de reconstrucción radica en seleccionar un número conveniente de neuronas para cada una de las capas de la red. El error de reconstrucción debería de disminuir conforme el número de neuronas aumenta por lo cual debe de utilizarse un parámetro de regularización para detener este proceso. Otra posibilidad es esperar hasta que el error de reconstrucción no decrezca significativamente de acuerdo a un umbral mínimo de cambio.

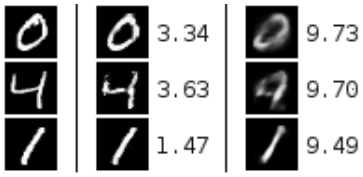


Figura 4.2.1: Ejemplo de dígitos reconstruidos utilizando una RBM con 60,000 ejemplos por cada época. Izquierda: Originales, Centro: Reconstrucción con 300 unidades ocultas, Derecha: Reconstrucción con 10 unidades ocultas.

Considérese el problema de clasificación MNIST descrito en la sección 5.4. En la Figura 4.2.1 se aprecia el cambio que se consigue aumentando el número de neuronas ocultas

en la segunda capa de una RBM, esto representa un error de reconstrucción mucho mas bajo que detendría el algoritmo de aprendizaje.

Capítulo 5

Método de autocodificación en la identificación de sistemas no lineales

5.1. Autoencoders

Un autoencoder toma como entrada un vector $x \in [0,1]^d$, y primero realiza un mapeo (utilizando un codificador) a una representación oculta $y \in [0,1]^{d'}$ por medio de una asignación determinística caracterizada por pesos sinápticos y bias, es decir:

$$y = s\left(Wx + b\right) \tag{5.1.1}$$

donde s es una no-linealidad, como por ejemplo una sigmoide. La representación oculta o latente y, también llamada código, es entonces mapeada de vuelta (utilizando un decodificador) a una reconstrucción z se la misma forma y tamaño que x a través de una transformación similar, es decir:

$$z = s\left(W'x + b'\right) \tag{5.1.2}$$

donde $'$ no indica una operación de transposición, y z debe ser interpretada como una predicción de x dado el código obtenido en y. La matriz de pesos W' del mapeo regresivo se puede restringir de manera opcional a ser simplemente la matriz transpuesta de la matriz de pesos de la etapa de codificación, es decir: $W' = W^T$, esta asignación se le conoce como asociación de pesos. Los parámetros de este tipo de modelo (llamados W, b, b' y, si no se usan pesos asociados, también W') son optimizados tal que el error de reconstrucción promedio es minimizado. El error de reconstrucción puede ser medido de varias maneras, dependiendo de algunas suposiciones de distribución de probabilidad apropiadas en la entrada condicionada al código, es decir, usando el tradicional error cuadrático $L\left(x,z\right) = \left\|x - z\right\|^2$,o si la entrada es interpretada ya sea como vectores

binarios o como vectores con probabilidades binarias utilizando la entropía cruzada de reconstrucción definida como:

$$L_H(x, z) = -\sum_{k=1}^{d} [x_k \log z_k + (1 - x_k) \log (1 - z_k)] \tag{5.1.3}$$

Lo que se desea es que el código obtenido y sea una representación distribuida que capture las coordenadas sobre los factores principales de variación en los datos (similarmente a como la proyección de los componentes principales captura los factores principales de variación). Generalmente y es vista como una compresión con pérdida de x y por lo tanto no puede ser interpretada como una buena compresión (con pérdidas pequeñas) para todo x, así el entrenamiento resulta en una compresión que funciona particularmente bien para los ejemplos de entrenamiento, pero no para entradas arbitrarias. Estas son las principales limitaciones en el uso de los autoencoders y como estos generalizan, estas estructuras resultan en un error de reconstrucción pequeño sobre ejemplos de prueba pertenecientes a la misma distribución probabilística de ejemplos de entrenamiento, pero tienen como salida un error alto de reconstrucción usando entradas de otras distribuciones.

5.1.1. Consideraciones iniciales en el entrenamiento de autoencoders

Si existe una capa oculta lineal (el código) y se usa un criterio de error cuadrático medio para entrenar a la red, entonces las k unidades ocultas aprenden a proyectar la entrada en el span de las primeras k componentes principales de los datos. Si la capa oculta es no lineal, el autoencoder se comporta diferente del análisis de componentes principales [6], con la habilidad de capturar aspectos multi-modales de la distribución de los datos de entrada.

Un evento que puede ocurrir durante el entrenamiento de los autoencoders es que si no existen otras restricciones además de minimizar el error de reconstrucción entonces un autoencoder con n entradas y una dimensión de codificación de al menos n podría potencialmente aprender solamente la función identidad, por lo cual muchos autoencoders serían inútiles. Algunos experimentos sugieren que en práctica, cuando se entrenan utilizando gradiente descendente estocástico, los autoencoders no lineales con mas unidades ocultas que entradas (llamados sobrecompletos) proporcionan representaciones útiles [32].

Se ha sugerido que para alcanzar una buena reconstrucción de entradas con valores continuos, un autoencoder de una sola capa oculta con unidades ocultas no lineales necesitan inicialmente pequeños pesos en la capa de codificación (para garantizar que el aprendizaje comience en la etapa aproximadamente lineal de las no linealidades) y pesos de magnitud considerable en la capa de decodificación.

Como la regularización implícita o explícita hace difícil alcanzar automáticamente soluciones con pesos de magnitud elevada, el algoritmo de optimización encuentra codificaciones que solamente trabajan bien para ejemplos similares a los pertenecientes al conjunto de entrenamiento, lo cual es la característica principal de los autoencoders. Esto significa que la representación está explotando regularidades estadísticas presentes en el conjunto de entrenamiento, en lugar de aprender a replicar la función identidad.

5.1.2. Autoencoders ruidosos

La idea principal del uso de los autoencoders ruidosos es simple. Para forzar a la capa oculta a descubrir características robustas y prevenir que la red simplemente aprenda la función identidad, entrenamos el autoencoder de tal manera que reconstruya la entrada desde una versión corrupta de ella.

Un autoencoder ruidoso es una versión estocástica del autoencoder tradicional. Intuitivamente, un autoencoder ruidoso realiza dos cosas: trata de codificar la entrada (preserva información de la entrada), e intenta deshacer el efecto de un proceso de corrupción estocásticamente aplicado a la entrada del modelo.

En la literatura, el proceso de reconstrucción estocástico consiste en fijar aleatoriamente algunas de las entrada como cero (inclusive la mitad del total de ellas). Así, el autoencoder ruidoso está tratando de predecir los valores corruptos de los valores no corruptos. Esto es importante porque si el autoencoder es capaz de predecir cualquier subconjunto de variables del resto, esto es una condición suficiente para capturar completamente la distribución conjunta de un conjunto de vectores de entrada de entrenamiento (al igual que el muestreo de Gibbs).

5.2. Autoencoders ruidosos apilados

Los autoencoders ruidosos pueden ser apilados para formar una red de arquitectura profunda simplemente alimentando la representación latente (o código de salida) del autoencoder ruidoso ubicado en la capa inferior como entrada a la capa actual. El preentrenamiento no supervisado de tal arquitectura es hecho una capa a la vez. Cada capa es entrenada como un autoencoder ruidoso minimizando el error de reconstrucción de su entrada (la cual es el código de salida de la capa anterior). Una vez que las primeras k capas son entrenadas, se puede realizar el entrenamiento de la capa $k + 1$ debido a que hasta ese momento se hace posible el cálculo del código o representación latente de la capa previa.

Una vez que todas las capas son preentrenadas, la red pasa a través de una segunda etapa conocida como ajuste fino. Generalmente, para esta última etapa, se considera un ajuste fino supervisado donde lo que se desea es minimizar el error de predicción durante una tarea supervisada. Para ello, para como objeto de clasificación o identificación, se

añade una capa de regresión logística a la última capa de la red acorde a lo presentado en la Sección 3.1(más precisamente sobre el código de salida de la última capa).

El entrenamiento supervisado se realiza de la forma en como se entrena una perceptrón multicapa, es decir, utilizando métodos usuales de gradiente descendente.

Se puede analizar a un autoencoder ruidoso apilado como si tuviera dos partes diferenciadas, la primera se trata de una lista de autoencoders, mientras que la segunda es un MLP (Perceptron multicapa). Durante la fase preentrenamiento la primer sección es la que se utiliza, es decir, el modelo se trabaja como una lista de autoencoders utilizando el paradigma de capa egoísta por lo que el entrenamiento de cada autoencoder se realiza separadamente. En la segunda fase del entrenamiento se utiliza la MLP. Las dos estructuras (autoencoders y MLP) están conectadas porque los autoencoders y las capas sigmoides de la MLP comparten parámetros y además, los autoencoders tienen como entrada representaciones latentes de las capas intermedias de la MLP.

5.3. Algoritmos de autocodificación

5.3.1. Autoencoders ruidosos

Lo que se desea es implementar un autoencoder utilizando algún lenguaje de alto nivel con capacidad de manejo de clases y objetos, esto es porque se desea diseñar una clase que pueda ser usada después como elemento constructor de una arquitectura profunda. El primer paso es definir los atributos que tendrá la clase autoencoder que incluyen W, b y b', dado que se está utilizando una configuración de pesos atados. Una forma usual de inicializar los atributos de dicha clase sería:

$W = aleatorio(tamaño = [unidades\_visibles, unidades\_ocultas], mínimo, máximo)$

$bias = zeros(tamaño = [unidades\_visibles])$

$bias\_oculto = zeros(tamaño = [unidades\_ocultas])$

$W\_oculto = transpuesta(W)$

Aquí, el proceso de selección aleatoria de pesos se realiza sobre una distribución uniforme con valores mínimo y máximo calculados por:

$mínimo = -4 * sqrt\,(6/(unidades\_ocultas + unidades\_visibles))$

$máximo = +4 * sqrt\,(6/(unidades\_ocultas + unidades\_visibles))$

Después, se definen la manera de calcular la representación latente y la señal final reconstruida:

$valores\_ocultos = sigmoide\,(W * entrada + bias)$

$valores\_reconstruídos = sigmoide(W\_oculto * valores\_ocultos + bias\_oculto)$

Y usando estos valores, se puede calcular la función de costo y las actualizaciones de los parámetros correspondientes a un paso del proceso de aprendizaje por gradiente descendente estocástico.

$costo = suma(entrada * log(valores\_reconstruídos) + (1 - entrada) * log(1 - valores\_reconstruídos))$

$gradient\_x = calcula\_gradiente(costo, x)$

Así, se calcula la influencia que tiene el parámetro x en la función de costo de tal manera que sea posible estimar una expresión para su actualización, esto se hace para cada parámetro del modelo, en este caso los dos vectores de bias y las dos matrices de pesos. Se aprecia además, que la función de costo corresponde a la función de entropía cruzada.

$parametro\_x = parametro\_x - tasa\_aprendizaje * gradiente\_x$

De esta manera, se puede definir una función iterativa, tal que aplicada, actualice los parámetros de forma que el costo de reconstrucción sea minimizado.

Como se ha descrito, el método mas usado durante este proceso iterativo es el de gradiente descendente estocástico que requiere un movimiento a lo largo de la totalidad de los batches del conjunto de entrenamiento, esto es:

```
mientras (epoca < epoca_total) y (salir = falso):
    epoca = epoca + 1
    por cada minibatch en el conjunto de entrenamiento:
        perdida = f(parametros, x_batch)
        gradiente = ...calculagradiente
        parametros = parametros - tasa_aprendizaje * gradiente
        si perdida <= objetivo
            salir = verdadero
regresar parametros
```

Se mencionó anteriormente la necesidad de introducir algún proceso de corrupción para evitar que el modelo aprenda simplemente la función identidad lo cual le quita su capacidad de representación abstracta en la capa oculta.

Para convertir una clase de tipo autoencoder, como la que se ha descrito anteriormente, en una de tipo autoencoder ruidoso, lo único que se necesita es añadir una etapa de corrupción estocástica operando en la entrada del modelo. La entrada puede ser

corrompida de varias maneras, en el presente trabajo se utiliza el mecanismo de corrupción de enmascaramiento de la entrada haciendo que algunos elementos de ésta sean aleatoriamente cero.

$máscara = distribución\_binomial(tamaño = tamaño\_delaentrada, n = 1, p = 1 - nivel\_corrupcion)$

donde n es el valor de la máscara en caso de que un elemento en particular de la entrada se conserve y p la probabilidad de que un elemento de la entrada de convierta en cero, esta probabilidad esta parametrizada por la variable $nivel\_corrupcion$ que usualmente tiene valores entre 0.1 y 0.2. Así la entrada nueva se calcula por medio de un producto punto a punto de la entrada con la máscara.

$entrada\_corrupta = entrada. * máscara$

Esta entrada nueva es la que se utilizará para calcular las salidas tanto intermedia como final del modelo pero la entrada con la cual se calcula el error el reconstrucción es la la entrada original, esto se hace con el objetivo de construir un modelo que aprenda a predecir valores faltantes de la entrada bajo la distribución de probabilidad del conjunto de entrenamiento.

5.3.2. Autoencoders apilados (SDA)

Los autoencoders y las máquinas restringidas de Boltzmann constituyen las estructuras básicas de construcción de las llamadas arquitecturas profundas, como se ha mencionado, se utiliza una estrategia de entrenamiento de tipo capa egoísta durante el ajuste de los pesos sinápticos iniciales del modelo y además, el ajuste final se efectúa considerando el modelo como una MLP optimizada por medio de técnicas de gradiente descendente estocástico. Si se desea que el modelo tenga n capas, se deben construir n capas de tipo sigmoide (la MLP está formada por estas capas) y n capa de tipo autoencoder ruidoso (con las características de la clase construida en la Sección 5.3.1). Enlazamos dinámicamente las capas sigmoides tal que estas formen una MLP, y se crea cada autoencoder i tal que éste comparta la matriz de pesos y bias de su etapa codificadora con la correspondiente capa sigmoide i.

`mientras` ($i < número\_capas$)`:`

    `si` $i = 1$:

        $entrada\_capa = entrada$

    `caso contrario:`

        $entrada\_capa = capa\_sigmoide\_anterior.salida$

    $i = i + 1$

$capa\_sigmoide = constructor(entrada\_sigmoide = entrada\_capa)$

$capa\_autoencoder = constructor(entrada\_encoder = entrada\_capa$

$W = capa\_sigmoide.W$

$bias = capa\_sigmoide.bias$

$capa\_sigmoide\_anterior = capa\_sigmoide$

$i = i + 1$

Una vez realizadas tales asignaciones, se agrega una capa final al modelo que calcule la respuesta total de la red, la estructura de esta última capa depende fundamentalmente de la tarea que el modelo esté desempeñando, será una capa lineal en el caso de una tarea de identificación o una capa de regresión logística si el objetivo es una clasificación de patrones.

$capa\_final = constructor(entrada\_final =$

$capa\_sigmoide\_anterior.salida)$

Con las consideraciones anteriores, los parámetros que se asignan a la arquitectura profunda serán los pesos y biases de las capas sigmoides y de la capa final; los parámetros de los autoencoders no se consideran como parámetros del modelo porque están compartidos con las capas sigmoides y su actualización se realiza de manera conjunta.

Existen dos etapas durante el entrenamiento de un conjunto de autoencoders ruidosos apilados, la primera es un preentrenamiento de tipo capa egoísta y la segunda es un aprendizaje supervisado.

Para la etapa de pre-entrenamiento, se realizan iteraciones sobre todas las capas del modelo. Por cada capa se implementa una función que ejecuta un algoritmo de gradiente descendente que optimiza los pesos para reducir el error de reconstrucción de cada capa. Esta función es aplicada utilizando el conjunto de datos de entrenamiento durante un número fijo de épocas de entrenamiento.

`desde` $i = 1$ `hasta` $i = número\_de\_capas$`:`

`mientras` $(epoca < epocas\_preentrenamiento)$`:`

$epoca = epoca + 1$

`por cada` $minibatch$ `en el conjunto de entrenamiento:`

$preentrena\_capa[i](entrada = batch\_actual)$

`regresar` $parametros$

El ciclo de ajuste fino es el mismo que el que se utilizará para el entrenamiento de una MLP, la única consideración es que se debe elegir adecuadamente la función de costo que se desea minimizar de acuerdo a la tarea o problema que está desempeñando.

En el caso de utilizar una capa de regresión logística la función de costo consistirá en el cálculo de la log-verosimilitud negativa mientras que para fines de regresión, se utiliza la función clásica de error cuadrático medio.

$$E_c = \frac{1}{N} \sum \left(y_{ext,i} - y_i \right)^2$$

Para optimizar la velocidad de convergencia de ajuste fino se utilizan las condiciones de paro temprano (presentadas en la Sección 3.4) donde los parámetros actualizados por medio del gradiente descendente estocástico son los parámetros del modelo (capas sigmoides y capa de salida).

5.4. Aplicación: Clasificación de dígitos MNIST

La base de datos MNIST es una aglomeración ordenada de dígitos escritos a mano (ver Figura 5.4.1) y es comúnmente usada para el entrenamiento de diversos sistemas de procesamiento de imágenes. La base de datos también es utilizada durante procesos de entrenamiento y prueba en el campo del aprendizaje automático. Las imágenes en blanco y negro fueron tomadas reorganizando los conjuntos de datos de la la base de datos original NIST que carece de practicidad al presentar un conjunto de prueba con muestras poco representativas respecto al conjunto de entrenamiento.

La base de datos está constituida por 40,000 imagénes de entrenamiento, 10,000 imágenes para validación y 10,000 imágenes para prueba; cada una de las imágenes constituye una matriz de 28x28 píxeles con un valor en escala de grises normalizado entre 0 y 1. Existe una gran cantidad de artículos que explican intentos de alcanzar la tasa de error de predicción mas baja por lo que está base de datos se ha convertido en una herramienta estándar para la medición del desempeño de cualquier topología de clasificación de patrones .

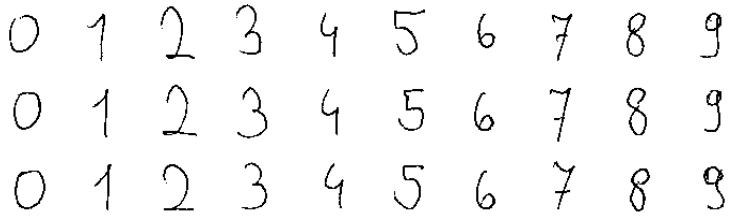


Figura 5.4.1: Dígitos escritos a mano alzada pertenecientes de la base de datos MNIST

A continuación se observa el desempeño de una topología profunda basada en auto-encoders comparándola con los resultados obtenidos por medio de arquitecturas poco profundas y sin preentrenamiento.

5.4.1. Regresión logística

La primer técnica de clasificación es ser probada y que es de importancia en el desarrollo de técnicas de aprendizaje profundo es la regresión logística, en el caso del problema de clasificación MNIST el modelo consta de 28x28 entradas (784) que proporcionan 10 salidas donde cada salida corresponde a la probabilidad de que la imagen de entrada corresponda a cierto dígito.

La tasa de aprendizaje supervisado fue de 0.1 con un número máximo de épocas de 1000. El algoritmo de entrenamiento fue aplicado utilizando condiciones de paro temprano para evitar el fenómeno de sobreajuste. Cada minibatch constó de 500 ejemplos y la condición de seguimiento de paro temprano fue una mejora significativa del 0.5 %.

Con estas restricciones se consigue que el algoritmo pare en la época 96 con un error de validación del 7.22 % y un error de prueba de 7.69 %. El tiempo de ejecución del algoritmo fue de 230 segundos.

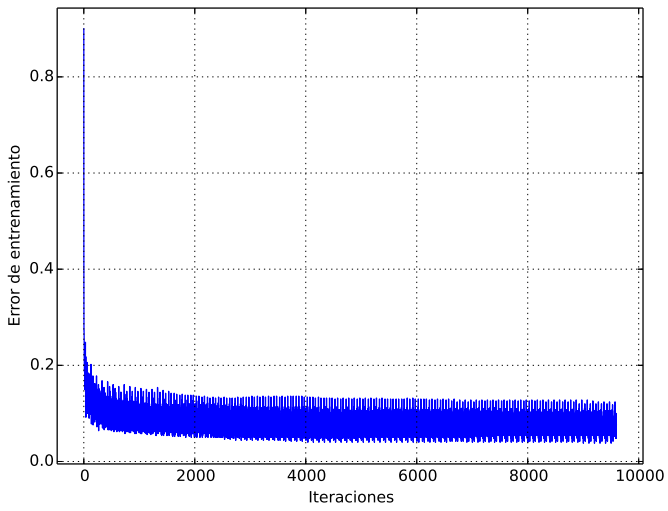


Figura 5.4.2: Error de aprendizaje durante el entrenamiento del modelo de regresión logística

La Figura 5.4.2 anterior muestra un comportamiento oscilatorio en el valor del error de entrenamiento, esto es debido a que cada muestra del error fue tomada una vez fueron actualizados los parámetros del modelo debido a la influencia de un minibatch en particular, y al ser la MNIST una base datos no uniformemente distribuida una actualización en particular puede representar un incremento en el error durante la siguiente iteración de entrenamiento, aunque como también se observa, la tendencia es la disminución del error.

Por otra parte, el error de validación y el error de prueba muestran una clara mejoría conforme el número de épocas aumenta.

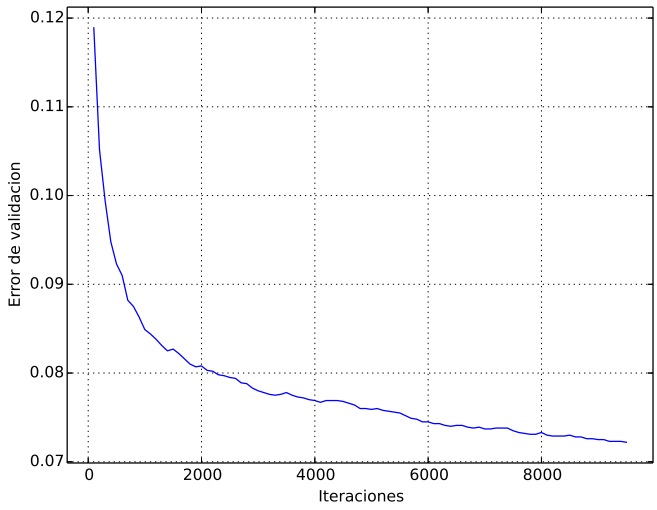


Figura 5.4.3: Evolución del error de validación respecto al número de iteraciones de entrenamiento

Como se ve en las Figuras 5.4.3 y 5.4.4, los errores de validación y prueba muestran comportamiento descendentes similares pero con una clara tendencia a que el error de validación siempre fue mejor que el error de prueba, esto es porque la MNIST tiene como conjunto de validación ejemplos pertenecientes al mismo conjunto muestral del que provienen los ejemplos de entrenamiento mientras que el conjunto de prueba fue tomado a partir de dígitos obtenidos de un sector poblacional distinto.

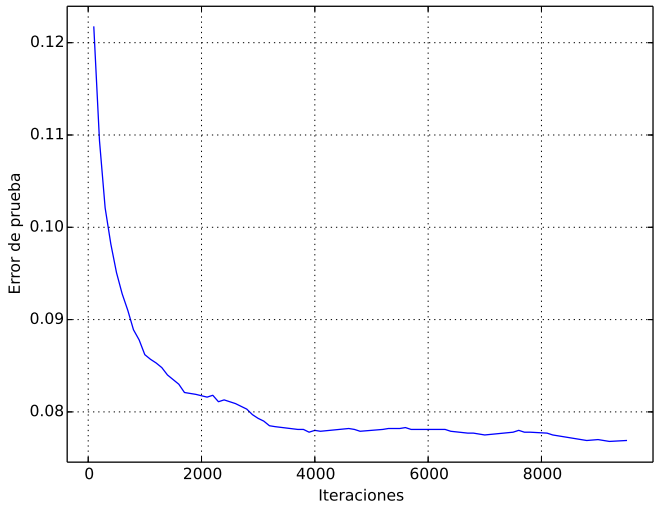


Figura 5.4.4: Evolución del error de prueba respecto al número de iteraciones de entrenamiento

5.4.2. Perceptrón multicapa

Un modelo perceptrón multicapa es un regresor logístico donde en lugar de alimentar la entrada a una capa de regresión logística usual, se ingresa una capa intermedia denominada capa oculta que tiene un función de activación no lineal (en este caso tangente hiperbólica). Se pueden utilizar tantas capas ocultas como se desee incrementando la profundidad de la arquitectura del modelo pero con el riesgo de sufrir el efecto de desvanecimiento de gradiente como se ha explicado anteriormente.

La topología utilizada para el problema de clasificación de la base de datos MNIST fue constituida por una sola capa oculta, 500 unidades ocultas por cada capa, tamaño de minibatch de 20 ejemplos y tasa de aprendizaje de 0.01.

Con esta topología, se obtuvo un error de validación de 1.79 % y un error de prueba de 1.92 %, el tiempo de ejecución del algoritmo fue de 183 minutos. La dinámica del error de entrenamiento (Figura 5.4.5) se sigue de manera similar al caso de regresión logística con un comportamiento oscilatorio y tendencia descendente debido a que no todos los minibatches constituyen ejemplos representativos de la distribución de probabilidad del conjunto de entrenamiento completo.

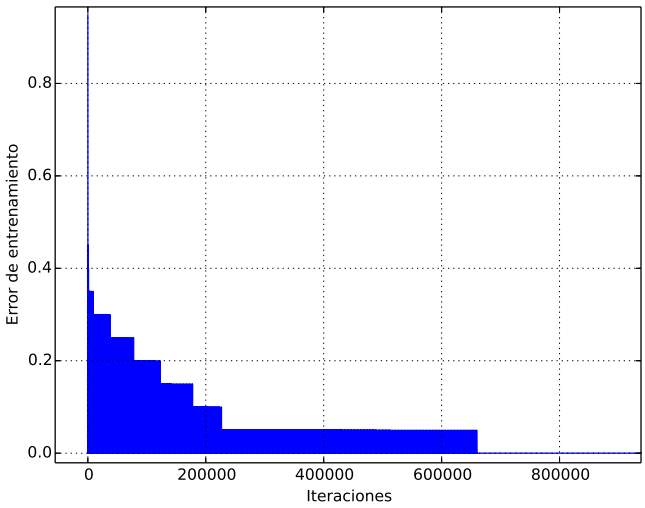
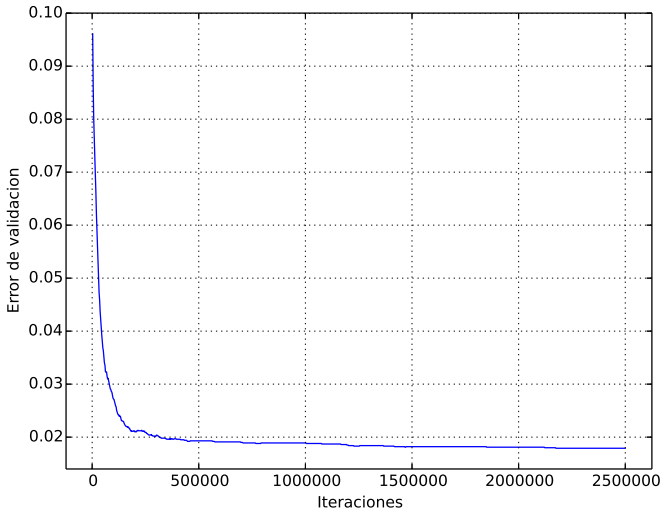
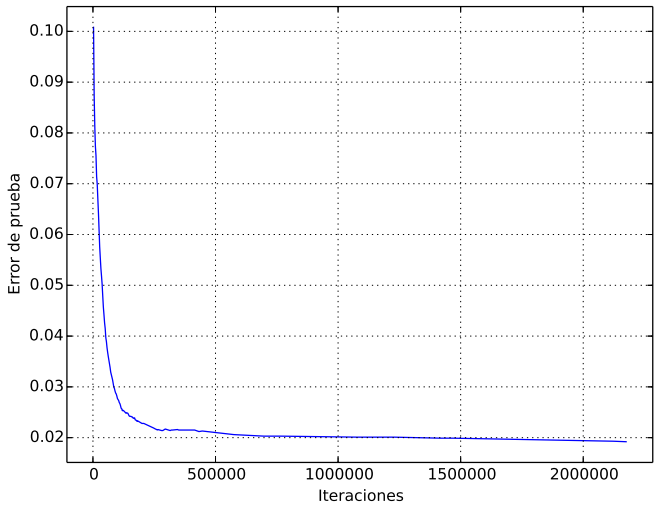


Figura 5.4.5: Evolución del error de entrenamiento durante las primeras 800000 iteraciones utilizando minibatches de 20 elementos

Los errores de validación y prueba (Figura 5.4.6) tienen una dinámica con tendencia descendente sin las oscilaciones presentadas por el error de entrenamiento, esto es porque el ambos conjuntos de ejemplos tienen representación sobre la totalidad de la distribución de entrenamiento a diferencia de los minibatches utilizados durante el entrenamiento. A si mismo, el error de prueba nuevamente es mayor que el error de validación debido a las similitudes que tiene cada conjunto con el conjunto de entrenamiento.



a)



b)

Figura 5.4.6: Evolución de los errores de validación y prueba del perceptrón respecto al número de iteraciones de entrenamiento

5.4.3. Autoencoders ruidosos apilados

Los autoencoders ruidosos constituyen bloques de construcción para una arquitectura profunda. Para probar el desempeño de un modelo profundo basado en autoencoders ruidosos apilados se consideró nuevamente el problema de clasificación MNIST considerando que cada autoencoder compartiría parámetros con una capa oculta de tipo sigmoide perteneciente a una Perceptrón profunda. La capa de salida estuvo determinada por un algoritmo de regresión logística usual, con 10 salidas de probabilidad, para cada uno de los dígitos a clasificar cuyo valor máximo de probabilidad constituye la

salida de predicción del modelo.

Se escogió para fines de prueba una arquitectura de 3 capas ocultas con 500 unidades cada una, 20 épocas de preentrenamiento para cada capa y 1000 épocas de ajuste fino supervisado con tasas de aprendizaje de 0.1 y 0.01 respectivamente.

Dado que se trabajan con autoencoders ruidosos se consideran los índices de corrupción para cada capa, en este problema se utilizaron índices 0.1, 0.2 y 0.3 respectivamente para cada una de las capas del modelo, se aprecia que se le otorga un mayor grado de corruptibilidad a la capa mas cercana a la salida, esto tiene como objetivo otorgarle una mayor capacidad de representación al modelo completo.

La función de costo en la etapa de preentrenamiento fue la entropía cruzada que tuvo que ser disminuida por medio de gradiente descendente por capa egoísta para cada una de las representaciones intermedias. La evolución del error de preentrenamiento para cada una de las capas de la red se observa en la Figura 5.4.7. El algoritmo de optimización de los autoencoders corrió por 160 minutos:

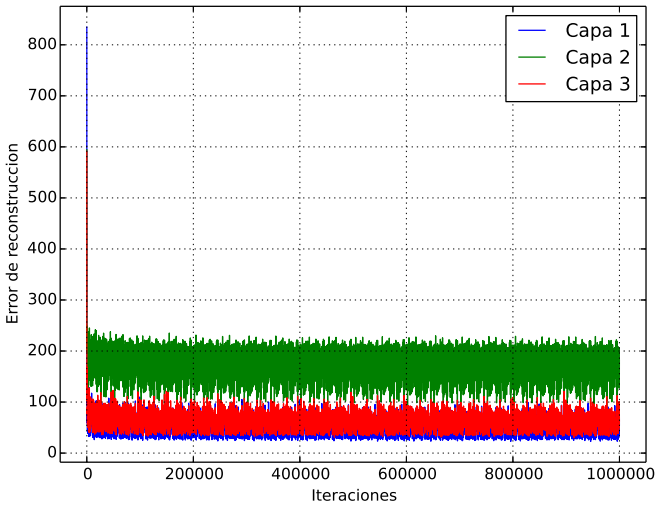


Figura 5.4.7: Evolución de la función de costo de preentrenamiento para cada una de las capas del SDA

El error medio de validación final fue de 1.22 % con un error de prueba total de 1.34 %. La dinámica del error de entrenamiento muestra un comportamiento oscilatorio similar al presentado por los modelos de regresión logística y Perceptrón multicapa pero teniendo solamente los valores discretos de 0 y 1 debido a que se utilizaron minibatches unitarios con lo que se eliminan las variables de representabilidad del conjunto de entrenamiento (ver Figura 5.4.8).

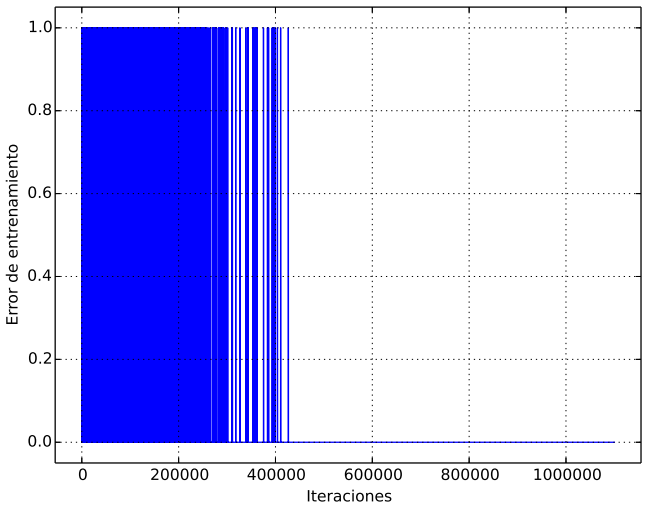
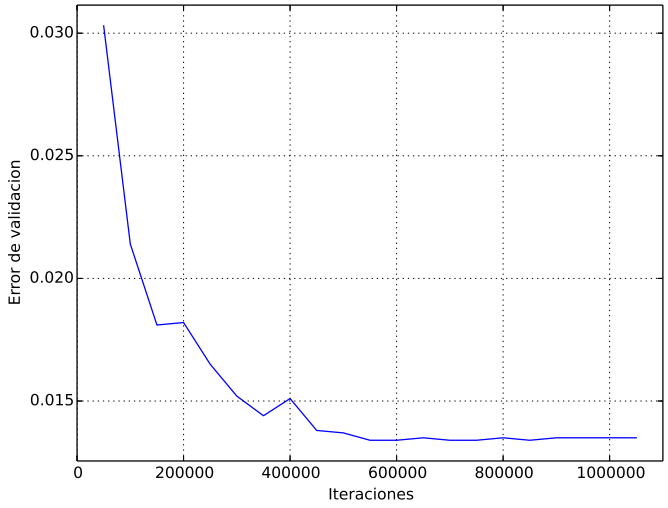
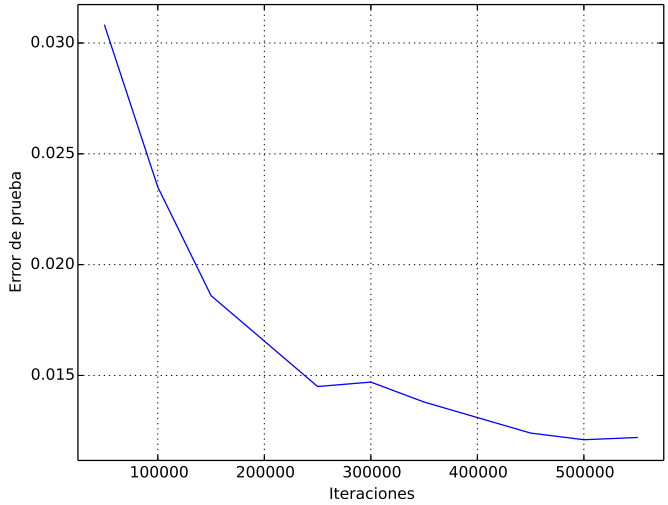


Figura 5.4.8: Evolución del error de entrenamiento durante las iteraciones utilizando minibatches unitarios

El error de validación y el error de prueba (Figura 5.4.9) nuevamente presentan comportamientos descendentes debido al algoritmo de entrenamiento utilizado, siendo nuevamente el valor de error de prueba ligeramente mayor que el error de validación para todas las épocas de entrenamiento. El programa de entrenamiento tomó 82 minutos en ejecutarse.



a)



b)

Figura 5.4.9: Evolución de los errores de validación y prueba de los autoencoders ruidosos respecto al número de iteraciones de entrenamiento

Capítulo 6

Aprendizaje restringido de Boltzmann en la identificación de sistemas no lineales

6.1. Máquinas Restringidas de Boltzmann

Los modelos basados en energía asocian una energía escalar con cada configuración del conjunto de variables de interés. El proceso de aprendizaje corresponde a modificar la forma de esta función de energía tal que ésta tenga propiedades deseadas, por ejemplo, en algunos casos es deseable tener configuraciones con baja energía. Los modelos basados en energía probabilísticos definen una distribución de probabilidad a través de la función de energía como:

$$p(x) = \frac{e^{-E(x)}}{Z}$$

El factor de normalización Z es llamado función de partición por su analogía con sistemas físicos.

$$Z = \sum_{x} e^{-E(x)}$$

Un modelo basado en energía puede aprender implementando un proceso de gradiente descendente estocástico sobre la log-verosimilitud empírica negativa de los datos de entrenamiento. Al igual que en el caso de la regresión logística se define primero la log-verosimilitud y después se define la función de costo como su versión negativa con la meta de minimizarla:

$$\begin{aligned}\mathcal{L}(\Theta, D) &= \frac{1}{N}\sum_{x^{(i)}\in D} \log p(x^{(i)}) \\ \ell(\Theta, D) &= -\mathcal{L}(\Theta, D)\end{aligned}$$

Para alcanzar la optimización del modelo se calcula el gradiente sobre los parámetros $-\frac{\partial \log p(x^{(i)})}{\partial \Theta}$, donde Θ son los parámetros del modelo.

6.1.1. Modelos basados en energía con unidades ocultas

En muchos casos de interés, es imposible la observación de los ejemplos de salida x en su totalidad, o también puede ser que se quiera introducir algunas variables no observadas para incrementar la capacidad expresiva del modelo que se está diseñando. Así que consideremos primero una parte observada (aún denotada por x) y una parte oculta h. Se puede escribir

$$P(x) = \sum_h P(x,h) = \sum_h \frac{e^{-E(x,h)}}{Z} \tag{6.1.1}$$

En tales casos, para llevar esta forma a la presentada para sistemas totalmente visibles, se introduce el concepto de energía libre, definida como:

$$F(x) = -\log \sum_h e^{-E(x,h)}$$

lo cual permite reformular

$$P(x) = \frac{e^{-F(x)}}{Z} \text{ con } Z = \sum_x e^{-F(x)}$$

Así, el gradiente de la log-verosimilitud negativa de los datos tiene una forma divisible en dos sumandos:

$$-\frac{\partial \log p(x)}{\partial \Theta} = \frac{\partial F(x)}{\partial \Theta} - \sum_{\tilde{x}} p\left(\tilde{x}\right) \frac{\partial F(\tilde{x})}{\partial \Theta} \tag{6.1.2}$$

Nótese que el gradiente contiene dos términos, los cuales son referidos como fases positiva y negativa. Los términos positivo y negativo no hacen referencia a los signos positivos y negativos de la ecuación, sino que reflejan el efecto que tienen en la densidad de probabilidad definida por el modelo. El primer término incrementa la probabilidad de los datos de entrenamiento (reduciendo la energía libre correspondiente), mientras que el segundo término decrementa la probabilidad de las muestras generadas por el modelo.

Usualmente es difícil determinar una expresión para este gradiente analíticamente , debido a que implica el cálculo de $E_P\left[\frac{\partial F(x)}{\partial \Theta}\right]$. Esta expresión representa la esperanza matemática sobre todas las posibles combinaciones de la entrada x (bajo la distribución P formada por el modelo)

El primer paso en la realización de este cálculo viable es estimar la esperanza utilizando un número fijo de muestras del modelo. Las muestras utilizadas para la estimación de la fase negativa del gradiente son conocidas como partículas negativas, las cuales se denotan por $\aleph$. El gradiente entonces se escribe como

$$-\frac{\partial \log p(x)}{\partial \Theta} \approx \frac{\partial F(x)}{\partial \Theta} - \frac{1}{|\aleph|}\sum_{\widetilde{x}\in\aleph}\frac{\partial F(\widetilde{x})}{\partial \Theta} \tag{6.1.3}$$

donde se desea que idealmente elementos $\widetilde{x}$ de $\aleph$ sean muestreados de acuerdo a P (es decir, un algoritmo MonteCarlo). Con la fórmula anterior, se tiene prácticamente un algoritmo completo para el entrenamiento de una EBM, el único elemento faltante es la elección de estas partículas negativas $\aleph$. La bibliografía [40] sugiere que métodos como las cadenas de Markov de tipo MonteCarlo funcionan especialmente bien para modelos tales como las máquinas restringidas de Boltzmann que constituyen la base de una red de creencia profunda.

Las máquinas de Boltzmann son una forma particular del llamado campo aleatorio de Markov en su versión logarítmica lineal, es decir, para el cual la función de energía es lineal en sus parámetros. Para hacer este modelo lo suficientemente poderoso para representar distribuciones complicadas (es decir, ir de una configuración paramétrica limitada a una no paramétrica), consideraremos que algunas de las variables que describen a la representación nunca son observadas (llamadas ocultas). Teniendo mas variables ocultas (o unidades ocultas), se incrementa la capacidad de modelado de la máquina de Boltzmann (aunque también se incrementa su costo computacional). Las máquinas restringidas de Boltzmann (RBM) solamente toman en cuenta aquellos modelos en los que no existen conexiones del tipo visible-visible y oculta-oculta (ver Figura 6.1.1).

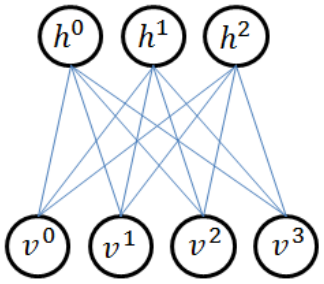


Figura 6.1.1: Máquina restringida de Boltzmann con dependencias v-o

La función de energía $E(v,h)$ de una RBM es definida como:

$$E\left(v,h\right) = -b^T v - c^T h - h^T W v \tag{6.1.4}$$

donde W representa los pesos que conectan la capa oculta con la capa visible y b, c son los offsets de las unidades visibles y ocultas respectivamente.

Esto se traduce directamente en la fórmula para la energía libre:

$$\digamma\left(v\right) = -b^T v - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i v)} \tag{6.1.5}$$

Debido a la estructura específica de las RBMs, las unidades ocultas y visibles son condicionalmente independientes dada una u otra. Usando esta propiedad se puede escribir:

$$\begin{aligned} p(h|v) &= \prod_i p(h_i|v) \\ p(v|h) &= \prod_j p(v_j|h) \end{aligned}$$

6.1.2. RBMs con entradas binarias

En el caso simplificado de usar unidades binarias (donde v_j y $h_i \in \{0,1\}$), obtenemos versiones probabilísticas de la función de activación neuronal usual de tipo sigmoide:

$$\begin{aligned} P\left(h_i = 1|v\right) &= \text{sigm}\left(c_i + W_i v\right) \\ P\left(v_j = 1|h\right) &= \text{sigm}\left(b_j + W_j^T h\right) \end{aligned}$$

Así, la energía libre de una RBM se simplifica a:

$$\digamma\left(v\right) = -b^T v - \sum_i \log\left(1 + e^{(c_i + W_i v)}\right) \tag{6.1.6}$$

6.1.3. Ecuaciones de aprendizaje y actualización con unidades binarias

Si se aplica un método de aprendizaje basado en el gradiente descendente de por ejemplo la verosimilitud entre los patrones y los de salida, se obtienen las ecuaciones de actualización [34] para la matriz de pesos sinápticos W y los bias b, c.

$$
\begin{aligned}
-\frac{\partial \log p(v)}{\partial W_{ij}} &= E_v\left[p\left(h_i|v\right)\cdot v_j\right] - v_j^{(i)}\cdot \operatorname{sigm}\left(W_i\cdot v^{(i)} + c_i\right) \\
-\frac{\partial \log p(v)}{\partial c_i} &= E_v\left[p\left(h_i|v\right)\right] - \operatorname{sigm}\left(W_i\cdot v^{(i)}\right) \\
-\frac{\partial \log p(v)}{\partial b_j} &= E_v\left[p\left(v_j|h\right)\cdot v_j\right] - v_j^{(i)}
\end{aligned}
$$

Sin embargo, estas fórmulas no fueron utilizadas durante la implementación del algoritmo de aprendizaje en el presente trabajo debido a que se utilizaron herramientas de manejo de álgebra simbólica para la obtención del valor del gradiente a los largo del entrenamiento.

6.1.4. Muestreo en una RBM

Las muestras de $p(x)$ pueden ser obtenidas implementando una cadena de Markov para convergencia, usando muestreo de Gibbs como el operador de transición.

El muestreo de Gibbs de la unión de N variables aleatorias $S = (S_1, ..., S_N)$ se realiza a través de una secuencia de N sub-pasos de muestreo de la forma $S_i \sim p(S_i|S_{i-1})$ donde S_{-1} contiene las otras $N-1$ variables aleatorias en S excluyendo S_i.

Para RBMs , S consiste de un conjunto de unidades visibles y ocultas. Sin embargo, debido a que estas unidades son condicionalmente independientes, podemos simplificar el proceso de muestreo de Gibbs a uno llevado en bloques. Con estas características, las unidades visibles son muestreadas simultáneamente dados valores fijos de las unidades ocultas. Similarmente, las unidades ocultas son muestreadas simultáneamente dados valores para las unidades visibles. Un paso en la cadena de Markov se realiza entonces como:

$$
\begin{aligned}
h^{(n+1)} &\sim \operatorname{sigm}\left(W^T v^{(n)} + c\right) \\
v^{(n+1)} &\sim \operatorname{sigm}\left(W h^{(n+1)} + b\right)
\end{aligned}
$$

donde $h^{(n)}$ hace referencia al conjunto de todas las unidades ocultas en el paso n de la cadena de Markov. Lo que esto significa es que, por ejemplo, $h_i^{(n+1)}$ es elegido aleatoriamente como 1 (ó 0) con probabilidad $\operatorname{sigm}\left(W_i^T v^{(n)} + c_i\right)$, y similarmente, $v_j^{(n+1)}$ es escogido aleatoriamente como 1 (ó 0) con probabilidad $\operatorname{sigm}\left(W_j h^{(n+1)} + b_j\right)$.

Esto se ilustra en la Figura 6.1.2:

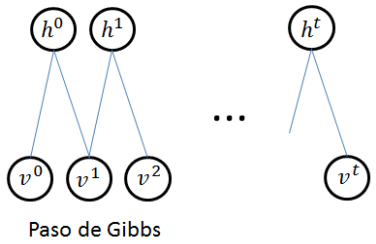


Figura 6.1.2: Muestreo en una máquina restringida de Boltzmann

Conforme $t \to \infty$, las muestras $\left(v^{(t)}, h^{(t)}\right)$ representan adecuadamente ejemplos de $p(v, h)$

Con estas consideraciones, cada parámetro actualizado durante el proceso de aprendizaje requeriría correr una cadena en particular para converger. Esto es, sin embargo, inmensamente costoso desde el punto de vista computacional y por ello, se han desarrollado diversos algoritmos para el entrenamiento de RBMs con el objetivo de realizar un proceso de muestreo eficiente y menos costoso de la probabilidad $p(v, h)$ durante el proceso de entrenamiento.

Divergencia contrastiva (CD-k)

La divergencia contrastiva usa 2 eventos para acelerar el proceso de muestreo:

- como lo que queremos es que eventualmente $p(v) \approx p_{train}(v)$ (la cual es la verdadera forma de distribución de los datos), inicializamos la cadena de Markov con un ejemplo de entrenamiento (es decir, de una distribución que se espera parecida a p, tal que la cadena esté desde un inicio cerca de converger a su distribución final p)
- la CD no espera hasta que la cadena converja. Las muestras son obtenidas después de solo k pasos del proceso de Gibbs. En práctica, $k = 1$ es una elección bastante común.

CD Persistente

La divergencia contrastiva persistente usa otra aproximación para realizar las muestras de $p(v, h)$. Esto yace sobre una cadena de Markov, la cual tiene un estado persistente (es decir, no reiniciando una cadena por cada ejemplo observado). Por cada parámetro actualizado, extraemos nuevos ejemplos simplemente corriendo la cadena durante k pasos. El estado de la cadena es entonces preservado para actualizaciones subsecuentes.

La heurística detrás de esto es que si las actualizaciones de los parámetros son suficientemente pequeñas comparadas con la tasa de mezclado de la cadena de Markov, entonces la cadena debería ser capaz de "atrapar" estos pequeños cambios en el modelo.

6.1.5. Seguimiento y evaluación del progreso de entrenamiento

Las RBMs requieren procesos estocásticos durante su entrenamiento. Esto se debe a que debido a la función de partición Z, no es posible estimar la log-verosimilitud $\log(P(x))$ durante el entrenamiento. Así no se cuenta con una forma directa de obtener una métrica adecuada para la elección de los hiperparámetros de la red.

Algunas soluciones son las siguientes:

- Inspección de muestras negativas.
- Muestras negativas obtenidas durante el entrenamiento pueden ser visualizadas. Conforme el entrenamiento avanza, hemos definido que el modelo definido por la RBM se acerca progresivamente a la verdadera distribución de probabilidad subyacente, $p_{train}(x)$. Así, las muestras negativas deberían parecerse a los ejemplos tomados del conjunto de entrenamiento, si esto no se cumple se debería escoger un nuevo conjunto de hiperparámetros.
- Inspección visual de los filtros. Los filtros aprendidos por el modelo pueden ser desplegados. Eso implica graficar los pesos de cada una unidad como una imagen en escala de grises (una vez cambiada su forma a una matriz cuadrada). Los filtros deben atrapar características importantes de los datos, esto es problemático porque para un conjunto arbitrario de datos de entrenamiento no se tiene conocimiento de como deberían interpretarse estas características; sin embargo, en un problema de clasificación de patrones en el cual se utilicen datos provenientes de imágenes, los filtros se comportan como detectores de cambios máximos en gradiente.

6.1.6. Aproximaciones de la verosimilitud

Estos métodos de elección de los mejores hiperparámetros dependen en gran medida de la capacidad analítica de la persona que está implementando el algoritmo por lo que no son recomendables en todos los casos. Durante el entrenamiento de una RBM con PCD, se puede utilizar la pseudo-verosimilitud como aproximador. La pseudo-verosimilitud (PL) es mucho menos costosa de calcular, debido a que supone que todos los bits de información son independientes. Así

$$\begin{aligned} PL(x) &= \prod_i P\left(x_i | x_{-i}\right) \\ \log PL(x) &= \sum_i \log P(x_i | x_{-i}) \end{aligned}$$

Aquí x_{-i} denota el conjunto de todos los bits de x con excepción del bit i. La log-PL es entonces la suma de las log-*probabilidades* de cada bit x_i, condicionado al estado del resto de los bits. Para el caso del problema clásico de clasificación de dígitos MNIST esto implicaría sumar sobre 784 entradas, lo cual a pesar de las consideraciones anteriores es aún costoso. Para ello, se introduce una aproximación estocástica de la log-PL :

$$\begin{aligned} g &= N \cdot \log P\left(x_i | x_{-i}\right), \text{ donde } i \sim U(0, N), \\ E[g] &= \log PL(x) \end{aligned}$$

donde la esperanza matemática es tomada sobre la elección uniformemente aleatoria del índice i, y N es el número de unidades visibles. Para el trabajo con unidades binarias, se introduce la notación $\widetilde{x_i}$ para hacer referencia a x con el bit i cambiado $(1 \to 0,\ 0 \to 1)$. Entonces la log-PL para una RBM con unidades binarias se escribe como:

$$\log PL(x) \approx N \log \frac{e^{-FE(x)}}{e^{-FE(x)} + e^{-FE(\widetilde{x_i})}} \tag{6.1.7}$$

$$\approx N \cdot \log \left[\text{sigm} \left(FE(\widetilde{x_i}) - FE(x) \right) \right] \tag{6.1.8}$$

Durante la implementación del calculo de este costo, se debe regresar el mismo así las actualizaciones paramétricas correspondientes. Se debe hacer hincapié en que la modificación del diccionario de actualizaciones para todos los parámetros debe modificarse para incrementar el índice de bit i. Esto se traduce en el bit i moviéndose sobre todos los posibles valores $\{0, 1, ..., N\}$ desde una actualización a otra.

Nótese que para entrenamiento con divergencia contrastiva la función de costo de entropía cruzada entre la entrada y la reconstrucción (la misma función utilizada en un autoencoder ruidoso) es más eficiente que la pseudo-log-verosimilitud[3].

6.2. Máquinas restringidas de Boltzmann con entradas continuas

La literatura [6, 7, 9] describe adecuadamente el comportamiento de las máquinas restringidas de Boltzmann cuando son expuestas a entradas binarias pero carecen de la extrapolabilidad al caso continuo que es necesario para la implementación de técnicas de aprendizaje profundo para la resolución del problema de identificación de sistemas, por ello en la siguiente Sección se describen algunas posibilidades para solventar el problema de considerar una RBM con entradas continuas.

6.2.1. Unidades ocultas y visibles binarias

En el caso de las RBMs expuestas en la Sección 6.1 se realizó la simplificación de considerar la entrada al modelo como binaria, es decir, donde los valores entrantes a las unidades visibles pueden tomar solamente los valores de 0 y 1, tomando esto en consideración es como fueron calculadas las fórmulas de probabilidad condicional utilizadas durante el proceso de muestreo de Gibbs obtenido para fines de clasificación. La obtención de las fórmulas de probabilidad se expone a continuación:

De la ecuación 6.1.1 se sabe que la probabilidad condicional está dada por:

$$P(v) = \sum_h P(v,h) = \sum_h \frac{e^{-E(v,h)}}{Z}$$

la probabilidad de que un evento h suceda queda representada por una expresión similar:

$$P(h) = \sum_v \frac{e^{-E(v,h)}}{Z}$$

mientras que la probabilidad conjunta $P(v,h)$ se determina por:

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}$$

donde v representa a los valores tomados por las unidades visibles y h a los valores tomados por las unidades ocultas. De la teoría de probabilidad se sabe que la probabilidad condicional puede obtenerse por el cociente de la probabilidad conjunta de eventos y el valor de la probabilidad del suceso condicionante, así la probabilidad condicional $P(v|h)$ se calcula como:

$$P(v|h) = \frac{P(v,h)}{P(h)} = \frac{\frac{e^{-E(v,h)}}{Z}}{\sum_v \frac{e^{-E(v,h)}}{Z}} = \frac{e^{-E(v,h)}}{\sum_v e^{-E(v,h)}}$$

Además se sabe por la ecuación 6.1.4 que la energía para una configuración específica se específica por:

$$E\left(v,h\right) = -b^T v - c^T h - h^T W v$$

Sustituyendo en la expresión para la probabilidad condicional se obtiene:

$$P(v|h) = \frac{e^{b^T v + c^T h + h^T W v}}{\sum_v e^{b^T v + c^T h + hWv}} \tag{6.2.1}$$

Al no variar el valor de h a lo largo de la sumatoria podemos reducir está expresión a:

$$\frac{e^{b^T v+c^T h+h^T W v}}{\sum_v e^{b^T v+c^T h+h^T W v}} = \frac{e^{c^T h} e^{b^T v+h^T W v}}{e^{c^T h}\sum_v e^{b^T v+h^T W v}} = \frac{e^{b^T v+h^T W v}}{\sum_v e^{b^T v+h^T W v}}$$

En el caso del caso estudiado que corresponde a una máquina restringida de Boltzmann no existe dependencia directa entre los valores tomados por las unidades visibles respecto a las demás unidades visibles por lo que la dependencia es directa con las unidades ocultas (lo contrario también ocurre). Así podemos escribir:

$$\begin{aligned} p(h|v) &= \prod_i p(h_i|v) \\ p(v|h) &= \prod_j p(v_j|h) \end{aligned}$$

desarrollando la expresión para $P(v|h)$ considerando como valores independientes cada uno de los valores del vector de unidades visibles obtenemos:

$$\frac{e^{b^T v+h^T W v}}{\sum_v e^{b^T v+h^T W v}} = \frac{e^{\sum\left(b_j v_j+h^T W_j v_j\right)}}{\sum_v e^{\sum(b_j v_j+h^T W_j v_j)}}$$

$$\frac{e^{\sum\left(b_j v_j+h^T W_j v_j\right)}}{\sum_v e^{\sum(b_j v_j+h^T W_j v_j)}} = \frac{\prod e^{b_j v_j+h^T W_j v_j}}{\sum_v \prod e^{b_j v_j+h^T W_j v_j}} = \prod_j \frac{e^{b_j v_j+h^T W_j v_j}}{\sum_v e^{b_j v_j+h^T W_j v_j}}$$

De acuerdo a esta última expresión y asociándola con el hecho de que la probabilidad condicional del vector completo de unidades visibles es el producto de las probabilidades condicionales de cada uno de los elementos se puede realizar la siguiente igualdad:

$$p(v_j|h) = \frac{e^{b_j v_j+h^T W_j v_j}}{\sum_v e^{b_j v_j+h^T W_j v_j}} \tag{6.2.2}$$

En el denominador se observa que debemos calcular el valor de la sumatoria recorriendo la totalidad de posibles valores de v, en el caso de unidades binarias estos valores son solamente 0 y 1, con lo que podemos calcular la probabilidad condicional $p(v_j = 1|h)$ como:

$$p(v_j = 1|h) = \frac{e^{b_j v_j+h^T W_j v_j}}{\sum_v e^{b_j v_j+h^T W_j v_j}}|_{v_j=1} = \frac{e^{b_j+h^T W_j}}{1+e^{b_j+h^T W_j}}$$

$$p(v_j = 1|h) = \text{sigm}(b_j + h^T W_j) \tag{6.2.3}$$

Un análisis similar puede hacerse para obtener la probabilidad condicional de los valores de las unidades ocultas respecto a valores dados de las unidades visibles con lo que obtenemos finalmente:

$$P\left(h_i = 1|v\right) = \text{sigm}\left(c_i + W_i v\right) \tag{6.2.4}$$
$$P\left(v_j = 1|h\right) = \text{sigm}\left(b_j + h^T W_j\right) \tag{6.2.5}$$

A pesar de las limitaciones que se tiene considerando a las unidades visibles binarias, está configuración ha funcionado adecuadamente inclusive con sistemas con entradas continuas normalizadas unitarias (como es el caso de imágenes en escala de grises) [40], esto se debe a que la entrada se asimila como una distribución de probabilidad que ha afectado a la entrada original.

6.2.2. Unidades visibles continuas en el intervalo $[0, \infty]$

De la ecuación 6.2.2 podemos generalizar la idea de probabilidad condicional para unidades visibles de tipo continuo transformando la sumatoria en una integral a lo largo del dominio visible.

$$p(v_j|h) = \frac{e^{(b_j + h^T W_j)v_j}}{\int_v e^{(b_j + h^T W_j)v_j} dv_j} \tag{6.2.6}$$

Puede demostrarse que en general el caso en el que $v \in [-\infty, \infty]$ carece de convergencia en el término integral de la ecuación con lo que se analiza solamente el caso semidefinido positivo $[0, \infty]$.

$$p(v_j|h) = \frac{e^{(b_j + h^T W_j)v_j}}{\int_0^\infty e^{(b_j + h^T W_j)v_j} dv_j}$$

Denotemos entonces como $a_j = a_j(h) = b_j + h^T W_j$ al termino que multiplica al valor que toma la unidad visible, entonces podemos desarrollar la expresión como:

$$p(v_j|h) = \frac{e^{a_j v_j}}{\int_0^\infty e^{a_j v_j} dv_j} = \frac{e^{a_j v_j}}{\frac{1}{a_j} e^{a_j v_j}|_0^\infty}$$

Esta última expresión solamente tiene valores finitos si la estructura del sistema garantiza que $a_j(h) < 0 \forall h$, en el caso de que la condición se cumpla se obtiene la distribución de probabilidad condicional:

$$p(v_j|h) = -a_j e^{a_j v_j} \tag{6.2.7}$$

la cual es positiva de acuerdo a la condición de existencia de la integral. Con esta fórmula para la distribución de probabilidad no podemos realizar directamente el proceso de muestreo ya que en general las herramientas computacionales no permiten muestrear desde una distribución de probabilidad particular, para resolver este problema se procede a calcular la distribución de probabilidad acumulada $p_a(v_j|h)$ de la variable la cual siempre tendrá valores crecientes en el intervalo $[0, 1]$. Utilizando la función inversa de probabilidad acumulada p_a^{-1} se podrá entonces muestrear desde una distribución uniforme. El calculo de p_a se realiza de manera usual:

$$p_a(v_j|h) = \int_0^{v_j} p(v_j|h)dv_j = \int_0^{v_j} -a_j e^{a_j v_j} dv_j$$

$$p_a(v_j|h) = 1 - e^{a_j v_j} \tag{6.2.8}$$

y calculando p_a^{-1} obtenemos:

$$v_j = \frac{\ln(1 - p_a)}{a_j}$$

Si denotamos por U el resultado de un proceso de muestreo sobre una distribución uniforme en el intervalo entonces podemos relacionar a este valor U con el valor de la densidad acumulada p_a con lo que restaría encontrar el valor de la unidad visible correspondiente.

$$v_j = \frac{\ln(1 - U)}{a_j} \tag{6.2.9}$$

El valor esperado o esperanza matemática de acuerdo a la distribución $p(v_j|h)$ es:

$$E[v_j] = \int_0^{\infty} p(v_j|h)v_j dv_j = -a_j \int_0^{\infty} e^{a_j v_j} v_j dv_j$$

Integrando por partes se encuentra:

$$E[v_j] = -a_j \left[\frac{1}{a_j} v_j e^{a_j v_j} - \frac{1}{a_j^2} e^{a_j v_j} \right] |_0^{\infty}$$

$$E[v_j] = -\frac{1}{a_j(h)} \tag{6.2.10}$$

A las unidades visibles que tienen como distribución de probabilidad lo presentado anteriormente se les conoce como unidades exponenciales.

6.2.3. Unidades continuas visibles en el intervalo $[0, 1]$

En muchas de las aplicaciones el intervalo I de posibles valores para las unidades visibles queda restringido (depende sobretodo de la dinámica y de la fuente de la entrada al modelo), el caso mas simple es donde se ha normalizado la entrada (ya sea por condiciones naturales del sistema o por algún proceso de normalización) siendo $I = [0, 1]$

Cuando el calculo de la distribución de probabilidad se ve acotado como en este caso se habla de unidades visibles exponenciales truncadas, de la ecuación 6.2.6 se obtiene:

$$p(v_j|h) = \frac{e^{a_j v_j}}{\int_0^1 e^{a_j v_j} dv_j} = \frac{e^{a_j v_j}}{\frac{1}{a_j} e^{a_j v_j}|_0^1}$$

$$p(v_j|h) = \frac{a_j e^{a_j v_j}}{e^{a_j} - 1} \tag{6.2.11}$$

Para efectos de implementar un proceso de muestreo de Gibbs es necesario el calculo de la probabilidad condicional $p_a(v_j|h)$ como sigue:

$$p_a(v_j|h) = \int_0^{v_j} p(v_j|h) dv_j = \frac{a_j}{e^{a_j} - 1} \int_0^{v_j} e^{a_j v_j} dv_j$$

$$p_a(v_j|h) = \frac{e^{a_j v_j} - 1}{e^{a_j} - 1} \tag{6.2.12}$$

Despejando el valor de v_j se determina:

$$v_j = \frac{\log\left[1 + p_a(e^{a_j} - 1)\right]}{a_j}$$

Expresión que puede ser utilizada para calcular el nuevo valor de v_j durante un proceso de muestreo al sustituir el valor de la variable p_a por alguna muestra de distribución uniforme unitaria U, sustituyendo:

$$v_j = \frac{\log\left[1 + U(e^{a_j} - 1)\right]}{a_j} \tag{6.2.13}$$

Finalmente calculamos el valor esperado de la distribución:

$$E[v_j] = \int_0^{\infty} p(v_j|h) v_j dv_j = \frac{a_j}{e^{a_j} - 1} \int_0^1 e^{a_j v_j} v_j dv_j$$

Integrando por partes encontramos:

$$E[v_j] = \frac{a_j}{e^{a_j} - 1}\left[\frac{1}{a_j} v_j e^{a_j v_j} - \frac{1}{a_j^2} e^{a_j v_j}\right] |_0^1$$

$$E[v_j] = \frac{1}{1 - e^{-a_j}} - \frac{1}{a_j} \tag{6.2.14}$$

6.2.4. Unidades continuas visibles en el intervalo $[-\delta, \delta]$

Hasta ahora se han considerado solamente los casos en los que el valor de la entrada al modelo es positivo o puede ser normalizado para serlo; sin embargo, esto le quita naturalidad al proceso de identificación por lo que es necesario considerar el caso que ocurre cuando no se modifica el valor original de la entrada; para ello considérese un número δ tal que $\delta \in \mathbb{R}$ y $\delta > 0$, así es posible definir un intervalo de valores $I = [-\delta, \delta]$ posibles para cada una de las unidades visibles.

A la probabilidad condicional obtenida a partir de I también se le considera exponencial truncada, a partir de la ecuación 6.2.6 se puede calcular dicha probabilidad:

$$p(v_j|h) = \frac{e^{a_j v_j}}{\int_{-\delta}^{\delta} e^{a_j v_j} dv_j} = \frac{e^{a_j v_j}}{\frac{1}{a_j} e^{a_j v_j}|_{-\delta}^{\delta}}$$

$$p(v_j|h) = \frac{a_j e^{a_j v_j}}{e^{a_j \delta} - e^{-a_j \delta}} \tag{6.2.15}$$

La probabilidad acumulada para esta distribución condicional es:

$$p_a(v_j|h) = \int_{-\delta}^{v_j} p(v_j|h) dv_j = \frac{a_j}{e^{a_j \delta} - e^{-a_j \delta}} \int_{-\delta}^{v_j} e^{a_j v_j} dv_j$$

$$p_a(v_j|h) = \frac{e^{a_j v_j} - e^{-a_j \delta}}{e^{a_j \delta} - e^{-a_j \delta}} \tag{6.2.16}$$

Por lo que podemos realizar un proceso de muestreo utilizando la función inversa de p_a tomando un valor de p_a denotado por U sobre una distribución uniforme en el intervalo $[0, 1]$, así el valor de la unidad visible es:

$$v_j = \frac{\log\left[e^{-a_j \delta} + U\left(e^{a_j \delta} - e^{-a_j \delta}\right)\right]}{a_j} \tag{6.2.17}$$

Por último se obtiene una expresión para el valor esperado de esta distribución:

$$E[v_j] = \int_{-\infty}^{\infty} p(v_j|h) v_j dv_j = \frac{a_j}{e^{a_j\delta} - e^{-a_j\delta}} \int_{-\delta}^{\delta} e^{a_j v_j} v_j dv_j$$

Resolviendo se obtiene:

$$E[v_j] = \frac{a_j}{e^{a_j\delta} - e^{-a_j\delta}} \left[\frac{1}{a_j} v_j e^{a_j v_j} - \frac{1}{a_j^2} e^{a_j v_j} \right] |_{-\delta}^{\delta}$$

$$= \frac{a_j}{e^{a_j\delta} - e^{-a_j\delta}} \left[\frac{1}{a_j} \delta (e^{a_j\delta} + e^{-a_j\delta}) - \frac{1}{a_j^2} (e^{a_j\delta} - e^{-a_j\delta}) \right]$$

$$E[v_j] = \delta \frac{e^{a_j\delta} + e^{-a_j\delta}}{e^{a_j\delta} - e^{-a_j\delta}} - \frac{1}{a_j} \tag{6.2.18}$$

6.3. Redes de creencia profunda

Geoffrey Hinton demostró que las RBMs pueden ser apiladas y entrenadas de una manera egoísta para formar lo que el denominó redes de creencia profunda (DBN) [29]. Las redes de creencia profundas son modelos gráficos que aprenden a extraer una representación jerárquica profunda de los datos de entrenamiento. Estos modelos caracterizan la distribución conjunta h^k entre el vector observado x y las l capas ocultas como:

$$P(x, h^1, ..., h^l) = \left(\prod_{k=0}^{l-2} P\left(h^k | h^{k+1}\right) \right) P(h^{l-1}, h^l)$$

donde $x = h^0$, $P\left(h^k|h^{k+1}\right)$ es una distribución condicional para las unidades visibles limitadas sobre las unidades ocultas pertenecientes a la RBM en el nivel k, y $P(h^{l-1}, h^l)$ es la distribución conjunta oculta-visible en la RBM del nivel superior o de salida. Ésto es ilustrado por la Figura 6.3.1.

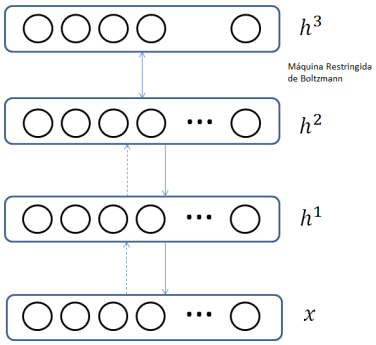


Figura 6.3.1: Red de creencia profunda formada por RBMs

El principio de entrenamiento no supervisado por capa egoísta puede ser aplicado a las DBNs con RBMs siguiendo el concepto de bloque constructor presentado por Hinton y Bengio [9, 29]. El proceso se realiza como se presenta a continuación:

1. Se entrena la primer capa como una RBM que modele la entrada original $x = h^{(0)}$ como su capa visible.

2. Usar esta primer capa para obtener una representación de la entrada que será usada como datos de ingreso para la segunda capa, dado que una RBM no contiene en su estructura una salida inherente, tenemos dos opciones para la asignación de esta representación. El conjunto de datos elegido es generalmente escogido como las activaciones promedio $p(h^{(1)} = 1|h^{(0)})$ o las muestras de $p(h^{(1)}|h^{(0)})$.

3. Entrenar la segunda capa como una RBM independiente, tomando los datos transformados (muestras o activaciones) como ejemplos de entrenamiento (para la capa visible de esa RBM).

4. Repetir pasos 2 y 3 de acuerdo al número deseado de capas, propagando hacía arriba ya sea las muestras o las activaciones.

5. Realizar un procedimiento de ajuste fino a todos los parámetros de esta arquitectura profunda respecto a una función de aproximación (proxy) de la log-verosimilitud de la DBN, o con respecto a un criterio de aprendizaje supervisado, para la realización de esto último es necesario añadir maquinaria extra en la arquitectura de la red que convierta la representación aprendida en predicciones supervisadas, por ejemplo un clasificador o función lineal.

El presente trabajo se enfoca en la implementación de una etapa de ajuste fino vía gradiente descendente supervisado. Específicamente, se hace uso del algoritmo de regresión logística para clasificar la entrada x basado en la salida de la última capa oculta $h^{(l)}$ de la DBN. El ajuste fino se implementa minimizando la función de costo de log-verosimilitud negativa con fines de clasificación (en el problema MNIST por ejemplo) o

el error cuadrático medio para identificación de funciones. Dado que el gradiente utilizado en la regla de actualización de parámetros es solamente distinto de cero para los pesos y biases de la subcapa oculta de cada capa de la DBN (es decir, es nulo para los biases visibles de cada RBM), este procedimiento es equivalente a la inicialización de parámetros de un MLP de arquitectura profunda con los valores de dichos parámetros obtenidos mediante una estrategia de entrenamiento no supervisado.

Para explicar cual es el efecto que tiene el preentrenamiento de tipo capa egoísta se toma como ejemplo una DBN poco profunda de 2 capas con capas ocultas $h^{(1)}$ y $h^{(2)}$ (con sus respectivas variables contenedoras de pesos $W^{(1)}$ y $W^{(2)}$). Hinton y Bengio [9, 29] establecen que $\log p(x)$ puede ser escrito de la forma:

$$\begin{aligned} \log p(x) &= KL\left(Q\left(h^{(1)}|x\right)||p\left(h^{(1)}|x\right)\right) + H_{Q\left(h^{(1)}|x\right)} \\ &\quad + \sum_{H} Q\left(h^{(1)}|x\right)\left(\log p(h^{(1)}) + \log p(x|h^{(1)})\right) \end{aligned}$$

donde $KL\left(Q\left(h^{(1)}|x\right)||p\left(h^{(1)}|x\right)\right)$ representa la divergencia KL entre la $Q\left(h^{(1)}|x\right)$ posterior de la primer RBM si ésta trabajara por si sola, y la probabilidad $p\left(h^{(1)}|x\right)$ para la misma capa pero definida por la DBN completa (es decir, tomando en cuenta la $p\left(h^{(1)}, h^{(2)}\right)$ anterior definida por la RBM de la capa de hasta arriba). $H_{Q\left(h^{(1)}|x\right)}$ es la entropía de la distribución $Q\left(h^{(1)}|x\right)$.

Puede demostrarse que si se inicializan ambas capas ocultas tal que $W^{(1)T} = W^{(2)}$, $Q\left(h^{(1)}|x\right) = p\left(h^{(1)}|x\right)$ entonces el término de la divergencia KL es cero. Si entonces se entrenara la RBM del primer nivel y se conservaran sus parámetros $W^{(1)}$ fijos, el proceso de optimización de $\log p(x)$ respecto a $W^{(2)}$, solamente podría incrementar la probabilidad la verosimilitud $p(x)$.

Además, se nota que si se aislan los términos que dependen solamente de $W^{(2)}$, se obtiene:

$$\sum_{h} Q\left(h^{(1)}|x\right) p\left(h^{(1)}\right)$$

Optimizar este resultado respecto a $W^{(2)}$ equivale a entrenar una RBM para la segunda etapa utilizando la salida de $Q\left(h^{(1)}|x\right)$ como la distribución de entrenamiento, cuando x es tomada de una distribución de entrenamiento para la primer RBM.

6.4. Algoritmos basados en modelos energéticos restringidos

6.4.1. Máquinas restringidas de Boltzmann

Al igual que con los autoencoders ruidosos, la implementación del proceso de entrenamiento de una RBM se realiza por medio de la construcción de una clase. Esta opción se vuelve bastante útil cuando una RBM es usada como una capa perteneciente a una arquitectura profunda, en cuyo caso, la matriz de pesos y el bias oculto son compartidos como parámetros de una capa sigmoide de una MLP.

La definición de los atributos que tendrá la RBM es importante, se deben incluir a W, b y c, dado que se está utilizando una configuración de pesos atados. Una forma usual de inicializar los atributos de dicha clase sería:

$W = aleatorio(tamaño = [unidades\_visibles, unidades\_ocultas], mínimo, máximo)$

$bias\_visible = zeros(tamaño = [unidades\_visibles])$

$bias\_oculto = zeros(tamaño = [unidades\_ocultas])$

En este caso no existe una variable para la matriz de pesos oculta debido a que una RBM solamente se constituye por una capa de interacción a diferencia de un autoencoder que está formado por dos capas. Los valores *mínimo* y *máximo* corresponden a los límites de la función de distribución de probabilidad utilizada para obtener los valores iniciales de W.

$mínimo = -4 * sqrt\,(6/(unidades\_ocultas + unidades\_visibles))$

$máximo = +4 * sqrt\,(6/(unidades\_ocultas + unidades\_visibles))$

Así, la lista de parámetros que definen a la clase RBM es:

$parámetros = [W, bias\_visibles, bias\_oculto]$

El siguiente paso es definir funciones que construyan las relaciones asociadas con las probabilidades condicionales descritas por 6.2.4:

$prob\_oculta\_dada\_visible = W * visibles + bias\_oculto$

Como se ha mencionado anteriormente, el cálculo de estas probabilidades permite construir una distribución de probabilidad desde la cual se realice el muestreo de nuevas unidades tanto ocultas como visibles. Este proceso es necesario para la realización de un paso del muestreo de Gibbs. El proceso de muestreo de las unidades visibles condicionado al valor de las unidades ocultas (o viceversa) se realiza asignando una probabilidad de éxito dada por $prob\_visible\_dada\_oculta$.

$muestras\_ocultas = binomial(tamaño = unidades\_ocultas,' exito = sigmoide(prob\_oculta\_dada\_visible$

$prob\_visible\_dada\_oculta = transpuesta(W) * muestras\_ocultas + bias\_visible$

$muestras\_visibles = binomial(tamaño = unidades\_visibles,' exito = sigmoide(prob\_oculta\_dada\_oculta$

El proceso de muestreo de Gibbs no está completo hasta la obtención de nuevos valores para las unidades visibles por lo que se utilizan las muestras obtenidas de las unidades ocultas para generar una nueva distribución de probabilidad que permita la creación de estos nuevos valores.

La serie de pasos anterior constituye una etapa de Gibbs de tipo vhv ($visible - hidden - visible$) que comienza dados valores iniciales de las unidades visibles y el cual es necesario para obtener muestras a partir del modelo RBM.

En algunos casos, es necesaria la implementación de un paso de Gibbs de tipo hvh ($hidden - visible - hidden$) que inicia a partir de información en las unidades ocultas y es utilizado para la puesta en marcha de actualizaciones para CD y PCD.

En general, estos procedimientos deberían ser incluidos como métodos de la clase RBM para facilitar el entrenamiento de una hipotética DBN. También es necesario el cálculo de la energía libre que estará denotada por $energía\_libre$ de acuerdo a la ecuación 6.1.4. La siguiente etapa es generar los gradientes necesarios para las actualizaciones de CD y PCD, para ello se debe calcular una cadena de muestreo. Se puede utilizar un mismo procedimiento de código que permita la elección entre procesos de CD y PCD utilizando la variable $persistente$.

$costo = energía\_libre(entrada) - energía\_libre(muestra\_final)$

$param = param + gradiente(costo, param) * tasa$

Las muestras ocultas se calculan a partir de la entrada de entrenamiento al modelo utilizando medio paso de Gibbs como se ha descrito anteriormente. Si la variable $persistente$ es nula, se inicializa la cadena de Gibbs con las muestras ocultas generadas lo cual implicaría el uso de un proceso CD. Una vez establecido el punto inicial de la cadena, se puede calcular la muestra obtenida al final de la cadena de Gibbs la cual es necesaria para el calculo del gradiente de optimización (ver ecuación 6.2.6). Para el cálculo de esta muestra se realiza un proceso iterativo durante k pasos del proceso de muestreo, donde cada paso consiste de un proceso de Gibbs de tipo hvh.

`si` $persistente$`:`

    $inicio\_cadena = persistente$

`caso contrario:`

    $inicio\_cadena = muestras\_ocultas$

La muestra que consideramos como el final de la cadena es la muestra de las unidades visibles del último paso del proceso ($muestra\_final$), con el valor de esta muestra

se procede entonces a calcular la energía libre correspondiente a la fase negativa del gradiente de verosimilitud con lo que conviene calcular primero la función la función de costo del proceso de entrenamiento, la cual es la diferencia existente entre la fase positiva y la fase negativa del proceso de muestreo.

$muestra\_actual = inicio\_cadena$

`desde` $i = 1$ `hasta` $i = k$`:`

$muestra\_Actual = proceso\_hvh(muestra\_actual)$

Esta función de costo es dependiente de todos los parámetros modificables del modelo por lo que un proceso de aprendizaje haría uso del gradiente respecto a los pesos sinápticos y bias durante la actualización paramétrica del modelo por cada iteración del entrenamiento. Sea $gradiente(costo, param)$ una función que calcule el gradiente de $costo$ respecto a $param$, los nuevos valores de los parámetros caracterizados por alguna tasa de aprendizaje se calculan como:

Una vez actualizados los parámetros, se procede a calcular el costo que tienen dichas actualizaciones sobre el comportamiento general del modelo, la definición de la forma de la función de costo depende del proceso de muestreo de Gibbs que se esté utilizando (PC o PCD).

Se observa en el pseudocódigo que si se está utilizando CD se utiliza un error de reconstrucción definido como la entropía cruzada del modelo similar al utilizado durante la optimización de los autoencoders ruidosos, el problema reside en definir cuales serán los valores que tomaremos como predicción del modelo y que serán comparados con la entrada original, [31] propone utilizar como valores de reconstrucción los datos pertenecientes al vector $prob\_visible\_dada\_oculta$ Del último paso de la cadena de Gibbs.

Por otro lado, si se utiliza PCD es necesario utilizar la pseudo-verosimilitud, la cual es calculada utilizando el procedimiento estocástico que termina en la ecuación 6.1.8. Teniendo definidos estos parámetros dentro de la clase de tipo RBM se puede implementar un algoritmo para sus entrenamiento.

`si` $persistente$`:`

$persistente = muestra\_actual$

$costo\_monitoreo = pseudo\_verosimiltud()$

`caso contrario:`

$costo\_monitoreo = error\_reconstruccion()$

Es importante la inicialización de la cadena de Gibbs utilizando la variables $persistente$ para garantizar el uso de PCD, en este caso no se implementa un algoritmo de paro temprano debido a que se está utilizando una solución de tipo no supervisada.

$persistente = x_{batch}[0]$

`mientras (`$epoca < epoca\_total$`) y (`$salir = falso$`):`

$epoca = epoca + 1$

`por cada` $minibatch$ `en el conjunto de entrenamiento:`

$costo = costo(parametros, x_{batch})$

$gradiente = ...calculagradiente$

$parametros = parametros - tasa\_aprendizaje * gradiente$

`si` $perdida <= objetivo$

$salir = verdadero$

`regresar` $parametros$

6.4.2. Redes de creencia profunda (DBN)

El proceso de implementación de una DBN es muy similar al utilizado en un SDA porque ambos involucran el principio de preentrenamiento no supervisado de tipo capa egoísta seguido por un ajuste fino supervisado similar al de una MLP. La diferencia radica en que se utilizan capas del tipo RBM en lugar de autoencoders. Al igual que con los autoencoders apilados, se enlazarán los parámetros de las capas RBM con los pesos y biases de las capas sigmoides que construyen una MLP.

Durante el entrenamiento de los autoencoders ruidosos apilados, no se enlazan los parámetros de la etapa de decodificación con sus correspondientes variables de las capas sigmoides (y así, solo sirven para la etapa de preentrenamiento), el bias visible de la RBM tampoco es sujeto a dicho enlace tomándose solamente como parámetros compartidos la matriz de pesos y el bias oculto.

EL resto del algoritmo de entrenamiento se lleva a cabo de manera similar al caso donde se usan autoencoders pero considerando que el proceso de preentrenamiento es notoriamente mas extenso y costos. Durante el preajuste, se deben tener en cuentas las siguientes consideraciones:

- Se implementa un proceso de muestreo de Gibbs utilizando divergencia constrastiva (CD), es decir, se configurará la variable persistente como nula.
- Los valores recomendados para k (número de pasos de Gibbs) están entre 1 y 10 [31].

`mientras (`$i < número\_capas$`):`

`si` $i = 1$:

$entrada\_capa = entrada$

```
caso contrario:
```

$entrada\_capa = capa\_sigmoide\_anterior.salida$

$i = i + 1$

$capa\_sigmoide = constructor(entrada\_sigmoide = entrada\_capa)$

$capa\_rbm = constructor(entrada\_rbm = entrada\_capa$

$W = capa\_sigmoide.W$

$bias = capa\_sigmoide.bias$

$capa\_sigmoide\_anterior = capa\_sigmoide$

$i = i + 1$

6.5. Aplicación: Clasificación de dígitos MNIST

Una red de creencia profunda es obtenida apilando varias máquinas restringidas de Boltzmann. Cuando se usa para clasificación, una DBN se trata como si fuera una MLP añadiéndole una capa de regresión logística como capa de salida. Para el problema de clasificación MNIST expuesto en la sección 5.4, se optó por una arquitectura profunda compuesta por 3 capas ocultas con 500 unidades cada una (similar a la topología utilizada con los autoencoders ruidosos). El error de reconstrucción a minimizar fue la entropía cruzada al trabajarse con divergencia contrastiva (CD), las tasas de aprendizaje fueron 0.1 y 0.01 para las etapas de ajuste fino y preentenamiento respectivamente. El número k de pasos de Gibbs se fija en 1 mientras que el número de ejemplos por minibatch fue 10; finalmente, el número de épocas de ajuste supervisado máximo se mantiene en 1000 con 20 épocas de preentrenamiento para cada una de las 3 capas del modelo.

Definida la topología a utilizar se procedió con el preentrenamiento de tipo capa egoísta en donde se utilizó la entropía cruzada como la función de error a minimizar por medio del aprendizaje de cada una de las RBMs, el proceso de muestreo fue realizado por medio de un proceso de Gibbs unitario. La evolución del error de preentrenamiento para cada una de las capas se observa en la Figura 6.5.1, el código corrió por 206 minutos:

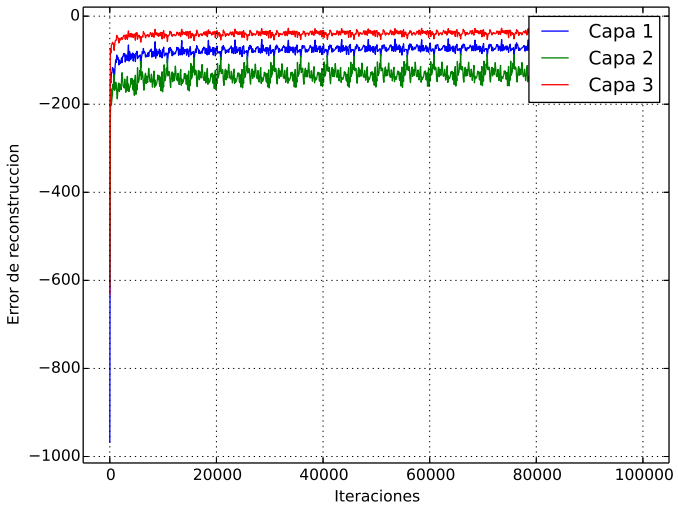


Figura 6.5.1: Evolución de la función de costo de preentrenamiento para cada una de las capas de la DBN

El error medio de validación final fue de 1.52 % con un error de prueba total de 1.61 %. La dinámica del error de entrenamiento (Figura 6.5.2) muestra oscilaciones de menor magnitud a lo presentado por los modelos de regresión logística y Perceptrón multicapa debido a que se utilizaron minibatches con menor número de ejemplos con lo que se elimina en magnitud la varianza respecto al conjunto de entrenamiento.

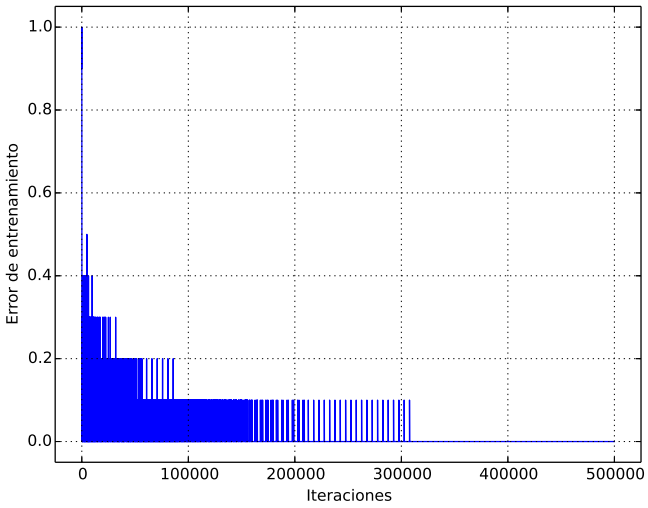
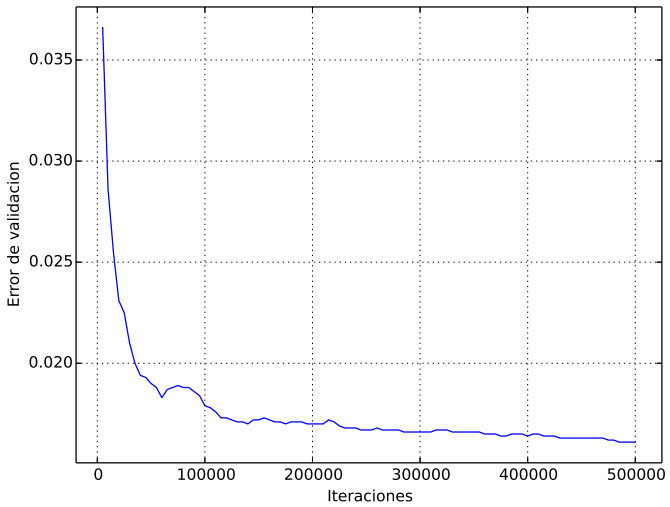
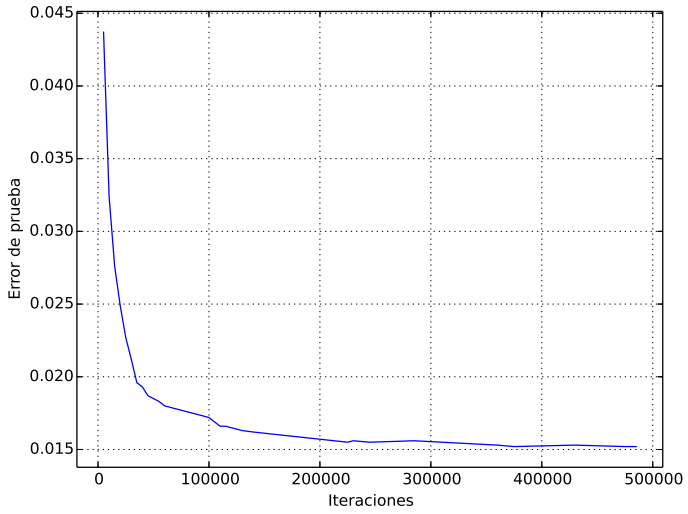


Figura 6.5.2: Evolución del error de entrenamiento durante las iteraciones utilizando minibatches con 10 ejemplos para una DBN

El error de validación y el error de prueba nuevamente presentan comportamientos minimizantes debido al algoritmo de gradiente descendente utilizado (se busca maximizar la log-verosimilitud del modelo), se observa en la Figura 6.5.3 que el valor de error de prueba es mayor que el error de validación durante todas las épocas de entrenamiento. La ejecución del programa de entrenamiento tomó 288 minutos.



a)



b)

Figura 6.5.3: Evolución de los errores de validación y prueba de una DBN respecto al número de iteraciones de entrenamiento

6.6. Aplicación: Identificación de sistemas

6.6.1. Sistema no lineal de primer orden

Se atacó el problema de identificación de sistemas no lineales utilizando como ejemplo el sistema presentado por [41] descrito por la ecuación 6.6.1.

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{6.6.1}$$

donde $u(k)$ es una entrada periódica discreta que cambiará acuerdo a la etapa o proceso de aprendizaje en el que esté el algoritmo de entrenamiento, la entrada del sistema queda descrita por la ecuación 6.6.2.

$$u(k) = A\sin\left(\frac{\pi k}{50}\right) + B\sin\left(\frac{\pi k}{20}\right) \tag{6.6.2}$$

donde los parámetros A y B cambian de acuerdo al propósito que se le esté asignando a la dinámica del sistema, esta variación queda descrita por la Tabla 6.1.

Tabla 6.1: Coeficientes utilizados para la señal de entrada al sistema 1

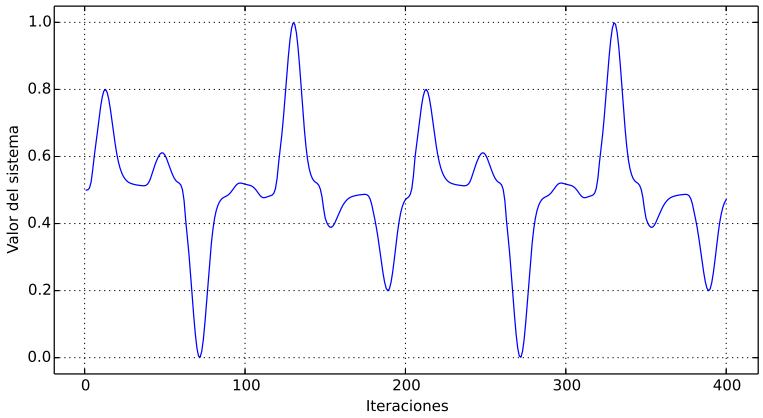
Coeficiente	Entrenamiento	Validación	Prueba
A	1	1.1	0.9
B	1	0.9	1.1

De acuerdo con la Sección 6.1, el proceso de preentremiento (según sean los límites de la integral de convergencia) requiere un conjunto de valores del sistema que se encuentren en el intervalo $[0, 1]$ por lo que se deben normalizar a este intervalo las entradas y salidas del sistema, la actualización a los nuevos valores se da por la ecuación 6.6.3 y la salida del sistema utilizada para entrenamiento es la mostrada en la Figura 6.6.1.

$$nuevoValor = \frac{viejoValor - minimo}{maximo - minimo} \tag{6.6.3}$$

El conjunto de datos de entrenamiento consiste en la dinámica del sistema a lo largo de las primeras 2000 iteraciones mientras que los conjuntos de validación y prueba contienen solamente 400 ejemplos.

a)



b)

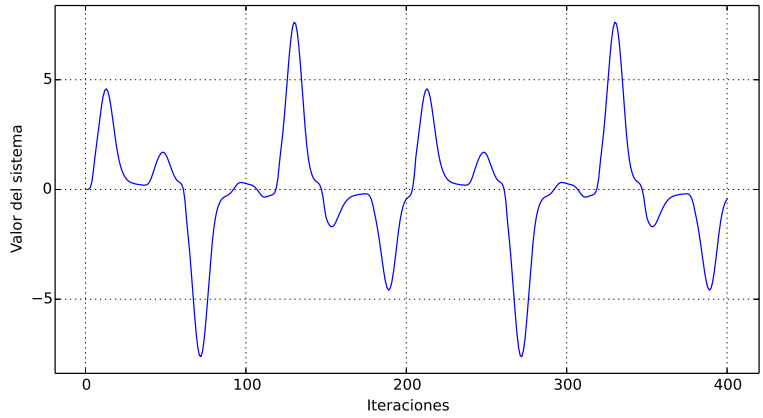


Figura 6.6.1: Gráficas del comportamiento del sistema 1 utilizado por Narendra, a) Sistema normalizado, b) Sistema sin normalizar

Se implementó el algoritmo de entrenamiento utilizando la selección aleatoria de hiperpaŕametros presentada en la Sección 4.2.1 para formar un hiperespacio de elección para el número de unidades y número de capas del modelo. Los límites de cada uno de los hiperparámetros se observan en la Tabla 6.2. El algoritmo de preentrenamiento usado fue el de una arquitectura DBN utilizando 1 paso de Gibbs con ajuste fino por medio de gradiente descendente y paro temprano con mejora de umbral del 5 %. Para ambas etapas se utilizó solamente una época de entrenamiento. Se consideró posibles valores para las unidades visibles en el intervalo continuo de $[0, 1]$.

Tabla 6.2: Rangos de hiperpaŕametros para el sistema 1

	Unidades por capa	Capas	Tasa de preentrenamiento	Tasa de ajuste fino
Mínimo	3	2	0.01	0.1
Máximo	10	6	0.1	0.5

Se graficó la dinámica del error cuadrático medio sobre el conjunto de prueba respecto a la variación de los hiperparámetros, dicho comportamiento se observa en la Figura 6.6.2.

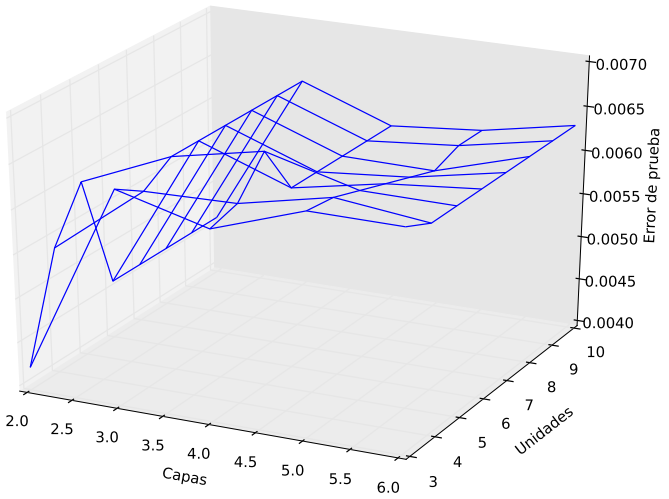


Figura 6.6.2: Dinámica del error de prueba a lo largo del espacio de hiperpaŕametros para el sistema 1

Se observa que utilizando solamente dos capas de identificación se tiene un error mínimo de prueba; sin embargo, se observa también que al aumentar el número de capas y unidades se llega a un nuevo mínimo local que corresponde a una configuración óptima del modelo dentro de una vecindad apropiada del espacio de hiperparámetros (profundidad ideal). Está configuración queda determinada por una arquitectura de 4 capas con 6 unidades por cada capa, la cual se describirá por medio de la notación $6, 6, 6, 6$.

Utilizando está arquitectura se comparan las salidas DBNs variando los intervalos de acción para las unidades visibles (continuos en $[0, 1]$, $[0, \infty]$ y discretos en $(0, 1)$) con los resultados obtenidos por medio de una arquitectura SDA, una MLP de iguales dimensiones pero sin preentreamiento y una MLP con arquitectura $20, 10$ [41].

En el caso del preentrenamiento de la arquitectura SDA, se aprecia en la Figura 6.6.3 que el costo de reconstrucción en la primer capa es el mayor a lo largo de todo el proceso de preentrenamiento, esto se debe a que es la primer capa la que interactúa directamente con el conjunto de datos de entrenamiento.

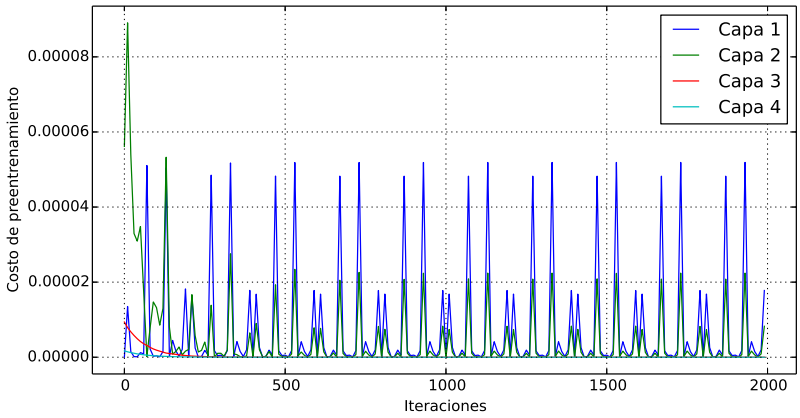


Figura 6.6.3: Evolución del costo durante el preentrenamiento utilizando una arquitectura SDA para el sistema 1

El caso de la etapa de preentrenamiento de la DBN (ver Figura 6.6.4) es diferente, al igual que con la SDA se aprecian valores mayores en la capa 1 pero dentro de la región negativa, esto es debido a la forma de la función de costo de las RBMs, mientras que un autoencoder utiliza para fines de identificación el error cuadrático medio, una RBM hace uso de la función de costo por medio de entropía cruzada.

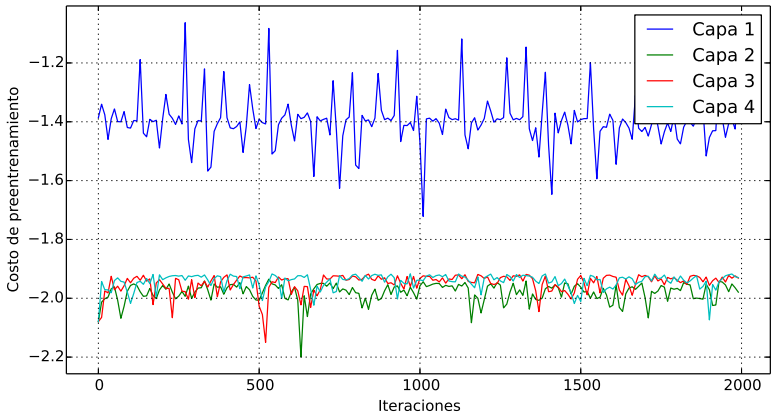


Figura 6.6.4: Evolución del costo durante el preentrenamiento utilizando una arquitectura DBN para el sistema 1

En la Figura 6.6.5, se presenta la dinámica de la salida del modelo durante la fase de ajuste fino, es observable que el modelo alcanza un desempeño favorable después de solamente 10 ejemplos de entrenamiento sin la presencia de sobreajuste. Se expone solamente el desempeño de la DBN debido a que constituye el modelo con menor error de prueba.

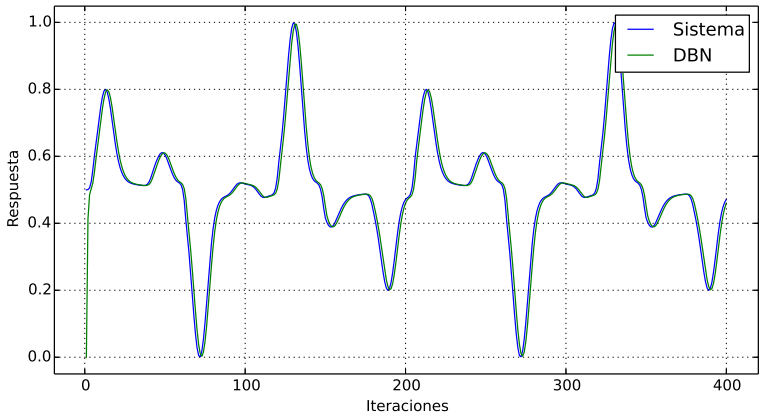


Figura 6.6.5: Etapa supervisada de entrenamiento del sistema utilizando una arquitectura DBN para el sistema 1

Para obtener una orientación visual acerca del verdadero comportamiento del modelo se presenta en la Figura 6.6.6 la salida del sistema y de los modelos ante el conjunto de datos de prueba, se observa que ninguno de los modelos tiene una salida completamente traslapable con la original pero la DBN (arquitectura $6, 6, 6, 6$) es más parecida, la salida de la MLP con arquitectura $20, 10$ es apenas ligeramente menos satisfactoria, pero se debe tomar en cuenta que esta arquitectura tiene un menor número de capas con lo que evita el problema de desvanecimiento de la señal del gradiente durante el ajuste fino. Este problema de desvanecimiento se resuelve en la DBN por medio del preentrenamiento, por último se observa también la salida de la MLP sin preentrenamiento con arquitectura $6, 6, 6, 6$ que ha quedado estancada en un mínimo local por lo que presenta un pobre desempeño comparada con su contraparte preentrenada (la DBN).

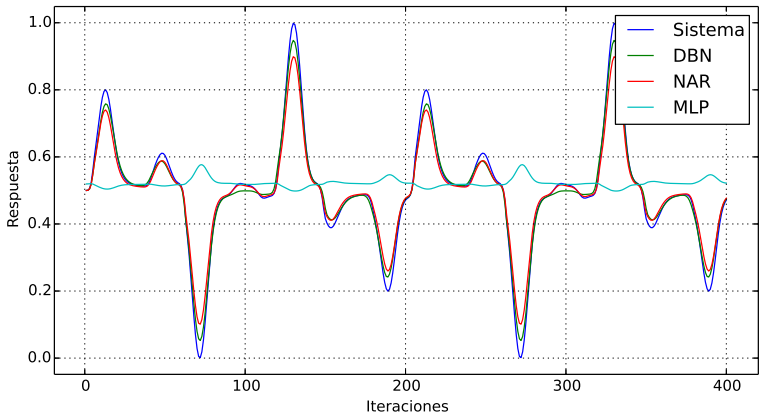


Figura 6.6.6: Salida de los modelos utilizando el sistema 1 normalizado

Finalmente en la Figura 6.6.7 se exponen las salidas de las mismas arquitecturas pero

utilizando datos del sistema sin normalizar, para esto se utilizaron unidades visibles en las RBMs en el intervalo de $[-8, 8]$, en general se aprecia un comportamiento similar al visto en el caso normalizado.

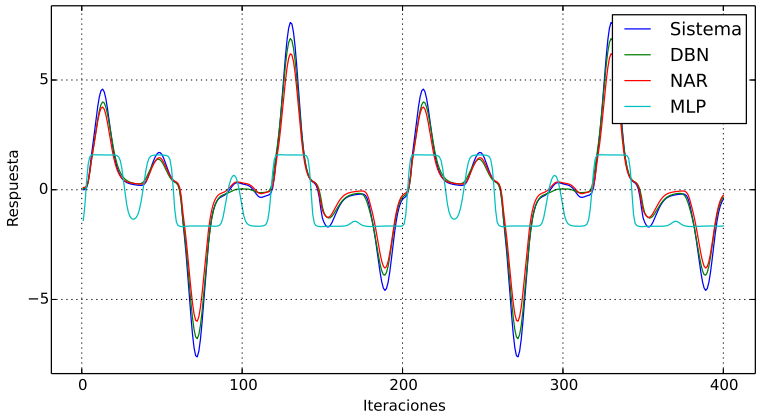


Figura 6.6.7: Salida de los modelos utilizando el sistema 1 sin normalizar

En la Tabla 6.3 se muestra el costo total de prueba para cada una las arquitecturas probadas. Es notable que el mejor desempeño se encuentra utilizando una DBN en el intervalo continuo $[0, 1]$ seguida por la DBN binaria en $[0, 1]$, los costos de la SDA y la DBN continua en $[-8, 8]$ son ligeramente mayores pero mejores al obtenido con la arquitectura $20, 10$, finalmente la MLP sin preentrenamiento y la DBN continua en $[0, \infty]$ tienen los peores comportamientos, esto se explica porque la MLP cae en un mínimo local al no ser sujeta a la optimización inicial de una RBM y la DBN no cumple la condición necesaria de existencia del denominador integral del proceso de muestreo por lo que los pesos sinápticos arrojados por el preentrenamiento no son los óptimos.

Tabla 6.3: Desempeño de prueba para distintas arquitecturas durante la identificación del sistema 1

	NAR$20, 10$	MLP	SDA
Costo($\times 10^{-3}$)	7.477	16.158	6.652

DBN	Binaria$[0, 1]$	Continua $[0, \infty]$	Continua $[0, 1]$	Continua $[-8, 8]$
Costo ($\times 10^{-3}$)	5.972	31.546	5.915	6.862

6.6.2. Sistema no lineal con término lineal

Una vez inspeccionado el comportamiento de una arquitectura profunda en la tarea de identificación de sistemas, se cambia el sistema a identificar por uno en la cual la dinámica de la entrada tenga tanto un componente lineal como un componente no lineal [41], la ecuación 6.6.4 describe este nuevo sistema.

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) + (u(k)+1)u(k)(u(k)-1) \tag{6.6.4}$$

donde $u(k)$ queda inmersa en una función $g(u)$ no lineal que puede ser separada en dos partes diferenciadas $u^3(k)$ y $u(k)$ dotando al sistema no lineal de una componente lineal, $u(k)$ varia dependiendo de cual conjunto de datos se este generando y está dada por la ecuación 6.6.5 al igual que en el Sistema 1.

$$u(k) = A\sin\left(\frac{\pi k}{50}\right) + B\sin\left(\frac{\pi k}{20}\right) \tag{6.6.5}$$

donde los parámetros A y B cambian de acuerdo al propósito que se le esté asignando a la dinámica del sistema, esta variación queda descrita por la Tabla 6.4, se nota que la variación entre los coeficientes es de magnitud mayor que en el sistema 1 haciendo del conjunto de prueba y del conjunto de entrenamiento elementos que necesitan mayor capacidad de generalización por parte del modelo para su identificación.

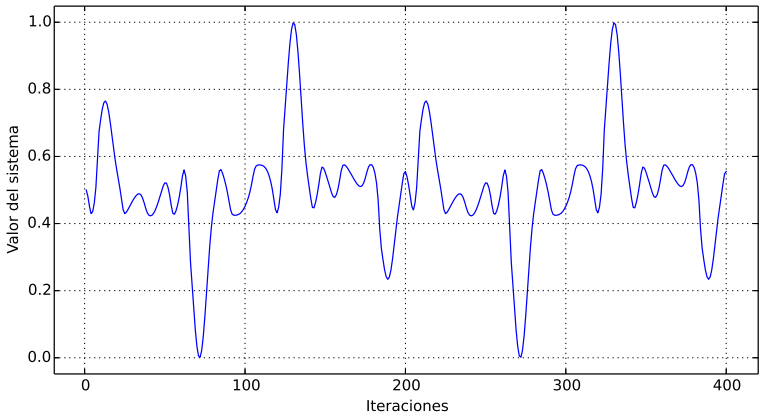
Tabla 6.4: Coeficientes utilizados para la señal de entrada al sistema 2

Coeficiente	Entrenamiento	Validación	Prueba
A	1	1.2	0.8
B	1	0.8	1.2

El proceso de preentrenamiento para el tipo de unidades visibles encontradas en tareas de clasificación (discretas en el rango $(0, 1)$) requiere un conjunto de valores del sistema que se encuentren en el intervalo $[0, 1]$ por lo que se normalizaron dentro de este intervalo las entradas y salidas del sistema, la actualización a los nuevos valores se da por la ecuación 6.6.3 y la salida del sistema utilizada para entrenamiento es la mostrada en la Figura 6.6.8.

Utilizando como ejemplo la base de datos MNIST en donde los conjuntos de validación y prueba constituyen una quinta parte de los ejemplos de entrenamiento se crea un conjunto de 2000 ejemplos de entrenamiento y 2 conjuntos de 400 elementos cada uno para fines de validación y prueba.

a)



b)

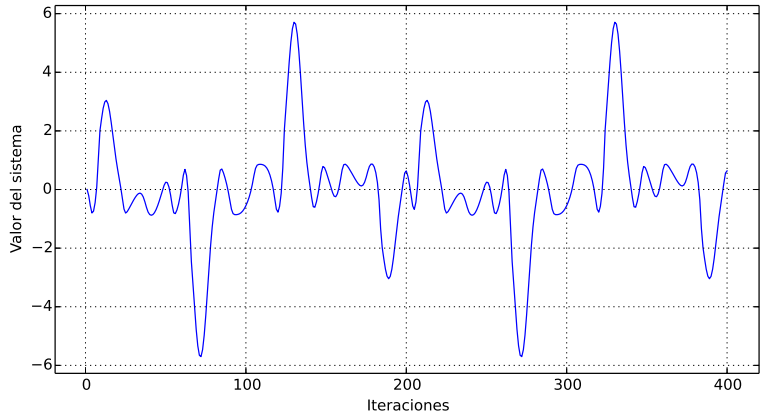


Figura 6.6.8: Gráficas del comportamiento del sistema 2 utilizado por Narendra, a) Sistema normalizado, b) Sistema sin normalizar

Algo muy importante es la elección de la arquitectura a utilizar, para ello se implementó el algoritmo de selección aleatoria de hiperpaŕametros. Los rangos de variabilidad para las distribuciones de probabilidad para cada uno de los hiperparámetros se observan en la Tabla 6.5. Se eligió como algoritmo de preentrenamiento utilizado aquel que dio mejores resultados durante la identificación del sistema 1, específicamente una arquitectura DBN utilizando 1 paso de Gibbs con ajuste fino por medio de gradiente descendente y paro temprano con mejora de umbral del 5 %. Solamente se utilizó una época de preentrenamiento y una época de ajuste fino del modelo, considerando unidades visibles de tipo continuo en el intervalo $[0, 1]$.

Tabla 6.5: Rangos de hiperpaŕametros para el sistema 2

	Unidades por capa	Capas	Tasa de preentrenamiento	Tasa de ajuste fino
Mínimo	2	1	0.01	0.1
Máximo	11	6	0.1	0.5

Se graficó la superficie generada por la dinámica del error cuadrático medio de identificación del conjunto de prueba sobre espacio de hiperparámetros, dicha malla se observa en la Figura 6.6.9.

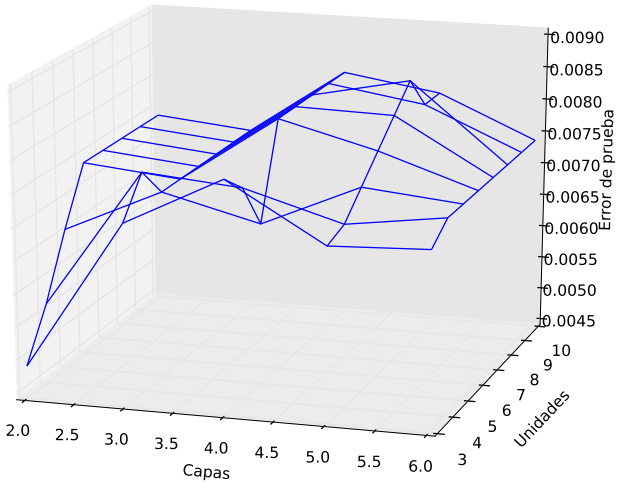


Figura 6.6.9: Dinámica del error de prueba a lo largo del espacio de hiperpaŕametros del sistema 2

La DBN ayuda a disminuir el efecto del problema de la dimensionalidad presente en los modelos con arquitecturas profundas. Se observa que utilizando solamente dos capas de identificación se tiene un error mínimo de prueba; sin embargo, en la zona media del la superficie se nota que también al aumentar el número de capas y unidades se llega a un nuevo mínimo local de la magnitud del error (profundidad ideal) estudiado. Está configuración queda determinada por una arquitectura de 4 capas con 5 unidades por cada capa, la cual se referenciará por medio de la notación $5, 5, 5, 5$.

Utilizando está arquitectura se comparan las salidas DBNs variando los intervalos de acción para las unidades visibles (continuos en $[0, 1]$, $[0, \infty]$ y discretos en $(0, 1)$) con los resultados obtenidos por medio de una arquitectura SDA, una MLP de iguales dimensiones pero sin preentrenamiento y una MLP con arquitectura $20, 10$[41].

En el caso del preentrenamiento de la arquitectura con autoencoders apilados, se aprecia en la Figura 6.6.10 que el costo de reconstrucción en la primer capa es el mayor a lo largo de todo el proceso de preentrenamiento, esto se debe a que es la primer capa la que tiene como tarea modelar la distribución de probabilidad de los datos de entrada que ingresan al modelo profundo como ya se había explicado en el análisis del sistema 1.

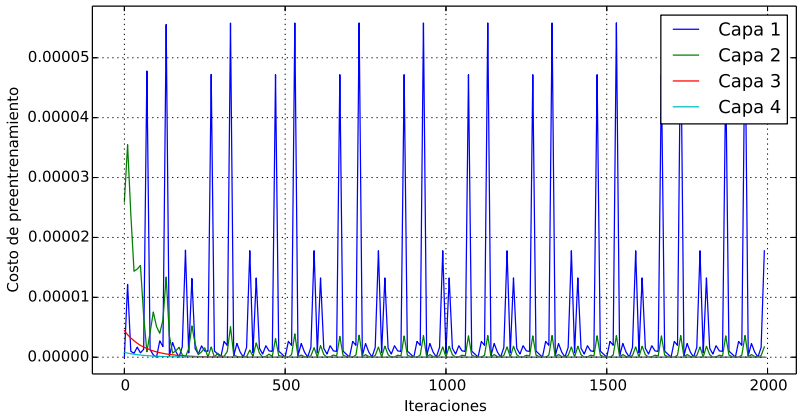


Figura 6.6.10: Evolución del costo durante el preentrenamiento utilizando una arquitectura SDA para el sistema 2

El caso de la etapa de preentrenamiento de la DBN (ver Figura 6.6.11) al igual que con la SDA se aprecian valores mayores en la capa 1 pero dentro de la región negativa, esto es debido a la forma de la función de costo de las RBMs, mientras que un autoencoder utiliza para fines de identificación el error cuadrático medio, una RBM hace uso de la función de costo por medio de entropía cruzada, la cual teóricamente consiste en la minimización de la función negativa de verosimiltud del modelo.

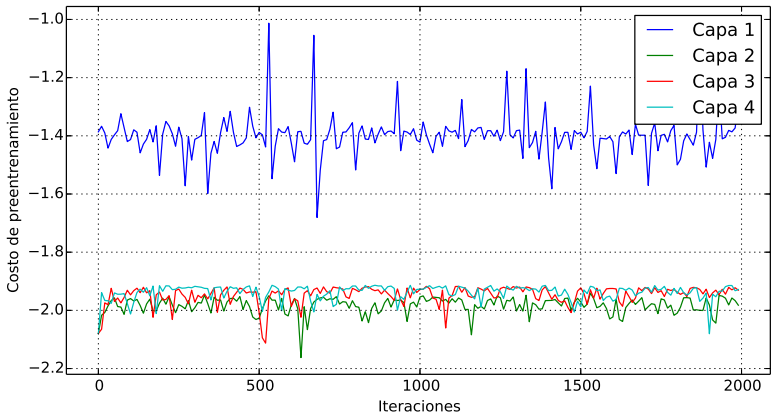


Figura 6.6.11: Evolución del costo durante el preentrenamiento utilizando una arquitectura DBN para el sistema 2

En la Figura 6.6.12, se presenta la dinámica de la salida del modelo utilizando RBMs en el preentrenamiento durante la fase de ajuste fino, es apreciable que la convergencia al sistema real se da solamente pasadas 15 iteraciones del sistema, posteriormente no existe disminución apreciable en el error de identificación.

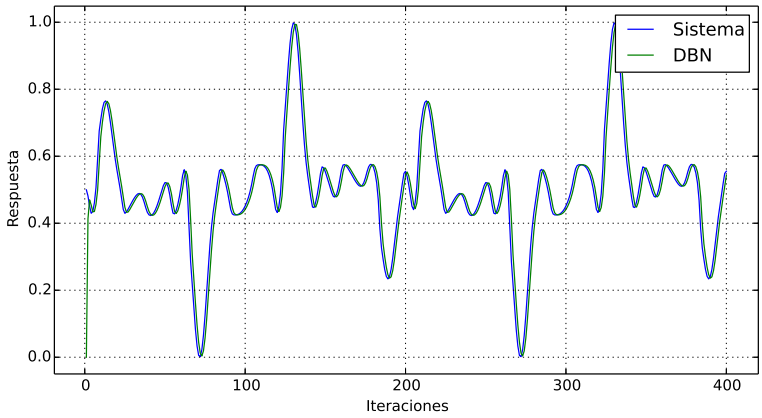


Figura 6.6.12: Etapa supervisada de entrenamiento del sistema utilizando una arquitectura DBN para el sistema 2

Algo importante es comparar la salida del modelo con la ofrecida por otras arquitecturas y técnicas de entrenamiento, en la Figura 6.6.13 se observa que ninguno de los modelos tiene una salida completamente traslapable con la original pero la DBN (arquitectura $5, 5, 5, 5$) es la más parecida, la salida de la MLP con arquitectura $20, 10$ tiene un desempeño también sobresaliente a pesar de no ser sujeta a ningún ajuste inicial de pesos, esto se debe a que la señal del gradiente es de magnitud considerable durante la etapa de ajuste fino, el problema de desvanecimiento del gradiente se resuelve en la DBN por medio del preentrenamiento, por último se observa también la salida de la MLP sin preentrenamiento con arquitectura $5, 5, 5, 5$ que ha quedado estancada en un mínimo local por lo que presenta un pobre desempeño comparada con su contraparte preentrenada (la DBN).

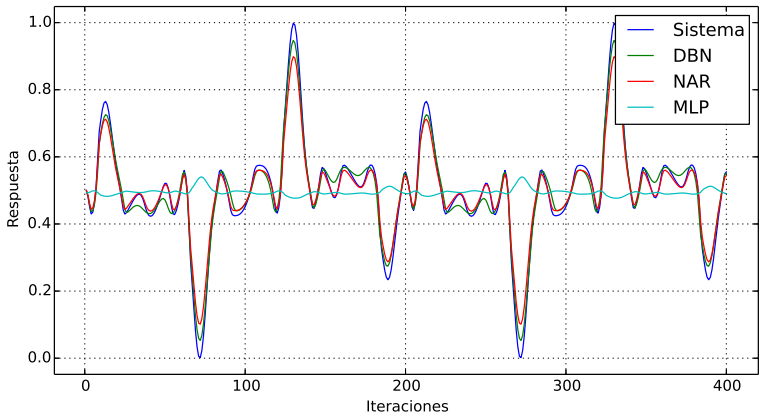


Figura 6.6.13: Salida de los modelos utilizando el sistema 2 normalizado

Finalmente en la Figura 6.6.14 se exponen las salidas de las mismas arquitecturas pero

utilizando datos del sistema sin normalizar, para esto se utilizaron unidades visibles en las RBMs en el intervalo de $[-6, 6]$, se aprecia como el modelo de arquitectura pequeña $(20, 10)$ es capaz de seguir de mejor manera al sistema en eventos de frecuencia grande y magnitud pequeña, pero en eventos de magnitud grande la DBN funciona mejor.

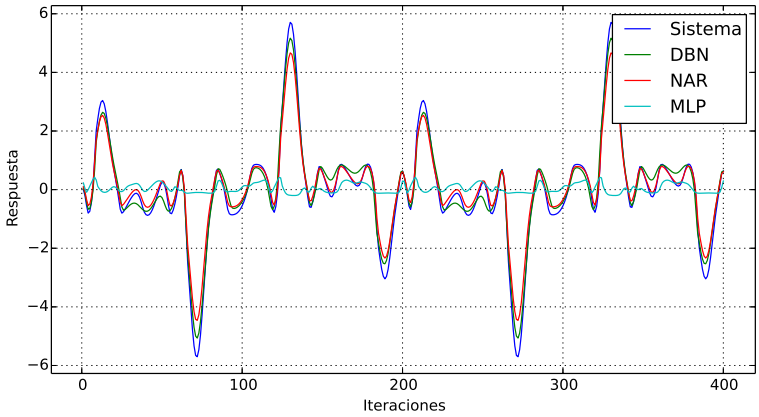


Figura 6.6.14: Salida de los modelos utilizando el sistema 2 sin normalizar

En la Tabla 6.6 se muestran los costos totales de prueba para cada una las arquitecturas probadas. Es notable que el mejor desempeño se encuentra utilizando una DBN en el intervalo continuo $[0, 1]$ seguida por la DBN binaria en $[0, 1]$, esto quiere decir que la mejora conseguida al pasar de unidades binarias a continuas existe, pero no es notoriamente apreciable al medir la magnitud final de la función de costo, los costos de la SDA y la DBN continua en $[-6, 6]$ son ligeramente mayores pero mejores al obtenido con la arquitectura $20, 10$, finalmente la MLP sin preentrenamiento y la DBN continua en $[0, \infty]$ tienen los peores comportamientos, esto se da porque la MLP cae en un mínimo local y sufre el fenómeno conocido como ensombrecimiento, que consiste en seguir la señal de entrenamiento pero sin converger en algún conjunto de pesos sinápticos determinado y la DBN no cumple la condición necesaria de existencia del denominador integral del proceso de muestreo por lo que los pesos sinápticos arrojados por el preentrenamiento no logran evadir la existencia de mínimos locales.

Tabla 6.6: Desempeño de prueba para distintas arquitecturas durante la identificación del sistema 2

	NAR$20, 10$	MLP	SDA
Costo$(\times 10^{-3})$	10.363	20.184	8.627

DBN	Binaria$[0, 1]$	Continua $[0, \infty]$	Continua $[0, 1]$	Continua $[-6, 6]$
Costo $(\times 10^{-3})$	7.026	25.738	6.827	8.563

6.6.3. Sistema no lineal de segundo orden

Se ha corroborado un mejor desempeño de las arquitecturas profundas, para investigar mas profundamente acerca de su comportamiento en la tarea de identificación se propone un sistema no lineal de segundo orden descrito por la ecuación 6.6.6:

$$y(k+1) = \frac{y(k) + y(k-1)}{1 + y^2(k) + y^2(k-1)} + u^3(k) + (u(k)+1)u(k)(u(k)-1) \tag{6.6.6}$$

donde $u(k)$ representa la entrada al sistema y está dada por la ecuación 6.6.7, se aprecia que la frecuencia de la señal es diferente a la utilizada en los dos sistemas de primer orden.

$$u(k) = A\sin\left(\frac{\pi k}{50}\right) + B\sin\left(\frac{\pi k}{30}\right) \tag{6.6.7}$$

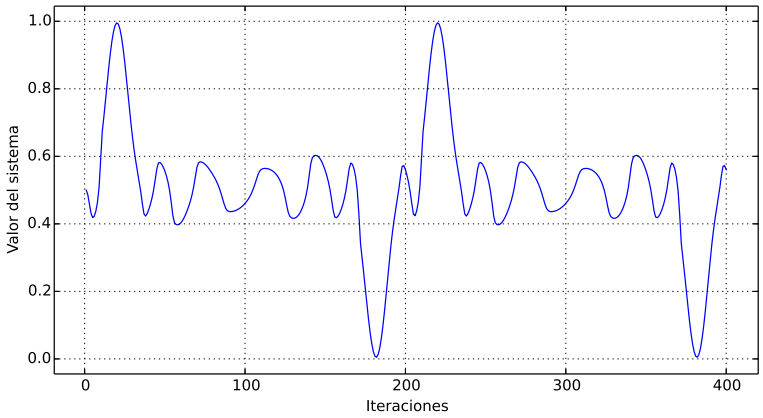
En la ecuación 6.6.7 los parámetros A y B cambian de acuerdo al propósito que se le esté asignando a la dinámica del sistema, esta variación queda descrita por la Tabla 6.7.

Tabla 6.7: Coeficientes utilizados para la señal de entrada al sistema 3

Coeficiente	Entrenamiento	Validación	Prueba
A	1	0.9	0.8
B	1	0.8	1.1

El proceso de preentrenamiento para el tipo de unidades visibles encontradas en tareas de clasificación (discretas en el rango $(0, 1)$) requiere un conjunto de valores del sistema que se encuentren en el intervalo $[0, 1]$ por lo que se normalizó dentro de este intervalo las entradas y salidas del sistema, la actualización a los nuevos valores se da por la ecuación 6.6.3 y la salida del sistema utilizada para entrenamiento es la mostrada en la Figura 6.6.15.

a)



b)

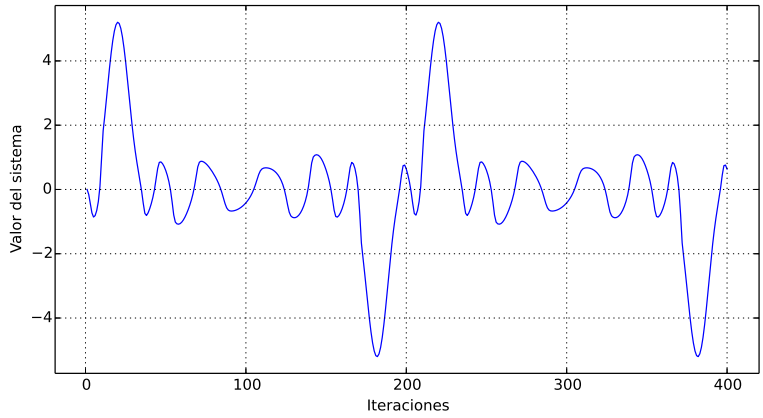


Figura 6.6.15: Gráficas del comportamiento del sistema 3 a) Sistema normalizado, b) Sistema sin normalizar

Al igual que en los sistemas anteriores se utilizaron 2000 ejemplos de entrenamiento y 400 ejemplos de validación y prueba. Los rangos de variabilidad para las distribuciones de probabilidad utilizadas en el algoritmo de selección aleatoria de cada hiperparámetro se dan en la Tabla 6.8. Para fines de prueba de trabajo con una DBN con paso de Gibbs igual a 1 y condición de mejora de ajuste fino de 5 %.

Tabla 6.8: Rangos de hiperpaŕametros para el sistema 3

	Unidades por capa	Capas	Tasa de preentrenamiento	Tasa de ajuste fino
Mínimo	3	2	0.012	0.1
Máximo	10	6	0.15	0.5

La superficie generada por la elección aleatoria se observa en la Figura 6.6.16.

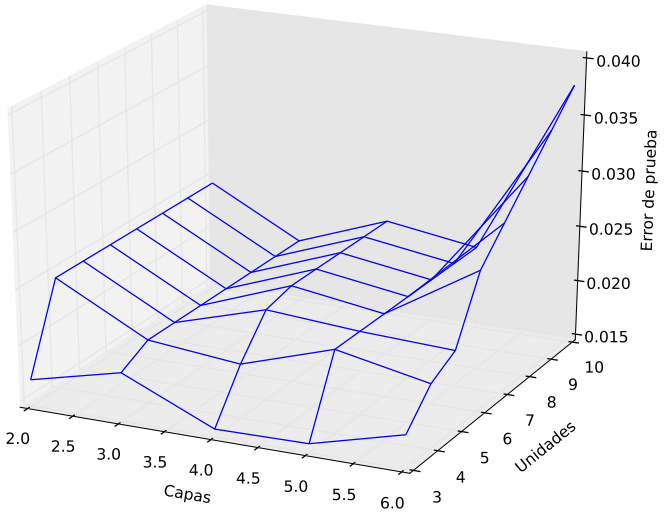


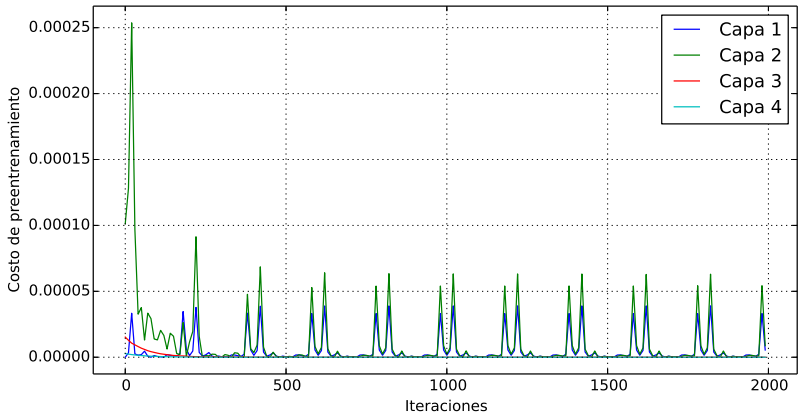
Figura 6.6.16: Dinámica del error de prueba a lo largo del espacio de hiperpaŕametros del sistema 3

Se observa nuevamente que al aumentar el número de capas y unidades se llega a un nuevo mínimo local (al igual que en los sistemas de primer orden) que corresponde a la configuración óptima del modelo. Está configuración queda determinada por una arquitectura de 4 capas con 3 unidades por cada capa, la cual se describirá por medio de la notación $3, 3, 3, 3$.

Utilizando está arquitectura se comparan las salidas DBNs variando los intervalos de acción para las unidades visibles (continuos en $[0, 1]$, $[0, \infty]$ y discretos en $(0, 1)$) con los resultados obtenidos por medio de una arquitectura SDA, una MLP de iguales dimensiones pero sin preentrenamiento y una MLP con arquitectura $20, 10$.

En el caso de las fases de preentrenamiento tanto para la SDA como para la DBN, ésta etapa se ejecutó por 6 segundos y 11 segundos respectivamente, lo cual es claro dada la diferencia de complejidad computacional entre los dos algoritmos, se ve en la Figura 6.6.17 que el costo de reconstrucción en la primer capa es el mayor a lo largo de todo el proceso de preentrenamiento en ambos casos.

a)



b)

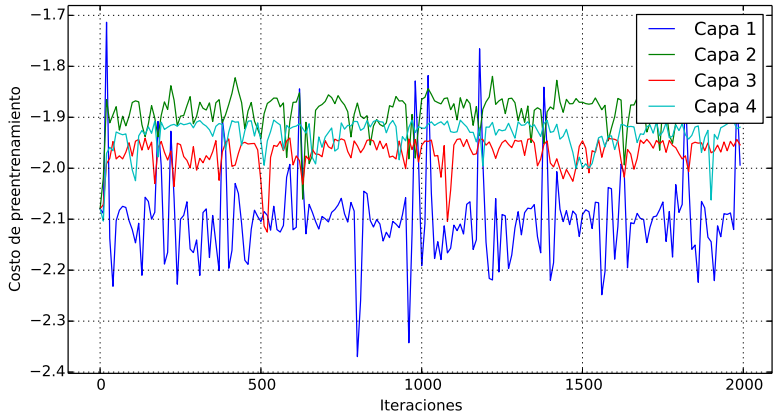


Figura 6.6.17: Evolución del costo durante el preentrenamiento del sistema 3 a)SDA b)DBN

En la Figura 6.6.18, se expone la dinámica del proceso de entrenamiento de la DBN, se expone está al ser la que tiene menor costo de prueba. Cabe señalar que el proceso de entrenamiento supervisado duro 4 segundos para la arquitectura $3, 3, 3, 3$ y 10 segundos para el modelo $20, 10$, lo cual sugiere que al tener menos unidades por capa el algoritmo de optimización por gradiente descendente es mas rápido en la arquitectura profunda.

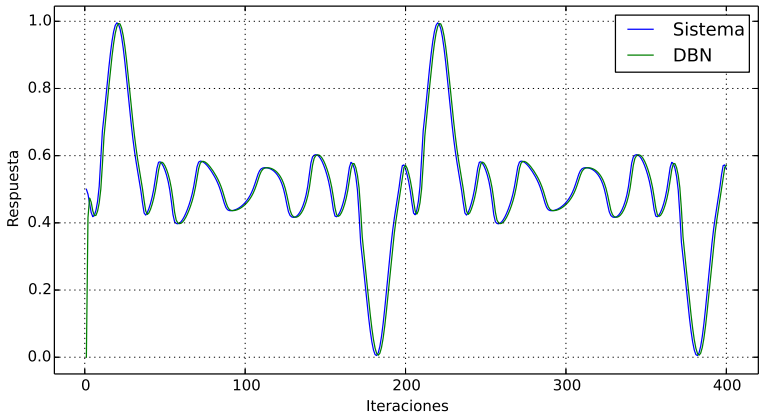


Figura 6.6.18: Etapa supervisada de entrenamiento del sistema utilizando una arquitectura DBN para el sistema 3

En la Figura 6.6.19 se observa que la DBN (arquitectura $3, 3, 3, 3$) es el modelo con mejor correlación al sistema, al igual que en los sistemas anteriores, la MLP con arquitectura $20, 10$ tiene un comportamiento similar mientras que la salida de la MLP sin preentrenamiento con arquitectura $3, 3, 3, 3$ se ha quedado estancada en un mínimo local.

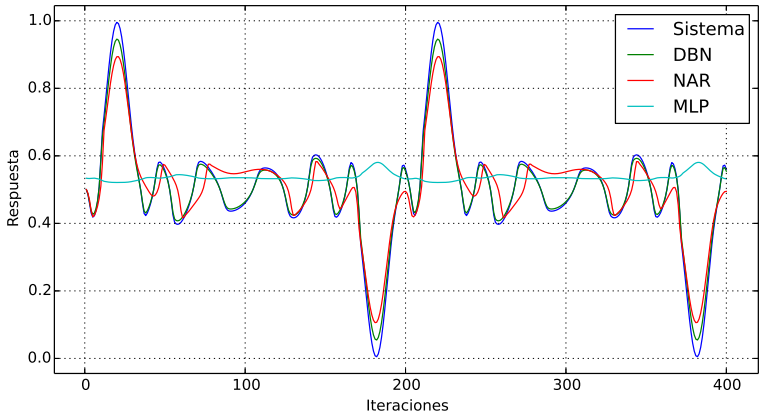


Figura 6.6.19: Salida de los modelos utilizando el sistema 3 normalizado

Finalmente en la Figura 6.6.20 se exponen las salidas de las mismas arquitecturas pero utilizando datos del sistema sin normalizar, para esto se utilizaron unidades visibles en las RBMs en el intervalo de $[-5, 5]$, en general se aprecia un comportamiento similar al visto en el caso normalizado.

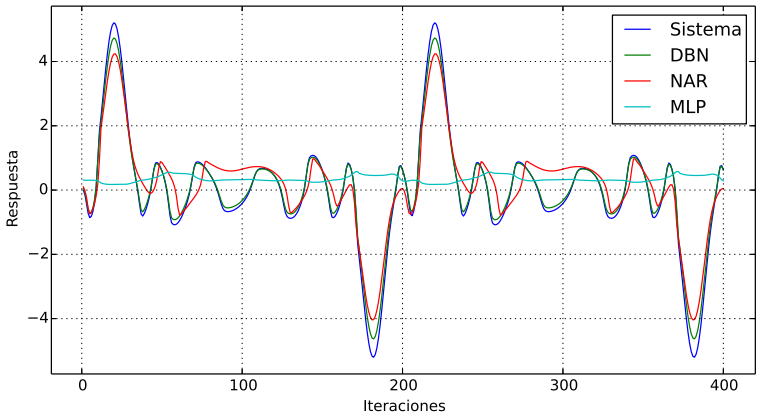


Figura 6.6.20: Salida de los modelos utilizando el sistema 3 sin normalizar

En la Tabla 6.9 se muestran los costos totales de prueba para cada una las arquitecturas probadas. Es notable que el mejor desempeño se encuentra nuevamente utilizando una DBN en el intervalo continuo $[0, 1]$ seguida por la DBN binaria en $[0, 1]$, esto quiere decir que la mejora conseguida al pasar de unidades binarias a continuas existe pero no es notoriamente apreciables al medir la magnitud final de la función de costo, los costos de la SDA y la DBN continua en $[-5, 5]$ son ligeramente mayores pero mejores al obtenido con la arquitectura $20, 10$, finalmente la MLP sin preentrenamiento y la DBN continua en $[0, \infty]$ tienen los peores desempeños..

Tabla 6.9: Desempeño de prueba para distintas arquitecturas durante la identificación del sistema 3

	NAR$20, 10$	MLP	SDA
Costo$(\times 10^{-3})$	21.738	51.945	18.243

DBN	Binaria$[0, 1]$	Continua $[0, \infty]$	Continua $[0, 1]$	Continua $[-5, 5]$
Costo $(\times 10^{-3})$	16.123	41.950	15.172	18.835

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

- Se presentan a las arquitecturas profundas como una alternativa en la solución de los problemas de clasificación de patrones e identificación de sistemas dinámicos, se exponen las dificultades históricas que han tenido diversos algoritmos de aprendizaje al ser aplicados sobre un número elevado de capas de abstracción del conjunto de datos de entrada. Se propone como alternativa someter al modelo neuronal a una etapa de preentrenamiento con el objetivo de evitar problemas de sobreajuste y de desvanecimiento de la señal del gradiente de la función de costo durante la etapa supervisada.

- En la presente tesis se estudió la capacidad de ajuste de pesos sinápticos proveniente del algoritmo de gradiente descendente aplicado a una arquitectura profunda, se comparó el comportamiento final del modelo obtenido al ser aplicado un proceso de preentrenamiento, encontrándose un índice de error de predicción menor en este último caso, lo que sugiere que el preentrenamiento ha movido al conjunto inicial de pesos sinápticos a una zona mas favorable del espacio de parámetros.

- Se identificaron las diferencias existentes entre la solución del problema de clasificación y el problema de identificación utilizando como bloque constructor máquinas restringidas de Boltzmann. En el caso de clasificación se encontró que es suficiente considerar entradas visibles binarias al modelo mientras que esta consideración arroja predicciones pobres cuando se considera la tarea de identificación de sistemas con salidas y entradas continuas, también se observó que el dominio considerado del espacio de trabajo para las unidades visibles debe ser personalizado para cada sistema para lograr un mejor desempeño por parte de la etapa de preentrenamiento, de esta manera, se deben especificar siempre que se pueda el rango de valores sobre los que se desempeña el sistema a identificar.

- Abordar los problemas de clasificación e identificación utilizando una arquitectura profunda fue posible encontrándose una mejora mas significativa en el caso de clasificación de patrones, en este problema inclusive, se redujo el índice de error obtenido al compararse con el arrojado por una arquitectura de una o dos capas. Por otro lado, la implementación del aprendizaje profundo para identificación tuvo reducciones en el error cuadrático medio obtenido para las arquitecturas probadas encontrándose arquitecturas óptimas para cada uno de los sistemas identificados; esto sugiere que para cada problema en particular existirá una profundidad ideal para su solución lo que se traduciría en la disminución del costo computacional total de la aplicación.

- Durante la implementación de los algoritmos de entrenamiento se utilizó la filosofía de capa egoísta utilizándose como bloques constructores de la arquitectura profunda autoencoders y máquinas restringidas de Boltzmann. Se encontró que con ambos tipos de bloques constructores el modelo mejora su capacidad de identificación respecto al que no ha sido sujeto a preentrenamiento, específicamente, las máquinas restringidas de Boltzmann con entradas continuas positivas acotadas demostraron tener el mejor desempeño en algunas de sus variantes mientras que el comportamiento de los autoencoders es similar al obtenido por medio de RBMs con unidades visibles en el rango de los números positivos y mejor al de las RBMs con entradas con dominio no acotado, es decir, entradas variantes en todo el conjunto de números reales.

- A pesar del mejor comportamiento de las RBMs sobre los autoencoders, el costo computacional del preentrenamiento por RBMs fue aproximadamente dos veces que el requerido por los autoencoders, para fines de identificación está diferencia es de algunos segundos pero para fines de clasificación de patrones con vectores de entrada del orden de cientos de elementos, la diferencia en tiempos se llega a traducir en horas por lo que se debe realizar un análisis costo-beneficio para cada aplicación en particular tomando en cuenta los recursos computacionales disponibles.

- Finalmente, debido a que la etapa de preentrenamiento es utilizada como herramienta para la mejora del algoritmo supervisado, es necesario tener a priori un conjunto de datos del sistema, esto tiene como consecuencia que una implementación del algoritmo de aprendizaje profundo totalmente en línea queda fuera de los alcances de las técnicas propuestas en el presente trabajo.

7.2. Trabajo Futuro

EL aprendizaje profundo tiene en la actualidad un gran número de trabajos en la literatura pero estos están orientados de manera casi unívoca al problema de clasificación,

de esta manera, existe un campo de oportunidad inmenso en el área de la identificación y control de sistemas dinámicos, algunos de los tópicos por investigar se enlistan a continuación:

- En el presente escrito, se ha utilizado solamente una topología de identificación de sistemas, se deben abordar otras estrategias para comparar su desempeño con el expuesto aquí.
- Como se ha dicho, la relación de orden entre el preentrenamiento y el ajuste fino del modelo no permiten una implementación en línea por lo que se debe trabajar en el desarrollo teórico de un algoritmo iterativo que desarrolle ambas etapas simultáneamente, es decir, por cada muestra del estado del sistema se avance en el ajuste de ambas fases.
- Solamente se ha considerado de unidades visibles de tipo continuo, se debe generalizar este concepto a las unidades ocultas lo que implica el cálculo teórico de una nueva función de energía para las RBMs, se necesita además estudiar las condiciones de convergencia del algoritmo de muestreo cuando se consideran entradas continuas cuyo dominio es desconocido.
- Implementar algoritmos distintos al gradiente descendente estocástico durante la etapa de ajuste fino para verificar las consecuencias del preentrenamiento para cada uno de ellos.
- Aplicar técnicas de aprendizaje profundo para la generación de señales de control de sistemas en tareas como seguimiento de trayectorias, control de velocidad y seguimiento de señales de referencia.
- En [21] se sugiere que una red neuronal recurrente puede verse como una red profunda “enrrollada” en el tiempo, se sugiere estudiar las consecuencias de la implementación de etapas de preentrenamiento como ajuste inicial de los parámetros de estas redes.
- Por último es necesario implementar otras técnicas de elección de hiperpaŕametros como las sugeridas en [10].

Bibliografía

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," Cognitive Science, vol. 9, pp. 147–169, 1985.

[2] J. Baxter, "A Bayesian/information theoretic model of learning via multiple task sampling," Machine Learning, vol. 28, pp. 7–40, 1997.

[3] Y. Bengio and O. Delalleau, "Justifying and generalizing contrastive divergence," Neural Computation, vol. 21, no. 6, pp. 1601–1621, 2009.

[4] Y. Bengio, O. Delalleau, and N. Le Roux, "The Curse of highly variable functions for local kernel machines," in Advances in Neural Information Processing Systems 18 (NIPS'05), (Y. Weiss, B. Sch¨olkopf, and J. Platt, eds.), pp. 107– 114, Cambridge, MA: MIT Press, 2006.

[5] Y. Bengio, O. Delalleau, and C. Simard, "Decision trees do not generalize to new variations," Computational Intelligence, To appear, 2009.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in Advances in Neural Information Processing Systems 19 (NIPS'06), (B. Sch¨olkopf, J. Platt, and T. Hoffman, eds.), pp. 153– 160, MIT Press, 2007.

[7] Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte, "Convex neural networks," in Advances in Neural Information Processing Systems 18 (NIPS'05), (Y. Weiss, B. Sch¨olkopf, and J. Platt, eds.), pp. 123–130, Cambridge, MA: MIT Press, 2006.

[8] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," in Large Scale Kernel Machines, (L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds.), MIT Press, 2007.

[9] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 1994.

[10] J. Bergstra, Y. Bengio, "Algorithms for Hyper-Parameter Optimization", in Journal of Machine Learning Research, pp 201-210 (2012)

[11] J. Bergstra, Y. Bengio, "Random Search for Hyper-Parameter Optimization", in Journal of Machine Learning Research, pp 281-305 (2011)

[12] J. Bergstra and Y. Bengio, "Slow, decorrelated features for pretraining complex cell-like networks," in Advances in Neural Information Processing Systems 22 (NIPS'09), (D. Schuurmans, Y. Bengio, C. Williams, J. Lafferty, and A. Culotta, eds.), December 2010.

[13] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," Biological Cybernetics, vol. 59, pp. 291–294, 1988.

[14] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.

[15] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and Regression Trees". Belmont, CA: Wadsworth International Group, 1984.

[16] M. A. Carreira-Perpi˜nan and G. E. Hinton, "On contrastive divergence learning," in Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05), (R. G. Cowell and Z. Ghahramani, eds.), pp. 33–40, Society for Artificial Intelligence and Statistics, 2005.

[17] P. Clifford, "Markov random fields in statistics," in Disorder in Physical Systems: A Volume in Honour of John M. Hammersley, (G. Grimmett and D. Welsh, eds.), pp. 19–32, Oxford University Press, 1990.

[18] R. Collobert and S. Bengio, "Links between perceptrons, MLPs and SVMs," in Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04), (C. E. Brodley, ed.), p. 23, New York, NY, USA: ACM, 2004.

[19] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 160–167, ACM, 2008.

[20] J. L. Elman, "Learning and development in neural networks: The importance of starting small," Cognition, vol. 48, pp. 781–799, 1993.

[21] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pretraining," in Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09), pp. 153–160, 2009.

[22] M. A. F. Figueiredo and A. K. Jain, "Unsupervised learning of finite mixture models," IEEE Trans. Pattern Anal. Machine Intell., vol. 24, no. 3, pp. 381–396, Mar 2002.

[23] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological Cybernetics, vol. 36, pp. 193–202, 1980.

[24] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 6, pp. 721–741, November 1984.

[25] B. Hassibi, D. Stork and G. Wolff, "Optimal Brain Surgeon and General Network Pruning", Standford University Press, 1992

[26] Simon Haykin, Neural Networks: a comprehensive foundation, (2nd edition), USA, Prentice Hall, ISBN: 0-13-908385-5, 2000

[27] K. A. Heller and Z. Ghahramani, "A nonparametric bayesian approach to modeling overlapping clusters," in Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07), pp. 187–194, San Juan, Porto Rico: Omnipress, 2007.

[28] G. E. Hinton, "Learning distributed representations of concepts," in Proceedings of the Eighth Annual Conference of the Cognitive Science Society, pp. 1–12, Amherst: Lawrence Erlbaum, Hillsdale, 1986.

[29] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, pp. 1527–1554, 2006.

[30] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, vol. 313, pp. 504–507, 2006.

[31] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations, (D. E. Rumelhart and J. L. McClelland, eds.), pp. 282–317, Cambridge, MA: MIT Press, 1986.

[32] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," in Advances in Neural Information Processing Systems 6 (NIPS'93), (D. Cowan, G. Tesauro, and J. Alspector, eds.), pp. 3–10, Morgan Kaufmann Publishers, Inc., 1994.

[33] G. Karer and I. Skrjanc, "Predictive Approaches to Control of Complex Systems", SCI 454, pp. 49–98, 2003.

[34] S. Kirkpatrick, C. D. G. Jr., and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, pp. 671–680, 1983.

[35] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," Journal of Machine Learning Research, vol. 10, pp. 1–40, 2009.

[36] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09), (L. Bottou and M. Littman, eds.), Montreal (Qc), Canada: ACM, 2009.

[37] S. Lawrence, C. Lee Giles, and Ah Chung, "What Size Neural Network Gives Optimal Generalization", Technical Report, Institute for Advanced Computer Studies, University of Maryland,1996

[38] Y. Le Cunn, J. Denker and S. Solla, "Optimal Brain Damage", AT&T Bell Laboratories, 1990

[39] A. M. Luciano and M. Savastano, "Fuzzy Identification of Systems with Unsupervised Learnig", IEEE Transactions on systems, man and cybernetics, Vol. 27, pp 138-142, Feb 1997

[40] A. Ludovic, P. Helene and S. Michele, "Unsupervised Layer-Wise Model Selection in Deep Neural Networks", H. Coelho et. al., IOS Press, 2010

[41] K. S. Narendra and K. Parthasarathy, "Gradient methods for optimization of dynamical systems containing neural networks", IEEE Transactions od Neural Networks, pp. 252-262, March 1991

[42] A. Rahimi and B. Recht, "Unsupervised Regression with applications to Nonlinear System Identification", Intel Research, Seattle, 2010

[43] D. Stathakis, "How many hidden layers and nodes?", in International Journal of Remote Sensing, Vol. 30, No. 8, 2133-2147, April 2009

[44] S. J. Yakowitz, "Unsupervised Learning and the Identification of Finite Mixtures", IEEE Transactions of Information Theory, pp. 330-338, May 1970