CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACANTENCO
DEPARTAMENTO DE CONTROL AUTOMÁTICO

Identificación de sistemas no lineales con redes neuronales convolucionales.

Tesis que presenta

Mario Antonio López Pacheco

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Control Automático

Director de la Tesis:

Dr. Wen Yu Liu

Ciudad de México Agosto 2017

Agradecimientos

A mis padres Edith Pacheco Ramírez y Mario López Ortega por todo el cariño, amor y apoyo que me han brindado toda mi vida.

A mi hermana, tías y tíos que me han apoyado durante toda mi educación.

A mi director de tesis el Dr. Wen Yu Liu por brindarme su apoyo, tiempo y guía para la realización de este trabajo.

Al Consejo de Ciencia y Tecnología y al Centro de Investigación y Estudios Avanzados, por el apoyo económico para la realización de mis estudios de posgrado.

A todos, Gracias.

Mario Antonio López Pacheco

Resumen

En este trabajo de tesis se propone una estructura de una red neuronal convolucional (CNN) para la identificación de sistemas no lineales, lo cual conlleva ventajas sobre otros tipos de redes neuronales como evitar mínimos locales y efectos del ruido. Por medio de dos paradigmas de aprendizaje, supervisado y no supervisado, se realiza la identificación. Además, se realizan comparaciones entre la CNN y un perceptrón multicapa (MLP), y cambiando la cantidad de elementos en las capas de la CNN. Los métodos propuestos son validados con tres conjuntos de datos de prueba.

Abstract

In this Thesis it is proposed a structure of a convolutional neural network (CNN) for nonlinear system identification, which has advantages over other types of neural networks such as avoiding local minima and noise effects. Through two learning paradigms, supervised and unsupervised learning, identification is made. Also, comparisons are made between CNN and a multilayer perceptron (MLP), changing the amount of elements in the CNN layers. The proposed methods are validated with three benchmark data sets.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

La identificación de sistemas es el proceso de obtener un modelo matemático de un sistema utilizando datos obervados de éste [1]. Los modelos matemáticos con los cuales se realiza este proceso van desde una ecuación lineal hasta modelos más complicados como lo son las redes neuronales. Una de las propiedades importantes de las redes neuronales es su naturaleza adaptativa, lo que quiere decir que obtienen su conocimiento de su entorno mediante un proceso adaptativo llamado aprendizaje [2].

Las redes neuronales convolucionales, son un caso particular y consisten de capas convolucionales y capas de submuestreo alternadas [3] seguidas de dos capas completamente conectadas. Sin embargo, se han enfocado en resolver problemas de clasificación de imágenes aprovechando el operador de convolución dentro de la red y dejando a un lado el problema de identificación.

En este trabajo se presenta el uso de redes neuronales convolucionales para la modelación de sistemas aplicando dos paradigmas de aprendizaje y realizando una comparación con una red multicapa y un comparación entre redes convolucionales cambiando la cantidad de parámetros de cada una.

1.1. Motivación

Existe una gran variedad de tipos de redes neuronales las cuales han sido utilizadas para varios propositos, entre ellos calsificación e identificación de sistemas. Para la tarea de identificación, los resultados que se han obtenido son muy buenos y difieren gracias a las propuestas de muchas arquitecturas de redes neuronales y algoritmos para el aprendizaje. En el caso ideal, los datos que se tienen del sistema no contienen ruido, pero en sistemas reales, existen muchas fuentes de ruido que pueden afectar la parte de identificación y en particular la parte de control.

Las redes neuronales convolucionales se han utilizado para la tarea de clasificación, en la cual se han obtenido muy buenos resultados y se han mejorado con el pasado de los años gracias a los componentes computacionales que de igual manera van mejorándose. Una particularidad de estas redes es la operación de convolución que realiza en las primeras capas, está operación se utiliza como un "filtro", lo cual tiene muchas aplicaciones en el tratamiento de imágenes.

Por la característica de convolución, podemos emplear este tipo de redes neuronales para la identificación, en particular cuando hay ruido en los datos de aprendizaje. En este trabajo se propone una arquitectura de red nerunal convolucional para la identifcación de sistemas.

1.2. Contribuciones

En este trabajo de tesis se trata con el problema de la identificación de sistemas no lineales mediante redes neuronales convolucionales. Esto se logra proponiendo una estructura para la red convolucional, esto es, definir la cantidad de unidades que se encuentran en cada capa de la red y la cantidad de capas que contendrá. La arquitectura propuesta consiste en 6 capas: dos capas convolucionales, dos capas de submuestreo intercaladas con las convolucionales y dos capas completamente conectadas.

Una vez construida la red, empieza la parte de aprendizaje, en la cual se proponen dos metodologías para ello. La primera consiste en utilizar aprendizaje fuera de línea. Para las

capas convolucionales, se escogen los valores iniciales de los parámetros de estas capas y se mantienen durante la fase de aprendizaje, las capas completamente conectadas se entrenan mediante mínimos cuadrados al final de la fase.

La segunda metodología de aprendizaje se realiza mediante aprendizaje en línea, utilizando algoritmos de gradiente descendiente (backpropagation), con los cuales todos los parámetros de la red se actualizan en cada iteración en la fase de entrenamiento. Este algoritmo se modifica para contemplar los parámetros de las capas convolucionales.

Además, se utiliza una red multicapa para comparar los resultados cuando hay ruido en los datos de entrenamiento, lo cual permite ver que la red neuronal convolucional propuesta tiene mejor desempeño y puede emplearse para casos en donde este presente ruido.

1.3. Objetivos

- Proponer una estrucutra de red neuronal convolucional para la identificación de sistemas.
- Diseñar el algoritmo para implementar la red neuronal convolucional propuesta.
- Realizar simulaciones con datos de prueba utilizando aprendizaje no supervisado y comparar los resultados con una red neuronal multicapa.
- Agregar ruido a los datos de aprendizaje y realizar las mismas simulaciones con aprendizaje no supervisado.
- Cambiar la arquitectura propuesta para agregar mas parámetros en la red.
- Realizar simulaciones con aprendizaje en linea con diferentes numeros de parámetros en la red y comparar los resultados.
- Agregar ruido a los datos de aprendizaje y realizar las mismas simulaciones con aprendizaje supervisado.

1.4. Estructura

El trabajo de tesis está estructurado en 5 capítulos. El primer capítulo contiene la introdcción del trabajo realizado. El capítulo 2 consiste en dar un marco teórico del uso de las redes neuronales, una breve historia de la evolución de las redes neuronales convolucionales y sus propiedades características. El capítulo 3 trata sobre la arquitectura de la red que se propone y las ecuaciones matemáticas del modelo, después se muestra el aprendizaje no supervisado y por mínimos cuadrados para entrenar la red y las simulaciones sobre tres ejemplos de prueba. En el capítulo 4 se mencionan métodos de aprendizaje en línea, además de modificación del algoritmo de backpropagation para entrenar la red, de igual manera se presentan las simulaciones realizadas con esta metodología de aprendizaje para los mismos sistemas de prueba pero agregando mas parámetros a la red y realizando comparaciones entre ellas. El último capítulo son las conclusiones generales acerca de la tesis.

Capítulo 2

Preliminares

2.1. Redes neuronales para la modelación de sistemas

Las redes neuronales fueron una abstracción de los sistemas nerviosos biológicos, constituidas por un conjunto de unidades llamadas neuronas conectados unas con otras. El primer modelo fue propuesto por McCulloch y Pitts en 1943 [4], era un modelo binario, donde cada neurona tenía un umbral fijo. Una de las clasificaciones más sencilla de los modelos de redes neuronales es la siguiente:

- Modelos inspirados en la biología.
- Modelos artificiales aplicados.

Los modelos de tipo biológico comprenden las redes que tratan de simular los sistemas neuronales, además de ciertas funciones como la auditiva o visual. En el cerebro humano se calcula que existen alrededor de cien millones de neuronas y cada una de ellas con alrededor de mil sinapsis.

Las neuronas y las conexiones entre ellas constituyen la clave para el procesamiento de información. Una neurona está formada de tres partes Figura 2.1:

- Cuerpo de la neurona.

- Dendritas.
- Axón.

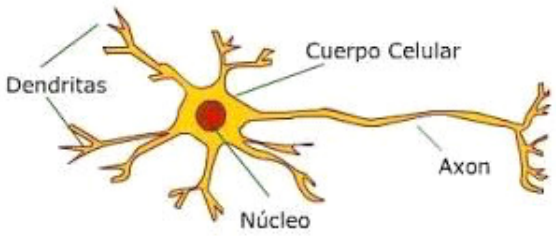


Figura 2.1: Neurona Biológica.

La forma en que dos neuronas se relacionan no es conocida en su totalidad y depende del tipo particular de neurona [5]. En general, la neurona manda su información a otras neuronas a través de su axón y se transmite por diferencias de potencial. Este proceso se modela como una regla de propagación $u(\cdot)$. La neurona recoge las señales por su sinapsis sumando todas las influencias positivas o negativas. Si las influencias positivas dominan, la neurona manda una señal positiva a otras neuronas, como si fuera una función escalón.

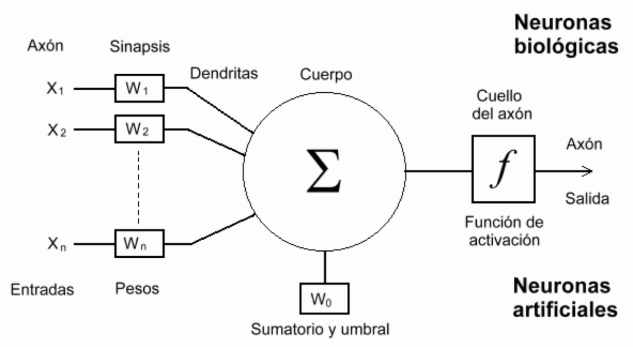


Figura 2.2: Neurona artificial basada en la biología.

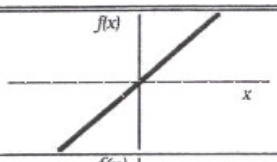
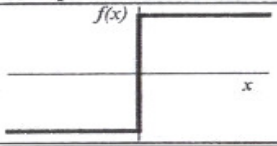
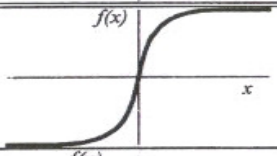
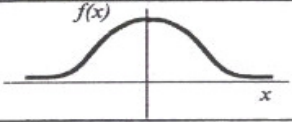
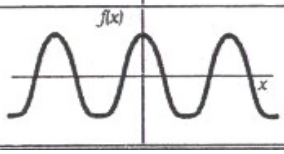
En la Figura 2.2 se muestra una neurona artificial donde sus elementos se comparan con las neuronas biológicas. Se tiene un conjunto de n entradas x con sus correspondientes pesos

sinápticos w. Una función de sumatoria Σ, una función de activación f y una salida y. La neurona puede adaptarse a su entorno modificando el valor de los pesos sinápticos. En este modelo la salida está dada por:

$$y = f(\sum_{i=1}^{n} w_i x_i) = f(u(x)) \tag{2.1}$$

La función de activación f [6] se elige de acuerdo a la tarea a realizar. Las más comunes se muestran en la Tabla 1.

Tabla 2.1: Funciones de activación.

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	{-1, +1} {0, +1}	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	[0, +1] [-1, +1]	
Gaussiana	$y = Ae^{-Bx^2}$	[0,+1]	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	[-1,+1]	

La otra categoría de redes neuronales son las artificiales aplicadas [7]. Este tipo de red siguen basándose en las redes neuronales biológicas, aunque poseen otras funcionalidades y estructuras de conexión. Algunas de las características de estas redes son [8]:

- **Auto-organización y adaptabilidad.** Emplean algoritmos de aprendizaje adaptativos, por lo que ofrecen mejores posibilidades de procesado robusto.
- **Procesado no lineal.** Aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumentar su resistencia frente al ruido.
- **Procesamiento en paralelo.** Se utilizan una gran cantidad de nodos con un alto nivel de interconectividad.

2.1.1. Redes Neuronales Multicapa

Un caso particular de las redes neuronales es cuando las neuronas se organizan por capas [9], ver Figura 2.3. Tenemos la capa de entrada la cual se constituye por las entradas a la red, la capa de salida la cual contiene las neuronas que forman la salida de la red, y las capas ocultas que se encuentran entre estas dos últimas capas mencionadas. Si la salida de una neurona va hacia dos neuronas diferentes, ambas reciben el valor completo de la salida.

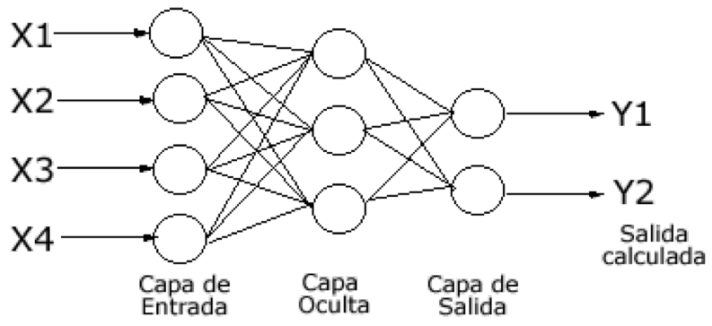


Figura 2.3: Red Neuronal Multicapa.

El problema con este tipo de red es proporcionar los pesos adecuados para tener una aproximación correcta de la salida, ya que solo se reciben en su mayoría los datos de entrada y salida, y no habría forma de obtener variaciones certeras en las capas ocultas. Se considera

w_{ji} como el peso sináptico entre la neurona de entrada i y la neurona oculta j. La entrada que recibe una neurona de una capa oculta está dada por:

$$u(x_i) = \sum_{i=1}^{N} w_{ji} x_i + \theta_j \tag{2.2}$$

donde θ_j es el umbral o sesgo de la neurona. La señal de salida de esta neurona está dada por [10]:

$$y_j = f(u(x_i)) \tag{2.3}$$

Hay dos etapas para utilizar una red neuronal: Aprendizaje y Generalización. En la etapa de aprendizaje [11], se utiliza un conjunto de datos o patrones de entrenamiento para calcular los pesos sinápticos de la red, iniciados en algún valor aleatorio, los cuales se van modificando de manera iterativa utilizando los datos de prueba, de manera que, una función de costo elegida se minimice, en su mayoría se escoge el error entre la salida de la red y la salida del sistema. En algunos casos la red no aprende, esto debido a que los datos de entrada no fueron escogidos de manera que representen el comportamiento deseado del sistema. También se da el caso en donde los datos sean insuficientes para lograr que los valores de los pesos converjan. La función de costo es importante en la etapa de entrenamiento. Los algoritmos de aprendizaje buscan entonces en el espacio de solución la que minimice el costo. Por ejemplo, considere que la función de costo a minimizar sea

$$C = E\left[(y - y_d)^2\right] \tag{2.4}$$

donde y es la salida de la red y y_d es la salida deseada. Frecuentemente se llama estática ya que solamente se pueden hacer aproximaciones. En situaciones practicas, podemos considerar solamente muestras, por tanto, la ecuación anterior puede reescribirse como:

$$\hat{C} = \frac{1}{N} \sum_{i=1}^{N} (y_i - y_{di})^2 \tag{2.5}$$

La función de costo finalmente dependerá de la tarea a realizar.

Paradigmas de aprendizaje

Existen tres paradigmas de aprendizaje populares, cada uno correspondientes a una tarea de aprendizaje en particular. El primero de ellos es aprendizaje supervisado [12]. En el tenemos un conjunto de datos de entrada y salida del sistema, y lo que se quiere es encontrar los valores de los pesos de la red a partir de ellos. La función de costo utilizada comúnmente es el error medio cuadrático, el cual intenta minimizar el error promedio cuadrático entre la salida de la red y la salida del sistema. Algunas tareas que entran dentro de este paradigma es el reconocimiento de patrones y la regresión. Puede verse como un aprendizaje donde tenemos un maestro que provee realimentación continua.

El segundo paradigma corresponde al aprendizaje no supervisado [12]. En este caso algunos datos de entrada se tienen al igual que la función de costo a minimizar. Está función varía dependiendo de la tarea a realizar. Algunas tareas que se encuentran dentro de este paradigma son problemas de estimación como la estimación de distribuciones estadísticas y filtraje.

El tercer paradigma corresponde al aprendizaje por refuerzo. Generalmente no existe datos de entrada y estos se generan con agentes que interaccionan con el entorno. En cada instante, el agente realiza una acción y el entorno produce una señal y costo instantáneo. El objetivo es encontrar alguna política que minimice el costo a largo plazo. El entorno es desconocido y puede modelarse como un proceso de decisión de Markov. Las redes neuronales usan aprendizaje por refuerzo como parte de un algoritmo general. La programación dinámica acompaña a las redes neuronales por Bertsekas y Tsitsiklis [13] y aplicados a problemas no lineales multi-dimensionales por la capacidad de las redes neuronales de reducir la perdida de precisión.

2.1.2. Algoritmo Backpropagation

En la etapa de aprendizaje, el objetivo es minimizar la función de costo [14]. La función de costo a minimizar es:

$$E^p = \frac{1}{2}\sum_{k=1}^{M}(y_{pk}^p - y_k^p)^2 \tag{2.6}$$

Como E^p está en función de los pesos de la red, el gradiente es un vector igual a la derivada de E respecto a cada uno de los pesos. El gradiente toma la dirección en la cual el error crece más rápido, y la dirección opuesta indica el decremento más rápido del error. Con esto el error puede reducirse ajustando cada peso en la dirección

$$-\sum_{p=1}^{P} \frac{\partial E^p}{\partial w_{ji}} \tag{2.7}$$

Donde P es número de datos en el conjunto de prueba. A un nivel práctico la forma de modificar los valores de los pesos de forma iterativa consiste en aplicar la regla de la cadena a la expresión del gradiente y añadir una tasa de aprendizaje η. De esta manera, el cambio en los pesos sinápticos en la última capa es:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = \eta \sum_{p=1}^{P} \delta_k^p y_j \tag{2.8}$$

donde

$$\delta_k^p = (y_{dk}^p - y_k^p) f'(u_k^p) \tag{2.9}$$

En el resto de las neuronas de las otras capas, se tiene:

$$\Delta w_{ji} = \eta \sum_{p=1}^{P} \delta_j^p x_i^p \tag{2.10}$$

donde

$$\delta_j^p = f(u_j^p) \sum_{k=1}^{M} \delta_k^p w_{kj} \tag{2.11}$$

Se puede notar que el error en las capas ocultas de la red viene determinado por la suma de los errores que se generan en las neuronas de la capa de salida que reciben como entrada la salida de esa neurona en la capa j. De aquí que el algoritmo se denomine propagación del error hacia atrás (Backporpagation). La actualización de los pesos se realiza una vez que se han utilizado los patrones de entrenamiento [15].

2.1.3. Modelación de sistemas no lineales

La teoría de sistemas de control se ocupa del análisis y diseño de componentes interactuantes de un sistema en una configuración que brinde un comportamiento deseado. Un modelo es una representación simplificada de un sistema. La mayoría de los modelos matemáticos utilizados son lineales, debido a su facilidad para ser manipulados comparados con los sistemas no lineales, y pueden representar en forma precisa el comportamiento de sistemas reales en muchos casos útiles [16]. Con el avance de la tecnología se han generado una gran cantidad de problemas que en su mayoría son no lineales.

Los sistemas no lineales pueden ser representados de la siguiente manera:

$$\begin{aligned} \dot{x} &= f(x, u, t) \\ y &= h(x, u, t,) \end{aligned} \tag{2.12}$$

en donde x es el vector de estados, u es el vector de entradas, y es el vector de salidas, t es el tiempo, f es un *conjunto de n funciones escalares* y h es un *conjunto de q funciones escalares.*

Las no linealidades pueden ser inherentes o intencionales. Las inherentes so propias del sistema como un convertidor dc/dc, y las intencionales son adicionales por diseño, por ejemplo, la histéresis. Los sistemas no lineales presentan las siguientes características [17]:

- **Tiempo de escape finito.** Una variable de estado de un sistema lineal inestable se va a infinito cuando el tiempo tiende a infinito, en cambio, en un sistema no lineal inestable puede hacerlo en tiempo finito.

- **Múltiples puntos de operación.** Pueden ser estables o inestables. Los estados del sistema convergen a uno u otro dependiendo del estado inicial del sistema. Comparado con los sistemas lineales que solo presentan un punto de equilibrio en el origen.

- **Ciclos limite.** Sistemas no lineales pueden oscilar con amplitud y frecuencia constante independientemente de la condición inicial

- **Subarmónicos, armónicos u oscilaciones casi-periódicas.** Un sistema no lineal puede generar frecuencias que son submúltiplos o múltiplos de la frecuencia de entrada cuando se aplica una entrada sinusoidal.
- **Múltiples modos de comportamiento.** Pueden existir más de un ciclo limite. Dependiendo de la entrada (amplitud y frecuencia) la salida puede exhibir armónicos, subarmónicos.
- **Caos.** Se produce en sistemas en que la salida es extremadamente sensible a las condiciones iniciales. La salida no está en equilibrio y no es oscilación periódica o casi periódica.

El punto de inicio en el análisis de un sistema de control es su representación por un modelo matemático, el cual usualmente es un operador entre entradas y salidas del sistema o como un conjunto de ecuaciones diferenciales [17]. Un modelo es útil para realizar simulaciones ya que representa la dinámica del sistema y para el diseño de controladores basados en modelo. El principal problema es la obtención del modelo.

La identificación de sistemas es una herramienta que permite representar el comportamiento real de sistemas con base en datos experimentales obtenidos del sistema. Esto es un proceso iterativo, y el éxito de la identificación depende de la calidad de las señales de entrada y de la identificabilidad del sistema.

2.2. Redes Neuronales Convolucionales

2.2.1. Convolución

Una convolución es un operador matemático que trabaja con dos argumentos (funciones), f y g, y las transforma en una tercera. La convolución en tiempo continuo se define como la integral del producto de dos funciones después de desplazar una de ellas una distancia t [18]. Sean las funciones f y g. Su convolución está denotada como:

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.13}$$

El intervalo de integración depende del dominio sobre el cual están definidas las funciones. Para el caso discreto se tiene que la intgral se intercambia por la sumatoria, dando como resultado:

$$f * g = \sum_{-\infty}^{\infty} f(k)g(n-k) \tag{2.14}$$

Donde una de las funciones está desplazada una distancia n.
Las propiedades de los diferentes operadores de convolución son:

- **Conmutatividad**

$$f * g = g * f \tag{2.15}$$

- **Asociatividad**

$$f * (g * h) = (f * g) * h \tag{2.16}$$

- **Distributividad**

$$f * (g + h) = f * g + f * h \tag{2.17}$$

- **Asociatividad multiplicado por un escalar**

$$a(f * g) = (af) * g = f * (ag) \tag{2.18}$$

- **Regla de la derivación**

$$D(f * g) = Df * g + f * Dg \tag{2.19}$$

  Donde Df denota la derivada de f, o en el caso discreto el operador de diferencia.

- **Teorema de convolución**

$$\mathcal{F}(f * g) = (\mathcal{F}(f)) \cdot (\mathcal{F}(g)) \tag{2.20}$$

  Donde $\mathcal{F}$ denota la Transformada de Fourier de f. Este teorema aplica tambien para la Transformda de Laplace.

- **Convolución con delta de Dirac**

$$\begin{aligned} f(t) * \delta(t) &= f(t) \\ f(t) * \delta(t - t_0) &= f(t - t_0) \\ f(t - t_1) * \delta(t - t_0) &= f(t - t_0 - t_1) \end{aligned} \tag{2.21}$$

Por lo general es de ayuda ver al operador de convolución como un producto matricial. La manera sencilla de verlo es como una función de tipo ventana deslizante aplicada a una matriz.

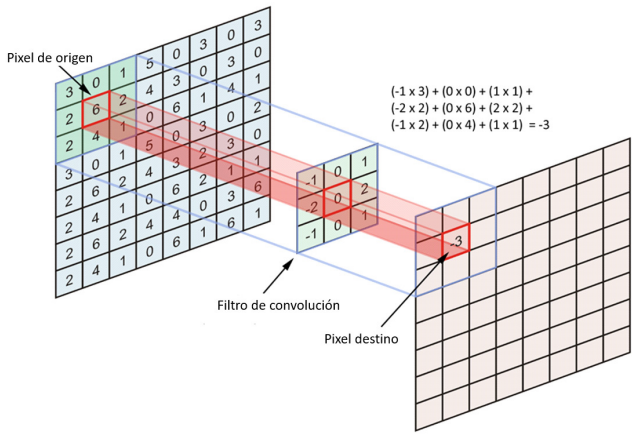


Figura 2.4: Aplicación de una convolución con matrices.

En la Figura 2.4 se observa un ejemplo del operador de convolución aplicado sobre una matriz en forma de un filtro. El resultado se obtiene multiplicando cada elemento del filtro de convolución con su correspondiente en la matriz de entrada para posteriormente sumarlos y así obtener el resultado del elemento de salida. Cabe mencionar que el tamaño del filtro toma importancia cuando se opera en las orillas de las matrices, ya que dependiendo de la distancia vertical u horizontal del centro del filtro a sus extremos es la cantidad de filas y columnas que desaparecerán en la matriz de salida [19]. Para evitar este problema se puede utilizar el rellenado con ceros, el cual consiste en extender los espacios numéricos añadiendo

valores nulos (ceros) en los extremos de la matriz de entrada antes de realizar la convolución [13].

En la Figura 2.5 se puede observar que los valores del vector de entrada (vector de la derecha) se multiplica con el operador de convolución (una fila de la matriz) y se suman entre sí para obtener el valor de salida (un elemento del vector del lado izquierdo).

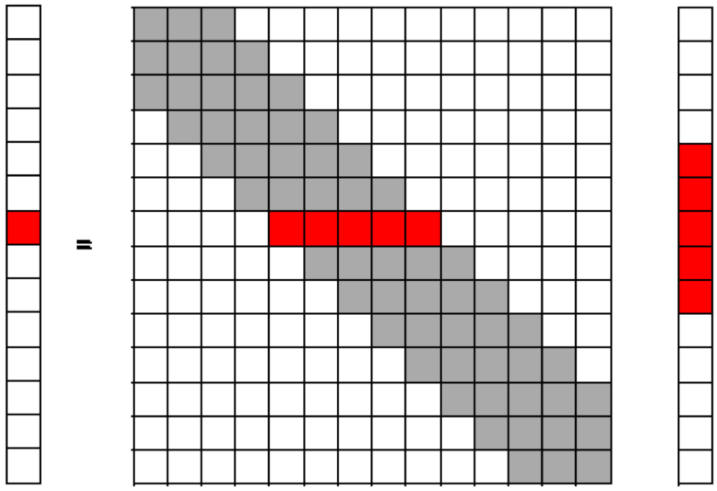


Figura 2.5: Convolución de una dimensión en notación vector-matriz.

En el procesamiento de imágenes, donde la convolución es comúnmente utilizada, se utiliza como un filtro para cambiar características en las imágenes, remarcar contornos, difuminar o reducir el ruido de alta o baja frecuencia. En el procesamiento de señales es utilizado suprimir porciones no deseadas de la señal o para separar la señal en partes.

2.2.2. Redes Neuronales Convolucionales

Presentadas por LeCun *et al* [20] a principios de la década de los noventa, las redes convolucionales son un ejemplo de una arquitectura de red neuronal especializada, la cual incluye conocimiento sobre la invariancia de formas bidimensionales utilizando patrones de conexión local y con restricciones en los pesos [21].

Las redes neuronales convolucionales presentan una arquitectura multicapa, donde cada capa está constituida por un número determinado de convoluciones con funciones de activación no lineal, ya sea $ReLU$ o $tanh$, para obtener los resultados [22]. El tipo de conexiones de las neuronas en la red convolucional está inspirado en la organización de la corteza visual de los animales, en la cual la respuesta de cada neurona puede representarse matemáticamente como una operación de convolución. Las neuronas de la corteza visual responden individualmente a estímulos externos en una cierta región espacial especifica llamada campo receptivo. Cada campo receptivo se sobrepone con sus adyacentes de tal manera que se forma una retícula del campo de visión [23]. Una característica importante en este tipo de redes es que son invariantes con respecto a la posición espacial. Considerando la corteza visual, tenemos dos tipos de células [24]: las células simples que responden al máximo a patrones específicos dentro de su campo receptivo. Y las células complejas que presentan campos receptivos de mayor tamaño y son localmente invariantes a la posición exacta de patrones en su campo de visión.

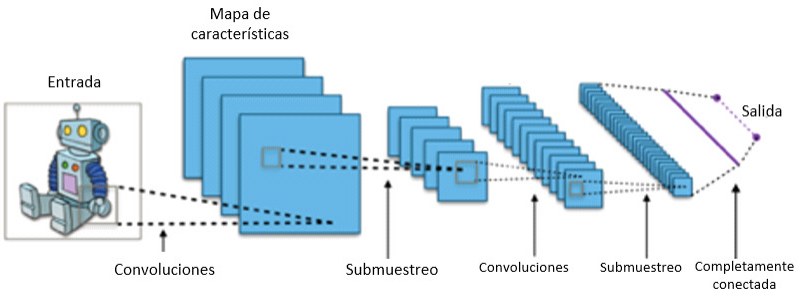


Figura 2.6: Arquitectura de un Red Neuronal Convolucional.

La arquitectura o estructura general de una red convolucional se muestra en la Figura 2.6 [25]. Algunas de las características más importantes de las redes convolucionales se presentan a continuación como son la conectividad dispersa y pesos compartidos.

Conectividad dispersa

Las redes neuronales convolucionales aprovechan las correlaciones espacialmente locales

reforzando los patrones de conectividad local entre neuronas de capas adyacentes [26]. Refiriéndose a la Figura 2.7. para una neurona o unidad de la capa $m+1$ sus entradas son un subconjunto de las neuronas de la capa m, en este caso una neurona de la capa $m+1$ tiene un campo receptivo de 3. Cada neurona del subconjunto correspondiente en la capa m tiene a su vez un campo receptivo de 3, los cuales son adyacentes.

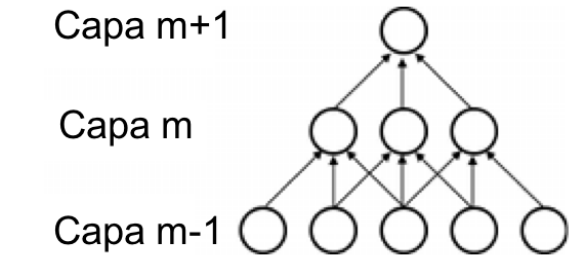


Figura 2.7: Conectividad dispersa.

Dado que cada neurona tiene un tamaño de campo receptivo fijo, no toman en cuenta las variaciones fuera de este, produciendo un tipo de *filtro* con su campo receptivo.

Pesos compartidos

La arquitectura de las redes neuronales convolucionales asegura que los filtros produzcan la respuesta más fuerte a un patrón de entrada localmente espacial [27]. En este tipo de red neuronal, cada filtro h_i se utiliza a lo largo de todo el campo visual de la capa. Estas unidades comparten la misma parametrización, es decir pesos y sesgos, formando un mapa de características.

En la Figura 2.8 se muestran en la capa m tres unidades que pertenecen al mismo mapa de características. Las líneas del mismo color indican que el mismo peso sináptico es utilizado. Utilizar estas unidades a lo largo del campo de visión permite identificar características sin importar su posición en él. Además, que el empleo de pesos compartidos mejora la eficiencia de aprendizaje reduciendo el número de parámetros libres para ser aprendidos [26]. También reduce la brecha entre el error en la etapa de entrenamiento y en la etapa de generalización [28] y la cantidad de memoria requerida para hacer funcionar la red neuronal. Una capa

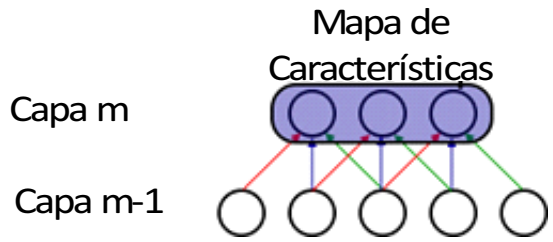


Figura 2.8: Mapa de características.

Convolucional completa está constituida por una gran cantidad de mapas de características, los cuales detectan diferentes características de manera local a lo largo de todo el campo de visión.

2.2.3. Tipos de Redes Neuronales Convolucionales

En los apartados siguientes se muestra una breve descripción de la evolución de las redes neuronales partiendo de los estudios de la corteza visual realizados por Hubel y Wiesel [29]. Se presenta una recapitulación de redes convolucionales existentes [30].

Neocognitrón

Fue introducido en 1980 [31]. Tiene una estructura multicapa y tiene un aprendizaje no supervisado, donde únicamente es necesario un conjunto de datos de entrada en la capa de entrada. El neocognitrón consiste de una conexión en cascada de un número de estructuras modulares precedidas por una capa de entrada. Cada una de estas estructuras modulares está compuesta de dos capas de células conectadas en cascada. La primera capa contiene células simples de acuerdo a la clasificación de Hubel y Wiesel. La segunda capa contiene células complejas. Se asume que cada célula de la misma capa tiene sinapsis de entrada de la misma distribución espacial, es decir tienen el mismo campo receptivo, pero en diferente posición. En la Figura 2.9 se muestra las interconexiones entre las capas de la red. Cada célula recibe conexiones de la capa anterior que se encuentren dentro de la elipse formada en esa capa. Dada la estructura de la red, mientras más profunda sea la capa, mayor será el

campo receptivo de las células en esa capa con respecto a la capa de entrada.

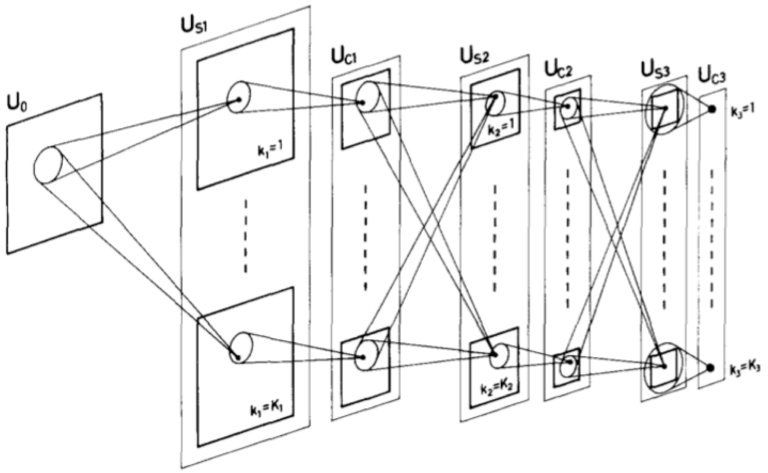


Figura 2.9: Diagrama esquemático Neocognitrón.

Para el autoaprendizaje, es necesario que las células simples en la misma capa tengan sinapsis de entrada con la misma distribución espacial, entonces la sinapsis se refuerza de la siguiente manera. Se escogen varias células simples representativas cada vez que llega un patrón de entrada, estas células son las que presentan las salidas más grandes. La sinapsis de entrada se refuerza en la misma manera que con el cognitron- r.m.s [32]. Las células no seleccionadas de esa capa no se refuerzan durante este proceso.

LeNet-5

Es una red neuronal convolucional de nivel 7 introducida por Yann LeCun *et al* [21] para clasificación de dígitos. Contiene 7 capas sin contar la de entrada y todas ellas tienen parámetros para entrenar. Como se observa en la Figura 2.10, la entrada consta de una imagen de 32x32 pixeles. El conjunto de centros de los campos receptivos de la última capa convolucional, C_3, forman un área de 20x20 en el centro de la entrada de 32x32. Los valores de los pixeles están normalizados. Las capas de convolución son etiquetadas como Cx, las capas de submuestreo como Sx, las capas completamente conectadas como Fx, donde x representa el índice de la capa.

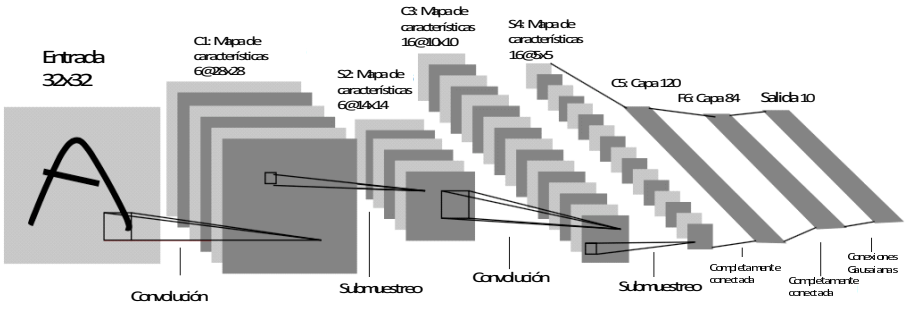


Figura 2.10: Arquitectura LeNet-5.

Como en otras redes neuronales clásicas, unidades hasta la capa F_6 realizan un producto punto entre su vector de entrada y el vector de pesos, a lo que se agrega un sesgo. La suma ponderada denotada a_i para la unidad i pasa a través de la función *tanh* para producir el estado x_i

$$x_i = f(a_i) \tag{2.22}$$

donde

$$f(a) = A \ \tanh(Sa) \tag{2.23}$$

siendo A la amplitud de la función y S la pendiente en el origen. Finalmente, la capa de salida esta conformada por Funciones Radiales Básicas (Radial Basic Functions, RBF por sus siglas en inglés), una de cada clase con 84 entradas cada una. La salida de cada unidad RBF se calcula como sigue:

$$y_i = \sum_j (x_j - w_{ji})^2 \tag{2.24}$$

La función de pérdida más simple que puede ser utilizada con esta red es el criterio de estimación de la máxima verosimilitud (Maximum Likelihood Estimation, MLE por sus siglas en inglés) que en este caso coincide con el MSE (Error Medio Cuadrático por sus siglas del inglés). Para el cálculo del gradiente de la función de pérdida con respecto a los pesos en todas las capas de la red convolucional se realiza con el algoritmo de backpropagation, el cual

debe ser modificado para considerar los pesos compartidos. Una forma sencilla de lograrlo es calcular las derivas parciales de la función de pérdida con respecto a cada conexión y posteriormente sumar los pesos que comparten los mismos parámetros.

Redes Neuronales invariantes por desplazamiento

Fue propuesta para el reconocimiento de caracteres en imágenes en 1988 [33]. El modelo propuesto fue una red multicapa con unidades tipo neurona con funciones de activación sigmoidales. Las unidades tienen sinapsis de entradas modificables durante el proceso de aprendizaje. La estructura se muestra en la Figura 2.11. Unidades en las capas se dividen en grupos. Con excepción de la capa de salida, cada unidad de la capa actual se conecta con al menos una unidad en cada grupo de la capa anterior a ella. Las unidades de la última capa se conectan únicamente con un grupo de la capa anterior [34].

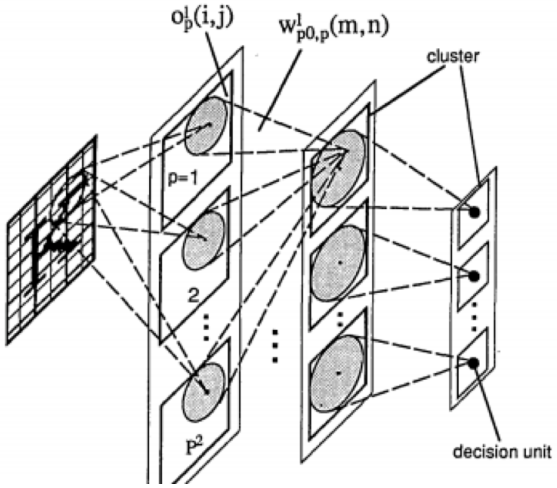


Figura 2.11: Arquitectura Red Neuronal invariante por desplazamiento.

Redes Convolucionales multi-escala

Sermanet *et al* [35] modificaron la red convolucional tradicional alimentando la primera y segunda etapa al clasificador como se muestra en a Figura 2.12. Cada etapa está compuesta de una capa convolucional, una transformación no lineal, y una capa de submuestro.

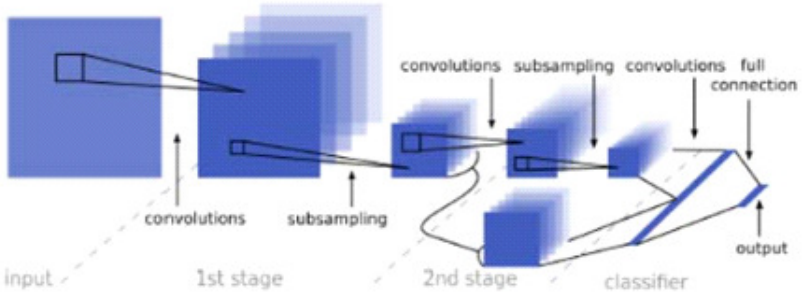


Figura 2.12: Arquitectura de dos etapas de una red convolucional multi-escala.

Alimentar la salida con todas las etapas, permite al clasificador usar solamente las características de alto nivel, lo que hace tener a lo global e invariante, pero con poca precisión. Una de sus aplicaciones fue en la clasificación de las señales de tráfico, en las cuales los resultados experimentales de la competencia GTSRB (German Traffic Sign Recognition Benchmark) produjeron un nuevo record de 99.17 % comparado con el desempeño humano de 98.81 %.

Visualización de Redes Neuronales Convolucionales

Las redes convolucionales de gran escala han demostrado un muy buen desempeño en tareas de clasificación. No se sabe la razón por la cual lo hacen, ni cómo se pueden mejorar. El desarrollo de nuevos modelos se reduce a prueba y error.

Zeiler [36] introdujo una técnica de visualización que permite observar el funcionamiento de las capas intermedias. Esta técnica utiliza una red multicapa deconvolucional (DeConvNet) para proyectar las activaciones de características de regreso al espacio de pixeles. DeConvNet no se ocupan para aprendizaje, solamente como prueba de una red convolucional ya entrenada.

Implementación con GPU

Muchas publicaciones han descrito una mayor eficiencia entrenando redes neuronales convolucionales usando la GPU (Graphic Processing Units) [37, 38, 39, 40]. Esto se debió a la popularización de las GPUs, las cuales se encuentran disponibles en la mayoría de las computadoras actuales. En [41] se propuso una red neuronal de dos capas completamente conectadas, con lo que se consiguió aumentar la velocidad tres veces en las etapas de aprendizaje y de generalización con respecto a un CPU P4 a 3GHz. En 2011 se refinó este proceso obteniendo resultados sorprendentes. En 2012, Ciresan mejoró significativamente el desempeño para múltiples bases de datos de imágenes, incluyendo MNIST databse, NORB database, HWDB10 dataset, y el ImageNet dataset.

Capítulo 3

CNN para el modelado de sistemas: no supervisado

La arquitectura de una red neuronal convolucional (CNN) está formada por el apilamiento de distintas capas las cuales se describen a continuación.

Se realizan pruebas con tres diferentes ejemplos de referencia, comparando la estructura propuesta de la CNN con una red neuronal multicapa. Para cada uno de ellos, se prueban tres casos de simulación. El primero caso consiste en utilizar los datos de prueba de cada referencia sin ruido. El segundo caso utilizando un ruido generado por la función *wgn* en el intervalo $[-0{,}1, 0{,}1]$. El último caso utilizando un ruido aleatorio.

3.1. Arquitectura de una Red Neuronal Convolucional

La estructura propuesta de la red neuronal convolucional para la modelación de sistemas no lineales se presenta en la Figura 3.1. Consiste en total de 5 capas. La secuencia de estás es una capa convolucional, seguida de una capa de submuestreo y de otra capa convolucional. Las últimas dos capas corrresponden a capas completamente conectadas.

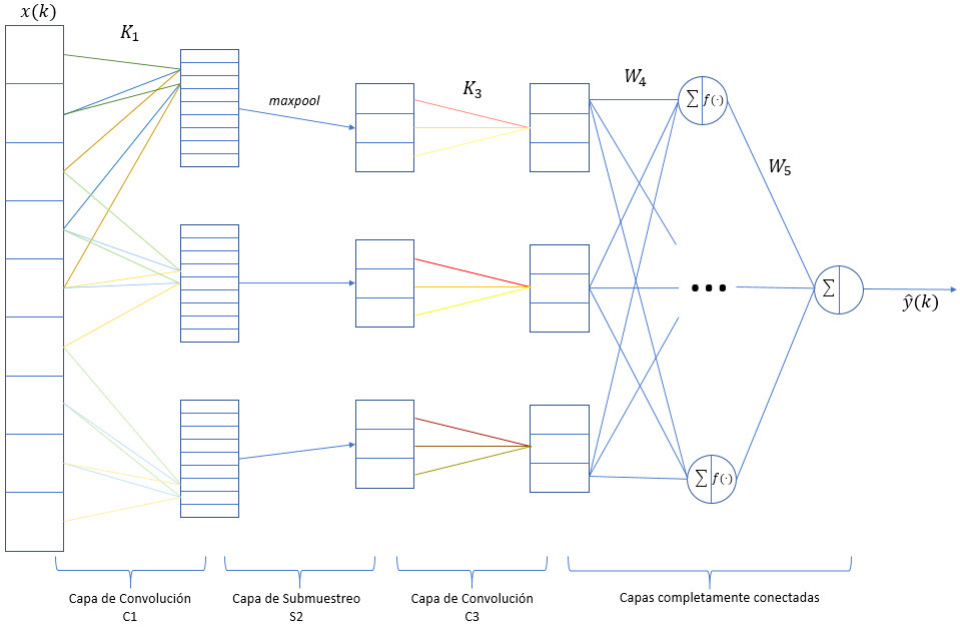


Figura 3.1: Estructura de la Red Neuronal Convolucional para la modelación de sistemas.

Capa Convolucional

Esta capa es el núcleo de las redes neuronales convolucionales. Los parámetros de esta capa consisten en un conjunto de filtros de aprendizaje, los cuales tiene un campo receptivo muy pequeño. En el paso hacia delante de aprendizaje, cada filtro se convoluciona con todo el campo de visión produciendo un mapa de características. Cada elemento del mapa de características puede interpretarse como la salida de una neurona que extrae información de una pequeña región de la entrada y comparte parámetros con neuronas que se encuentran en el mismo mapa. En general, el mapa de características de una capa está dado por [33]:

$$y_j^{(l)} = f\left(\sum_{i \in M_j} y_i^{(l-1)} k_{ji}^{(l-1)} + b_j\right) \tag{3.1}$$

donde M_j representa el conjunto de entradas seleccionadas y el superíndice l representa la capa actual. Cada mapeo de salida tiene un sesgo diferente. La convolución proporciona

facilidad para trabajar con entradas de tamaño variable.

Convolucón transpuesta

La convolución transpuesta puede verse como el paso hacia atrás correspondiente a la convolución, también conocida como deconvolución [42].

Capa de submuestreo

Una parte importante de las redes neuronales son las capas de submuestreo, que se aplican después de las capas convolucionales. La función principal de estas capas es reducir el tamaño de la entrada, aunque realizar esta reducción de tamaño conlleva perdidas de información, pero también trae beneficios a la red. En capas siguientes se reducirá la carga de cálculo, además de reducir el sobreajuste de la red.

Se dice que una red está sobre ajustada cuando tiene un desempeño muy bueno en la etapa de entrenamiento, pero en la etapa de generalización no, debido a que el modelo solo memoriza los datos de prueba y no puede generalizar las reglas para predecir datos futuros [43]. La operación más común para realizar esta reducción es *max-pooling* [44], que divide la entrada (imagen) en rectángulos, y de cada uno de estos rectángulos conserva el elemento de mayor valor. En la Figura 3.2 se muestra la operación de *max-pooling* reduciendo de una matriz de 4x4 a una matriz de 2x2 con los valores máximos en cada subregión creada en la matriz de entrada.

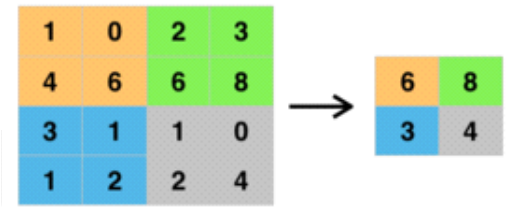


Figura 3.2: Operación max-pooling.

Lo más común es utilizar filtros de dimensión 2x2 con un paso de submuestreo de 2

unidades . La salida de esta capa tiene la forma general:

$$y_j^l = f(\beta_j \text{down}(y_j^{(l-1)}) + b_j) \tag{3.2}$$

donde $down(\cdot)$ representa una función de submuestreo. Algunas otras técnicas para realizarse en la capa de submuestreo se muestran a continuación:

- **Agrupamiento** L_p. Está inspirado en el proceso de agrupamiento biológico en las células complejas [43]. Teóricamente el agrupamiento L_p da una mejor generalización que el max-pooling. Puede representarse como:

$$y_{i,j,k} = \left[\sum_{(m,n)\in\mathcal{R}_{i,j}} (a_{m,n,k})^p \right]^{\frac{1}{p}} \tag{3.3}$$

  donde $y_{i,jk}$ es la salida del operador en la posición (i, j) en el k-ésimo mapa de características y $a_{m,n,k}$ es el valor de la característica en la posición (m, n) dentro de la región de agrupación $\mathcal{R}_{i,j}$ del k-ésimo mapa de características.

- **Agrupamiento estocástico.** Es un método inspirado en deserción. En lugar de elegir el máximo valor dentro de cada región de agrupamiento, el agrupamiento estocástico elige al azar la activación de acuerdo a una distribución multinomial, la cual asegura que las activaciones no maximales del mapa de características sean posibles de utilizar.

- **Agrupamiento espacial piramidal.** Fue introducida por He *et al* [45]. La ventaja de este agrupamiento es que puede generar una representación de longitud fija sin importar el tamaño de la entrada. Esta difiere de los agrupamientos por ventana deslizante en anteriores redes profundas, donde el número de ventanas depende del tamaño de la entrada. Puede sustituirse la última capa de agrupamiento por una de este tipo, lo cual hará posible trabajar con imágenes de diferentes tamaños.

- **Agrupamiento multi-escala sin orden.** [46] Es utilizado para mejorar la invariancia de las redes neuronales convolucionales sin perder su habilidad discriminativa. Se

extraen características profundas de activación para la imagen total y para los parches locales a diferentes escalas. La activación de la imagen completa se realiza como en otras redes convolucionales. La activación de parches locales se realiza por codificación de VLAD.

La combinación entre capas convolucionales y de submuestreo, está inspirado en el trabajo de Hubel y Wiesel sobre la noción de dos tipos de células en la corteza visual [47].

Capa completamente conectada

Al final de las combinaciones de las capas convolucionales y de submuestreo, se utilizan capas completamente conectadas donde cada elemento corresponde a una neurona y el número total de neuronas en estas capas corresponderá al número de clases que se desean predecir. La última capa por lo general es utilizada para tareas de clasificación. El operador softmax es el más usado. Otro método utilizado es el SVM, que combinado con las características de la red neuronal convolucional se pueden resolver diferentes tareas de clasificación [41].

Funciones de activación

Algunas de las funciones de activación se describen enseguida.

Sigmoide

Definida como:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

Actualmente esta función ya no es muy utilizada, debido a los problemas que causa como son la saturación de la salida y la desaparición de los gradientes en la retro-propagación, provocando un mal entrenamiento. Otra razón es que no está centrada en cero produciendo una dinámica de zigzag en el aprendizaje.

Tangente hiperbólica

Definida como:

$$f(x) = \tanh x \tag{3.5}$$

Esta función da resultados mejores que la función sigmoide porque está centrada en cero. Aunque sigue presentando los mismos problemas que la función sigmoide, como es la saturación de la salida.

ReLU

Unidad rectificada lineal (Rectified Linear Unit), se define como:

$$f(x) = \max(0, x) \tag{3.6}$$

Es muy utilizada en redes profundas ya que acelera el aprendizaje por un factor de 6 [41] comparado con las funciones anteriores y evita el problema del desvanecimiento del gradiente cuando hay muchas capas. La parte negativa de esta función, es que durante el proceso de aprendizaje muchas neuronas pueden dejar de funcionar. Esto se da cuando el gradiente mueve los pesos de tal manera que la neurona no se activa, entonces el gradiente será siempre cero para esa neurona y no se activará. El problema anterior puede evitarse seleccionando una tasa de aprendizaje adecuada o haciendo pequeñas variaciones en la función de activación:

- **Función softplus.** $f(x) = \ln(1 + e^x)$, versión lineal de ReLU.
- **Leaky ReLU's.** $f(x) = \begin{cases} x & \text{si } x > 0 \\ 0{,}01x & \text{otro} \end{cases}$ permite tener un gradiente pequeño cuando no está activada la neurona.
- **Noisy ReLU's.** $f(x) = \max(0, x + N(0, \sigma(x)))$, usada en máquinas de Boltzman restringuidas.
- **Neurona Maxout**

Se define como:

$$f(x) = \max(w_1^T x + b_1 w_2^T x + b_2) \tag{3.7}$$

Es una generalización de ReLU y de Leaky ReLU [48]. Tiene las ventajas de la función ReLU pero sin el problema de la muerte de neuronas. Sin embargo, puede llegar a aumentar los parámetros de aprendizaje al doble.

- **ELU**

Clevert *et al* [49] introdujo la unidad exponencial lineal (ELU) la cual permite un aprendizaje más rápido y mejora la precisión de la clasificación. Al igual que las modificaciones de la ReLU, evita el desvanecimiento del gradiente poniendo la parte positiva a 1. La función ELU está definida como:

$$f(x) = \max(x, 0) + \min(\lambda(e^x - 1), 0) \tag{3.8}$$

donde λ es un parámetro predefinido para el control del valor al cual la ELU se satura cuando hay una entrada negativa.

3.2. Aprendizaje no supervisado

La entrada a la red, está constituida por un vector $x^0(k) \in \Re^{n_y+n_u+1}$, con la siguiente forma:

$$x^0(k) = \left[\begin{array}{ccccccc} y(k-1) & ... & y(k-n_y) & u(k) & u(k-1) & ... & u(k-n_u) \end{array} \right]^T \tag{3.9}$$

Para facilitar la lectura, se omite la dependencia a k, teniendo en cuenta que los valores cambian en cada iteración.

La inclusión de ruido a la estructura se muestra en la Figura 3.3, donde se observa que el nuevo vector de entrada a la red se definen como:

$$x^0_{ruido}(k) = x^0(k) + \rho(k) \tag{3.10}$$

con ρ, el ruido en los datos de entrada.

Capas Convolucionales

Cada capa convolucional consiste de h filtros $K_h \in \mathbb{R}^{f_l}$, que se convolucionará con el vector de entrada a esa capa. El mapa de características generado al aplicar cada filtro, $x^{(l)}_h \in \mathbb{R}^{p_l}$, está dado por:

$$x^{(l)}_{ih} = \sum_{a=0}^{f_l-1} K_{h,a} y^{(l-1)}_{h,i+a} + b^{(l)}_h \tag{3.11}$$

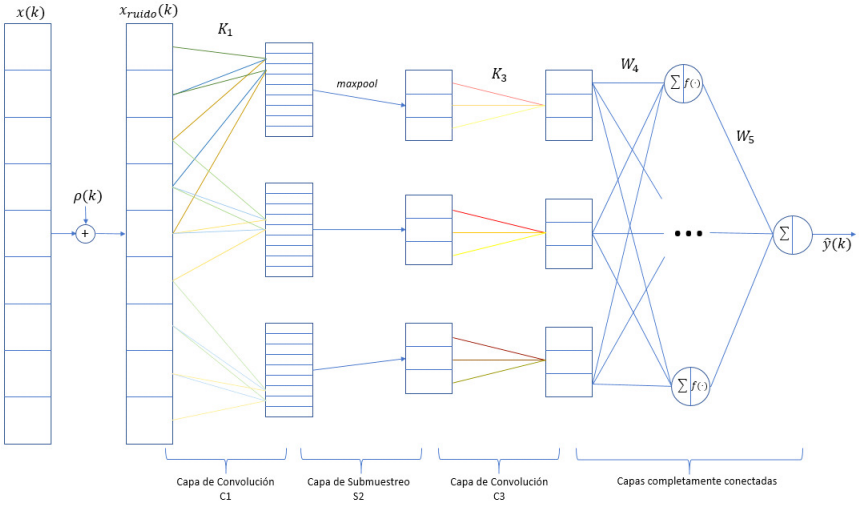


Figura 3.3: Estructura de la CNN para modelación con ruido en los datos de entrada.

con $i = 1, ..., p_l$, l indica el índice de la capa actual y $b_h^{(l)}$ es el sesgo correspondiente a cada filtro.

Para la primera capa, el vector de entrada $x^0 = y^0$ y es común para todos los filtros en esta capa. La operación de convolución se realiza con un paso $S = 1$ y un relleno de ceros $P = 1$. Por lo tanto, la dimensión de salida es $p_l = (W - f_l + 2P)/S \ + 1$, donde W es la dimensión del vector de entrada y_h^{l-1}. La salida de la capa de convolución, y_h^l, es entonces:

$$y_h^{(l)} = f\left(x_h^{(l)}\right) \tag{3.12}$$

Con $f(\cdot) = \tanh(\cdot)$.

Capa de submuestro

Una capa de submuestreo se coloca después de la primera capa convolucional con el fin de disminuir el tamaño de los datos. La operación que se realiza es *max-pooling* de dimensión s_2, con lo cual la salida de esta capa es:

$$y_h^{(l)} = \text{maxpool}\left(y_h^{(l-1)}, s_2\right) \in \Re^{p_l} \tag{3.13}$$

Capa completamente conectada

Se utilizan dos capas completamente conectadas en la parte final de la red, con una unidad en la capa de salida y L-unidades en la capa oculta. El vector de entrada a la capa oculta se obtiene concatenando las salidas de la segunda capa convolucional de la siguiente manera:

$$X = \left[\begin{array}{ccc} y_1^{(3)T} & \ldots & y_h^{(3)T} \end{array} \right]^T \in \Re^{h \cdot p_3} \tag{3.14}$$

La salida de la red se obtiene mediante:

$$\hat{y}(k) = W_5 \cdot f\left(W_4 \cdot X \ + b_4\right) \ + b_5 \tag{3.15}$$

Siendo $W_5 \in \Re^{1xL}$ los pesos en la capa de salida, $W_4 \in \Re^{Lxh \cdot p_3}$ los pesos de la capa oculta, $b_4 \in \Re^L$ y $b_5 \in \Re^1$ los sesgos de sus correspondientes capas, y $f(\cdot) = \tanh(\cdot)$.

El total de parámetros de aprendizaje de la red, incluye los elementos de los filtros de las dos capas convolucionales, el sesgo de cada uno de estos filtros y los pesos con sus sesgos en las dos capas completamente conectadas de la salida.

Para la actualización de los parámetros de la red, se propone el uso algoritmos aleatorios [50]. Por ello, los parámetros de las capas convolucionales y de la capa oculta se mantendrán fijos desde un inicio [51, 52], y solamente se actualizarán los parámetros de la capa de salida de manera supervisada. Esto se ejemplifica en la Figura 3.4.

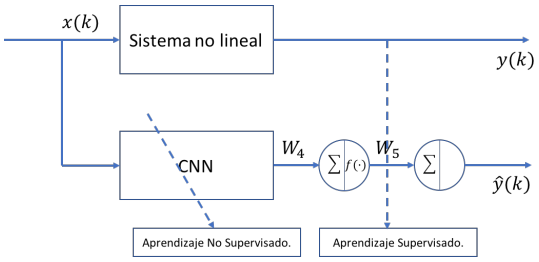


Figura 3.4: Esquema de aprendizaje.

La inicialización de los parámetros de las capas convolucionales se hace de manera alea-

toria dentro de un intervalo definido por [53]:

$$K_h^l \in \left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \tag{3.16}$$

donde n representa el total de entradas a esa capa. La inicialización de los pesos de las capas completamente conectadas se realiza de manera aleatoria en el intervalo $[0, 1]$.

Las dos capas convolucionales, se utilizan como filtros, en particular para disminuir el ruido que se pueda presentar en los datos de entrada, debido a que la operación de convolución funciona como un filtro lineal en el dominio espacial de los datos de entrada, siendo K_h^l los kernels de convolución.

3.3. Aprendizaje por mínimos cuadrados

Los parámetros de la capa de salida se actualizan mediante la pseudoinversa generalizada de Moore-Penrose [54] que se aplica a sistema lineales y se definirá a continuación.

Definición 3.1 *La matriz $A^+ \in \Re^{n\times m}$ es la matriz inversa de Moore-Penrose de $A \in \Re^{m\times n}$ si*

$$AA^+A = A, \quad A^+AA^+ = A^+, \quad (AA^+)^T = AA^T, \quad (A^+A)^T = A^+A \tag{3.17}$$

En particular, cuando A es de rango completo por columnas

$$A^+ = (A^TA)^{-1}A^T \tag{3.18}$$

Cuando A es de rango completo por filas

$$A^+ = A^T(AA^T)^{-1} \tag{3.19}$$

Definición 3.2 *$x_0 \in \Re^n$ se dice que es la solución de norma mínima por mínimos cuadrados del sistema lineal $y = Ax$ si*

$$\|x_0\| \leq \|x\|, \quad \forall x \in \{x : \|Ax - y\| \leq \|Az - y\|, \forall z \in \Re^n\} \tag{3.20}$$

donde $y \in \Re^m$ y $\|\cdot\|$ es la norma euclidiana

La solución por minímos cuadrados de un sistema lineal $y = Ax$, x_0, es:

$$\|Ax_0 - y\| = \min_x \|Ax - y\| \tag{3.21}$$

Sea By la solución de norma mínima por mínimos cuadrados del sistema lineal, si y solo si $B = a^+$

Reescribiendo todo el modelo de la CNN se tiene:

$$\hat{y}(k) = W_5\Phi(k) \tag{3.22}$$

donde $\Phi(k) = f\left(W_4 \cdot X + b_4\right)$. Esta es una representación lineal en los parámetros del sistema de la forma $y = Ax$, donde A puede o no ser una matriz cuadrada, la solución x puede encontrarse por la pseudoinversa generalizada de Moore-Penrose. El objetivo del aprendizaje es encontrar los valores de W_5 de tal manera que la función de costo J se minimice, siendo:

$$J = \sum_k \|y(k) - \hat{y}(k)\|^2 \tag{3.23}$$

Para esto, considere el conjunto de datos de entrenamiento $y(k)$ y $\Phi(k)$ con $k = 1, 2, \ldots, Q$, siendo Q el total de datos de entrenamiento. Agrupando todo el conjunto de entrenamiento:

$$\hat{Y} = \begin{bmatrix} \hat{y}(1) & \hat{y}(2) & \ldots & \hat{y}(Q) \end{bmatrix} = \begin{bmatrix} W_5\Phi(1) & W_5\Phi(2) & \ldots & W_5\Phi(Q) \end{bmatrix} = W_5\Psi \tag{3.24}$$

donde $\Psi = \begin{bmatrix} \Phi(1) & \Phi(2) & \ldots & \Phi(Q) \end{bmatrix}$. Considerando el error de modelado en cada instante $e(k) = y(k) - \hat{y}(k)$. Podemos reescribir (3.24) como:

$$\begin{gathered} Y = \begin{bmatrix} y(1) & y(2) & ... & y(Q) \end{bmatrix} = \begin{bmatrix} W_5\Phi(1) + e(1) & W_5\Phi(2) + e(2) & ... & W_5\Phi(Q) + e(Q) \end{bmatrix} \\ Y = W_5\Psi + E \end{gathered} \tag{3.25}$$

donde $E = \begin{bmatrix} e(1) & e(2) & \ldots & e(Q) \end{bmatrix}$. Para minimizar la función de costo, se necesita $\frac{\partial J}{\partial W_5} = 0$. Utilizando (3.19) se tiene que W_5^* puede minimizar a J cuando:

$$W_5^* = Y\Psi^T(\Psi\Psi^T)^{-1} = Y\Psi^+ \tag{3.26}$$

Dado que W_5^* es una solución por mínimos cuadrados de $Y = W_5^*\Psi + E$, este alcanza el error de aproximación mas pequeño dentro de todo el conjunto de entrenamiento y además de tener la norma mas pequeña para una solución por mínimos cuadrados de $Y = W_5^*\Psi$.

Los parámetros de la capa de salida, se calculan de acuerdo a (3.26). Con esto, junto con las inicializaciones de los parámetros de las capas convolucionales y de la capa oculta, se puede obtener resultados en la etapa de generalización muy buenos.

Se hace mención en [55] sobre problemas que se presentan al entrenar las capas ocultas y utilizando la pseudoinversa generalizada de Moore-Penrose para actualizar los parámetros de la capa de salida.

3.4. Simulación

3.4.1. Horno de Gas

Estos datos de prueba son obtenido de [56]. Los datos son la velocidad del gas en ft/s, $u(k)$ y el porcentaje de CO_2 presente en el gas de salida $y(k)$. El conjunto de datos tiene 296 parejas de datos $(u(k), y(k))$ en un intervalo de 9 segundos, de las cuales 200 muestras se utilizan para el aprendizaje y las restantes para la etapa de generalización. El vector de entrada se escoge de acuerdo a (3.9), con $n_y = 4$ y $n_u = 4$. Los datos de entrada se normalizan al intervalo $[-1, 1]$.

Valores Numéricos

Para las capas convolucionales, se escogen 3 filtros por capa, $h_1 = h_3 = 3$, de dimensión $f_1 = f_3 = 3$. La dimensión del mapa de características de cada filtro en la capa correspondiente es $p_1 = 9$ y $p_3 = 3$. En la capa de submuestreo, se realiza una operación maxpool con $s_2 = 3$, teniendo como salida un vector de dimensión tres. En la capa oculta se tiene un total de $L = 18$ neuronas. La red Neuronal con la que se realiza la comparación consta de $L = 80$ neuronas en la capa oculta y un total de 10 entradas, donde estas entradas corresponden al vector descrito en (3.9) con $n_y = 4$ y $n_u = 5$. En la Figura 3.5 se muestra que la parte de generalización para la CNN tiene buenos resultados, en ciertas regiones no coinciden, pero

en su mayoría el seguimiento lo hace de manera correcta. La Figura 3.6 muestra el resultado de la Red Neuronal de dos capas, el cual tiene un comportamiento muy parecido a la CNN.

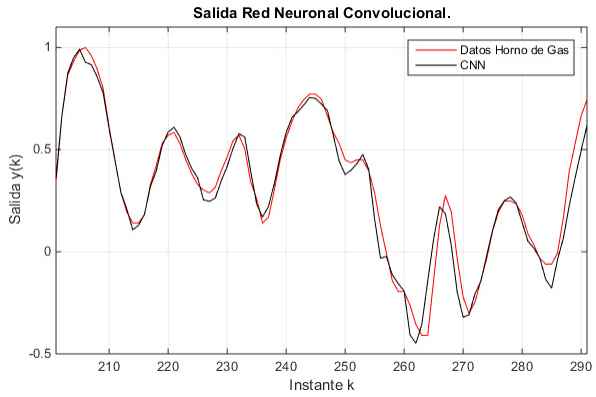


Figura 3.5: Red Neuronal Convolucional.

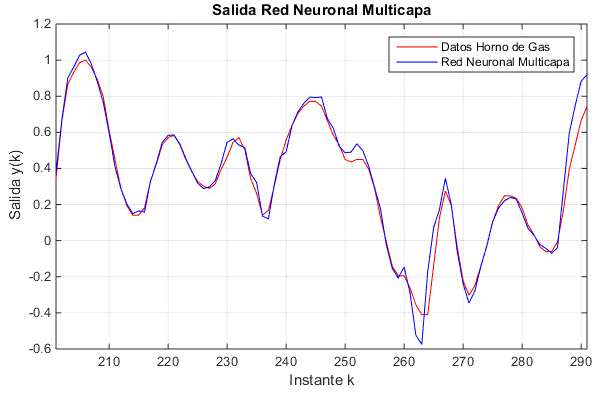


Figura 3.6: Red Neuronal Multicapa.

Con estos resultados se puede concluir que las dos redes tiene resultados muy parecidos y podrían mejorar o empeorar de acuerdo a las valores iniciales de los parámetros de ambas redes.

La función *wgn* de MATLAB, genera ruido blanco Gaussiano, el cual se agrega a los datos de entrada de la red. Las inicializaciones de los parámetros es la misma que cuando no hay ruido en los datos de entrada. Los resultados se muestran a continuación en las Figuras 3.7, 3.8. Como se observa en la Figura 3.7, el ruido presente en los datos de entrada no afecta demasiado en la generalización, la identificación sigue siendo adecuada. Por otro lado, con la red neuronal de dos capas se observa que ya no puede hacer una identificación adecuada del sistema, Figura 3.8.

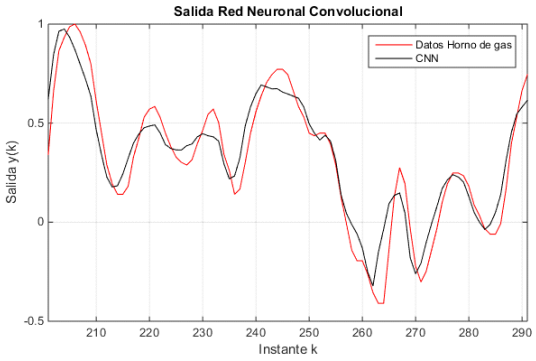


Figura 3.7: Red Neuronal Convolucional con ruido blanco Gaussiano.

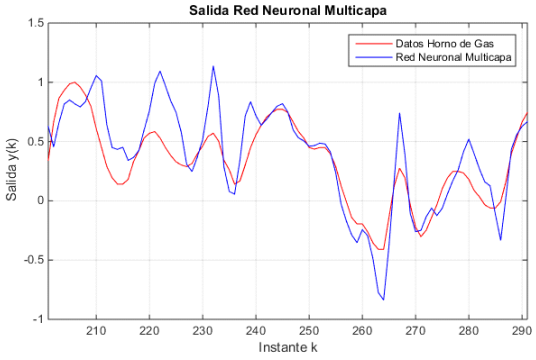


Figura 3.8: Red Neuronal Multicapa con ruido blanco Gaussiano.

Ahora se presentan los resultados obtenidos al aplicar un ruido aleatorio. Las mismas inicializaciones de los parámetros es utilizada como en los casos anteriores. En la Figura 3.9 se muestra el resultado de la red comparado con los datos originales de prueba. Aunque este presente el ruido en los datos de entrada, la CNN mantiene la forma general de la salida, mientras que la Red Neuronal Multicapa, Figura 3.10, pierde por completo la forma de la señal de referencia.

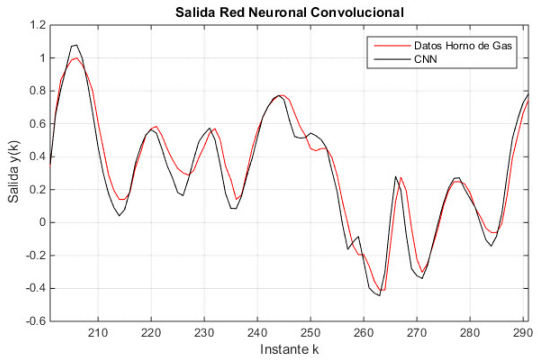


Figura 3.9: Red Neuronal Convolucional con ruido.

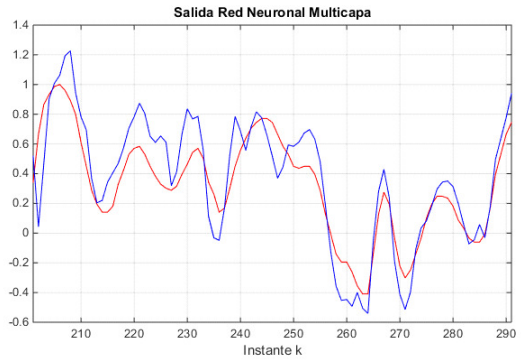


Figura 3.10: Red Neuronal Multicapa con ruido.

Para terminar con estas simulaciones se presenta una comparación de los errores de identificación obtenidos entre la CNN y la red multicapa. En la Figura 3.11 se muestra la

comparación sin la presencia de ruido. Mientras que en la Figura 3.12 se muestra ante la presencia de ruido, de esta se puede notar que la CNN tiene un error mas pequeño durante casi toda la etapa de generalización.

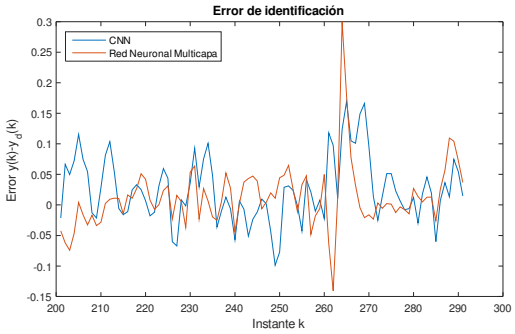


Figura 3.11: Error de identificación horno de gas.

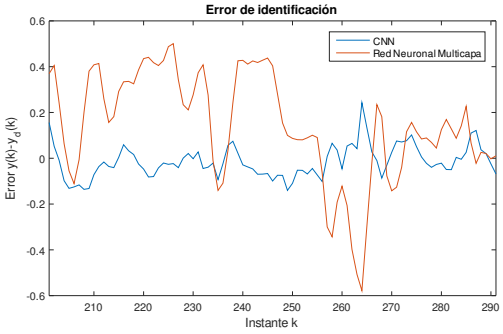


Figura 3.12: Error de identificación horno de gas con ruido.

En la Tabla 3.1 se muestra la comparación de los errores medios cuadráticos para los resultados obtenidos utilizando los datos del horno de gas.

Como se observa, cuando no existe ruido en los datos de entrada, ambas redes tienen resultados muy parecidos y ante la presencia de ruido la red neuronal convolucional tiene los errores más pequeños.

Tabla 3.1: Error medio cuadrático Horno de gas.

	Sin ruido	Ruido Blanco	Ruido aleatorio
CNN	0.0017	0.0025	0.0059
MLP	0.0014	0.0379	0.0111

Para este sistema en particular, los resultados obtenidos utilizando la CNN son buenos, aún en la presencia de ruido en los datos de aprendizaje, la red puede identificar el sistema.

3.4.2. Datos Wiener-Hammerstein

El sistema Wiener-Hammerstein es una estrutura orientada a bloques muy conocida [57], la cual consta de una función no lineal que se encuentra entre dos funciones lineales invariantes en el tiempo. Estas dos linealidades hacen más difícil la identificación de la parte no lineal. El conjunto de datos consta de 188000 parejas de entrada/salida. Para la parte de aprendizaje se utilizan 100000 datos y 88000 para la generalización, los datos se normalizan en el intervalo $[-1, 1]$. El vector de entrada se escoge de acuerdo a (3.9), con $n_y = 4$ y $n_u = 4$. La primera parte de la simulación consiste en utilizar los datos de aprendizaje sin ruido. Dada la cantidad de datos, se muestra solamente el intervalo $[180500, 180600]$. La Red Neuronal Multicapa mantiene la misma estructura con la misma cantidad de unidades en la capa oculta.

La segunda prueba se realiza com ruido blanco Gaussiano. Los resultados se muestran a continuación. Para la CNN, los resultados se presentan en la Figura 3.13. La señal de salida de la red presenta oscilaciones pero mantiene la forma genereal de la señal de referencia. La Red Neurona Multicapa, no funciona en la etapa de generalización, el ruido presente en los datos de entrada es muy grande y no puede generalizar de manera adecuada, esto se presenta en la Figura 3.14.

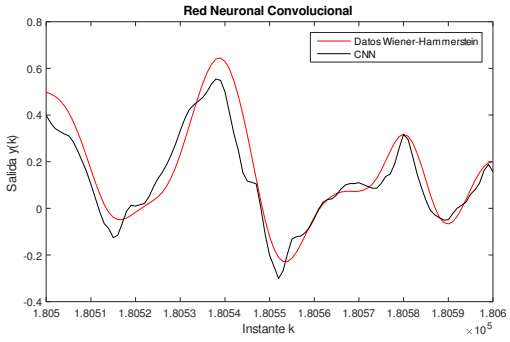


Figura 3.13: Red Neuronal Convolucional con ruido blanco.

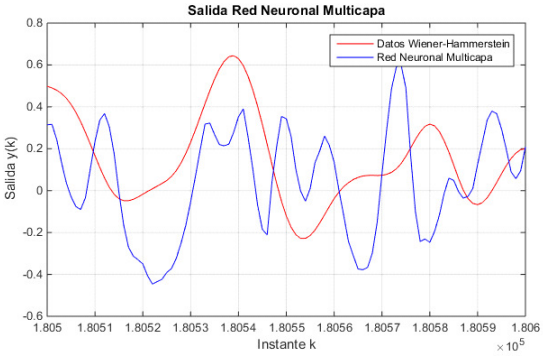


Figura 3.14: Red Neuronal Multicapa con ruido blanco.

La tercera parte de la simulación se realiza con ruido aleatorio, y los resultados son muy parecidos a los obtenidos con el ruido blanco Gaussiano. La Figura 3.15 muestra los resultados obtenidos con la CNN, la Figura 3.16 muestra los resultados obtenidos con la red neuronal multicapa (MLP), la generalización funciona mejor que en el caso anterior.

Como en el primer ejemplo de los datos del horno de gas, la CNN tiene mejores resultados ante la presencia de ruido, esto se esperaba debido a los filtros presentes en las capas convolucionales.

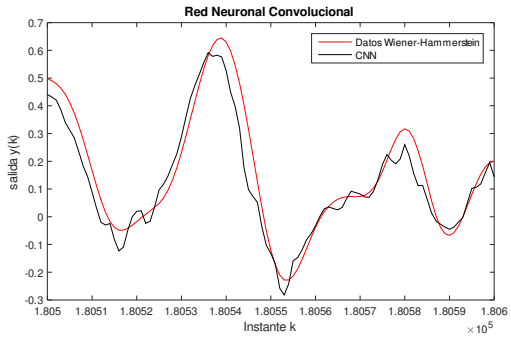


Figura 3.15: Red Neuronal Convolucional con ruido.

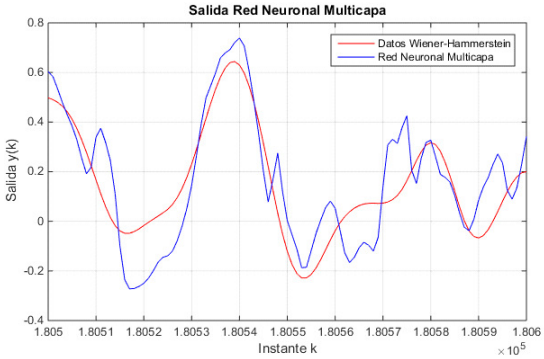


Figura 3.16: Red Neuronal Multicapa con ruido.

Se presenta la comparación de los errores de identificación para este sistema. Como se observa en la Figura 3.17 los errores obtenidos cuando no hay ruido son muy similares. Mientras que cuando hay ruido estos si se diferencian, ver Figura 3.18.

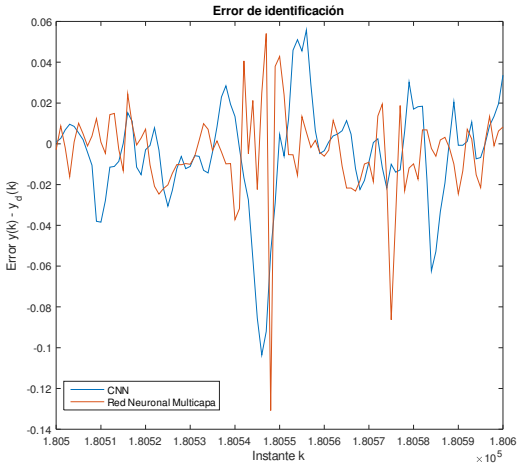


Figura 3.17: Error de identificación datos Wiener-Hammerstein.

Al igual que con la planta anterior, los errores medios cuadraticos se presentan la tabla siguiente. La red multicapa tiene un mejor desempeño comparando los errores cuando no hay ruido en los datos de entrenamiento con muy poca diferencia, pero la CNN la supera cuando hay ruido en base a los errores.

Tabla 3.2: Error medio cuadrático Datos Wiener-Hammerstein.

	Sin ruido	Ruido Blanco	Ruido aleatorio
CNN	$2.7171\text{x}10^{-4}$	0.0030	0.0019
MLP	$1.4929\text{x}10^{-4}$	0.0057	0.0029

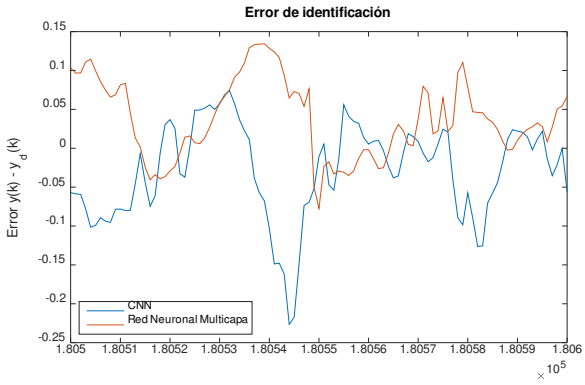


Figura 3.18: Error de identificación datos Wiener-Hammerstein con ruido.

3.4.3. Sistema no lineal

Para el tercer ejemplo se plantea utilizar el siguiente sistema no lineal [58]:

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{3.27}$$

donde u(k) es una señal periódica, la cual es diferente en la etapa de aprendizaje y en la de generalización. La señal de entrada se define como sigue:

$$u(k) = A\sin\left(\frac{\pi k}{50}\right) + B\sin\left(\frac{\pi k}{20}\right) \tag{3.28}$$

Para la etapa de aprendizaje, $A = B = 1$. En la etapa de generalización $A = 0{,}9$ y $B = 1{,}1$. Se generan 100 datos para la etapa de aprendizaje y 120 para la etapa de generalización. La estructura de la CNN se mantiene igual que en los ejemplos anteriores. Los resultados sin ruido en los datos de entrada se muestran en la Figura 3.19. La identificación de la planta tiene un buen desempeño, dado que los valores de iniciales de los parámetros de la red son aleatorios, los resultados pueden variar un poco de acuerdo a la inicialización. La Figura 3.20 y Figura 3.21 son los resultados obtenidos con ruido blanco Gaussiano y ruido aleatorio respectivamente. Ambos resultados son parecidos, presentan alteraciones en ciertas regiones, pero en general identificación de la planta aún en presencia de ruido es aceptable.

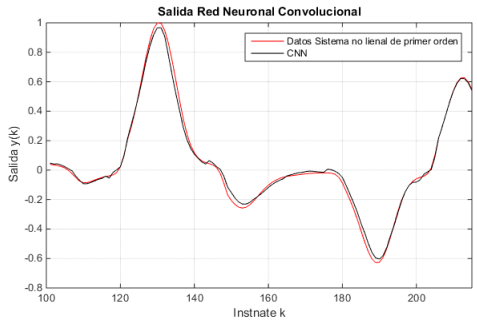


Figura 3.19: Red Neuronal Convolucional.

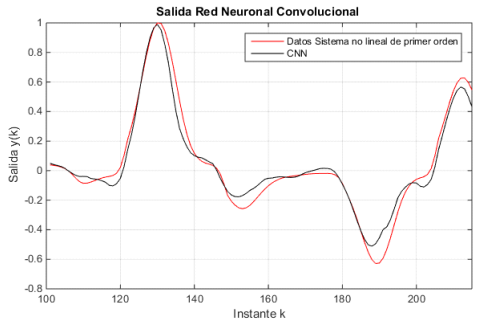


Figura 3.20: Red Neuronal Convolucional con ruido blanco.

Para este sistema, se muestra la comparación del error de identificación para los tres casos simulados, esto se ve en la Figura 3.22.

Se muestra a continucación la tabla con los valores de los errores medios cuadráticos para los tres resultados obtenidos.

Se puede observar que el ruido incrementa el error de genrealización, pero aún así, no es demasiado el incremento.

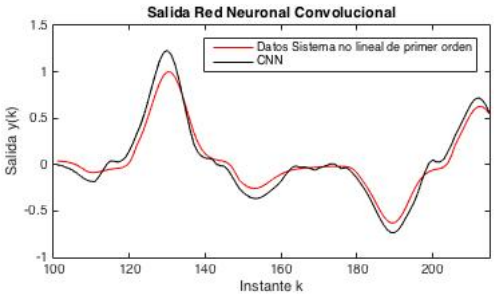


Figura 3.21: Red Neuronal Convolucional con ruido.

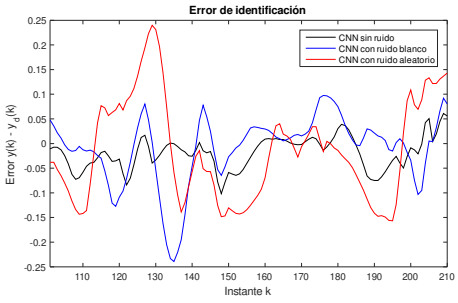


Figura 3.22: Error de identificación sistema no lineal.

Tabla 3.3: Error medio cuadrático Sistema no lineal.

	Sin ruido	Ruido Blanco	Ruido aleatorio
CNN	$7.1506 \text{x} 10^{-4}$	0.0025	0.0054

Capítulo 4

CNN para el modelado de sistemas: supervisado

En este capítulo se tratará el aprendizaje de las CNN con algoritmos en línea. El algoritmo a utilizar es el backpropagation (BP), modificándolo para utilizarlo con las redes convolucionales [59]. Además, se propone un aumento de filtros en las capas convolucionales para concluir sobre el efecto que tienen en la identificación de sistemas.

Se presenta a continuación los resultados de simulación de las mismas tres plantas del capítulo anterior, ahora utilizando el algoritmo backpropagation para la etapa de aprendizaje, ver Figura 4.1.

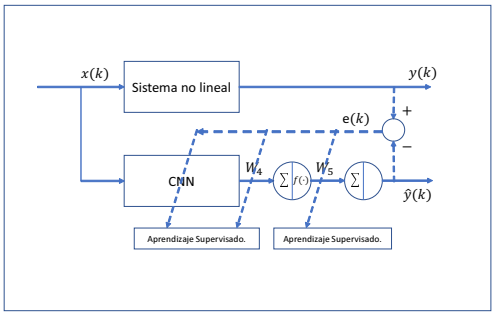


Figura 4.1: Esquema de aprendizaje.

Se presentan varios resultados, El primero conjunto de resultados, es utilizando solamente el algoritmo backpropagation para el aprendizaje con y sin ruido en los datos de entrenamiento. El segundo conjunto de resultados corresponde a una utilizar la inversa generalizada de Moore-Penrose para actualizar los pesos de la capas de salida, una vez aplicado el backpropagation, de igual manera con y sin ruido en los datos de entrenamiento. La estrucutra de la red para las simulaciones permanece igual, de acuerdo a la Figura 3.1 y se utilizan los mismos parámetros.

Los valores de η son diferentes para cada capa, las dos capas convolucionales comparten el mismo valor mientras que las dos capas completamente conectadas tienen valores diferentes entre sí. Para cada planta de prueba estos valores son indicados.

4.1. Aprendizaje para CNN

El sobreajuste es un problema que no debe despreciarse. Esto puede corregirse con técnicas de regularización, como Dropout. Fue introducida por Hinton et al [37] y ha sido probado que funciona bien reduciendo el sobreajuste. La salida de Dropout en [60], se utiliza en redes completamente conectadas y es $y = r * a\left(W^T x\right)$, donde $x = [x_1, x_2, ..., x_n]^T$ es la entrada de la capa completamente conectada, $W \in \Re^{dx}$ es la matriz de pesos, y r es un vector binario de tamaño d. Dropout puede prevenir que la red se vuelva muy dependiente de ciertas neuronas únicamente. Y puede forzar a la red a ser más precisa aun en la ausencia de información.

En la parte de optimización, existen varias técnicas para la optimización de la red, entre ellas se encuentra la inicialización de pesos [61]. Las redes convolucionales son dependientes de que exista una gran cantidad de datos de entrenamiento, lo que las hace que su aprendizaje sea muy difícil. Lo que conlleva una gran cantidad de parámetros y la función de pérdida sea no convergente. Para lograr una convergencia rápida, una inicialización apropiada de los pesos es requerida. Mientras que los sesgos pueden inicializarse en cero, los pesos deben seleccionarse de manera que no exista simetría en las unidades ocultas de una misma capa. Por ejemplo, Krizhecsky en [37] inicializa sus pesos a partir de una distribución Gausiana con media cero con desviación estándar de 0.01, y selecciona los sesgos de la segunda, cuarta y

quinta capa convolucionales en cero, al igual que todas las capas completamente conectadas.

Gradiente descendiente estocástico

El algoritmo de backpropagation es un método de aprendizaje estándar que utiliza el gradiente descendiente para actualizar los parámetros de la red [62]. Muchos algoritmos de optimización han sido propuestos. El algoritmo estándar actualiza los parámetros θ del objetivo $\mathcal{L}(\theta)$ como $\theta_{t+1} = \theta_t - \eta \nabla_\theta E\left[\mathcal{L}(\theta_t)\right]$, donde $E\left[\mathcal{L}(\theta_t)\right]$ es la expectativa de $\mathcal{L}(\theta)$ sobre todo el conjunto de aprendizaje y η la velocidad de aprendizaje. En el gradiente descendiente estocástico (SGD) [63] se estima el gradiente en base a un solo par entrada-salida escogido al azar del conjunto de entrenamiento:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}(\theta_t; x^{(t)}, y^{(t)}) \tag{4.1}$$

En la práctica la actualización de parámetros se calcula respecto a un pequeño lote en lugar de un par. La velocidad de convergencia se controla con el parámetro η_t. Algunos problemas que se presentan es que no es fácil elegir un parámetro de aprendizaje apropiado. Puede escogerse al comienzo un valor que de convergencia estable para posteriormente ir reduciendo el valor lentamente. Para hacer que la actualización del gradiente dependa de lotes con memoria y se acelere el aprendizaje, el momento se propone. La actualización clásica del momento es [64] :

$$\begin{aligned} \nu_{t+1} &= \gamma \nu_t - \eta_t \nabla_\theta \mathcal{L}(\theta_t; x^{(t)}, y^{(t)}) \\ \theta_{t+1} &= \theta_t + \nu_{t+1} \end{aligned} \tag{4.2}$$

donde ν_{t+1} es el vector de velocidad actual, γ es el término de momento el cual usualmente se coloca en 0.9. Otro método para utilizar el momento en la optimización del gradiente descendiente es el siguiente [65] :

$$\nu_{t+1} = \gamma \nu_t - \eta_t \nabla_\theta (\theta_t + \gamma \nu_t; x^{(t)}, y^{(t)}) \tag{4.3}$$

Comparándolo con el método clásico, en este primero el momento se mueve en la dirección del gradiente acumulado, se calcula el gradiente y luego se realiza la actualización del gradiente. Este logra que la optimización se mueva muy rápido y se logren mejores desempeños.

Métodos de paralización del Gradiente descendiente estocástico, mejoran este algoritmo para ser viables para máquinas de aprendizaje a gran escala. Los métodos de SGD pueden no converger. El proceso puede detenerse cuando el desempeño deje de crecer. Para prevenir el sobreajuste se puede detener la fase de aprendizaje en un periodo corto.

Normalización por lotes

La normalización de datos es usualmente el primer paso para el pre-procesamiento de datos [66] . La normalización global de datos logra que todos los datos tengan media cero y variancia unitaria. Sin embargo, mientras los datos fluyen a través de la red, causa la pérdida de capacidad de aprendizaje y precisión de la red. Suponga que la capa a normalizar tiene una entrada d-dimensional, $x = [x_1, x_2, ..., x_d]^T$. Primeramente, se normaliza la k-ésima dimensión como:

$$\hat{x}_k = \frac{(x_k - \mu_B)}{\sqrt{\delta_B^2 + \epsilon}} \tag{4.4}$$

donde μ_B y δ_B^2 son la media y variancia del lote pequeño respectivamente, y ϵ es una constante. La normalización de la entrada $\hat{x}_k$ se transforma en:

$$y_k = B\, N_{\gamma,\beta}(x_k) = \gamma \hat{x}_k + \beta \tag{4.5}$$

donde γ y β son parámetros de aprendizaje. La normalización por lote trae ventajas comparada con la normalización global de datos. Reduce la dependencia del gradiente en la escala de los parámetros o de su valor inicial, permitiendo el uso de relaciones de aprendizaje mucho más grandes sin tener el riesgo de la divergencia. También permite la utilización de funciones de activación saturadas sin que se quede estancado en el modelo saturado.

4.2. CNN con Backpropagation

Para utilizar el algoritmo backpropagation, se mantiene la misma estructura del vector de entrada propuesta en la ecuación (3.9). Los pesos sinápticos se actualizan en cada iteración de acuerdo al algoritmo extraido de [14, 67, 68, 69, 70].

La función de costo a minimizar J esta dada por:

$$J(k) = \frac{1}{2}e^2(k) \tag{4.6}$$

con $e(k) = \hat{y}(k) - y_d(k)$, el error en cada instante de muestreo k. Considerando que la salida la CNN esta dada por:

$$\hat{y} = \sum_{i=1}^{L} w_i^{(5)} \cdot y_i^{(4)} \tag{4.7}$$

donde $y_i^{(4)} = \tanh(\sum_{j=1}^{h \cdot \mathrm{mp}_3} w_{ij}^{(4)} \cdot x_j)$. Siendo x_j los elemetos de salida de la capa convolucional C_3.

Para la capa de salida de la red, el algoritmo de backpropagation es el utilizado como si se tratara de una red multicapa. La actualización de los i pesos sinápticos $w_i^{(5)}$, está dada por:

$$w_i^{(5)}(k+1) = w_i^{(5)}(k) - \eta \Delta w_i^{(5)}(k) \tag{4.8}$$

donde el término $\Delta w_i^{(5)}(k)$ es el gradiente la función de costo respecto al peso sináptico a actualizar:

$$\Delta w_i^{(5)}(k) = \frac{\partial J}{\partial w_i^{(5)}} \tag{4.9}$$

los cuales se obtienen aplicando la regla de la cadena

$$\frac{\partial J}{\partial w_i^{(5)}} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial \hat{y}}\frac{\partial \hat{y}}{w_i^{(5)}} \tag{4.10}$$

con

$$\frac{\partial J}{\partial e} = e, \qquad \frac{\partial e}{\partial \hat{y}} = 1, \qquad \frac{\partial \hat{y}}{w_i^{(5)}} = y_i^{(4)} \tag{4.11}$$

De (4.9), (4.21) y (4.22), se tiene que la actualización de los pesos de la capa de salida es:

$$w_i^{(5)}(k+1) = w_i^{(5)}(k) - \eta e \; y_i^{(4)} \tag{4.12}$$

Para la capa oculta se aplica el mismo algoritmo, teniendo que la actualización de los pesos sinápticos está dado por:

$$w_{ij}^{(4)}(k+1) = w_{ij}^{(4)}(k) - \eta \Delta w_{ij}^{(4)}(k) \tag{4.13}$$

De la misma manera aplicando la regla de la cadena obtenemos las expresiones para $\Delta w_j^{(4)}(k)$:

$$\frac{\partial J}{\partial w_{ij}^{(4)}} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y_i^{(4)}}\frac{\partial y_i^{(4)}}{\partial w_{ij}^{(4)}} \tag{4.14}$$

donde cada uno de los términos esta definido como:

$$\frac{\partial J}{\partial e} = e, \qquad \frac{\partial e}{\partial \hat{y}} = 1, \qquad \frac{\partial \hat{y}}{\partial y_i^{(4)}} = w_i^{(5)}, \qquad \frac{\partial y_i^{(4)}}{\partial w_{ij}^{(4)}} = \left(1 - \left(y_i^{(4)}\right)^2\right) x_j \tag{4.15}$$

Utilizando (4.14) y (4.15) se tiene que:

$$w_{ij}^{(4)}(k+1) = w_{ij}^{(4)}(k) - \eta \; e \, w_i^{(5)} \left(1 - \left(y_i^{(4)}\right)^2\right) x_j \tag{4.16}$$

Por último,dado que el vector X corresponde a la concatenación de las salidas de los filtros de la capa C_3, el error propagado a la salida de cada uno de estos filtros está dado por la siguiente expresión:

$$\frac{\partial J}{\partial x_j} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial \hat{y}}\left(\sum_i \frac{\partial \hat{y}}{\partial y_i^{(4)}}\frac{\partial y_i^{(4)}}{\partial x_j}\right) \tag{4.17}$$

con:

$$\frac{\partial J}{\partial e} = e, \qquad \frac{\partial e}{\partial \hat{y}} = 1, \qquad \frac{\partial \hat{y}}{\partial y_i^{(4)}} = w_i^{(5)}, \qquad \frac{\partial y_i^{(4)}}{\partial x_j^{(3)}} = \left(1 - \left(y_i^{(4)}\right)^2\right) w_{ij}^{(4)} \tag{4.18}$$

Por lo tanto se tiene que:

$$\frac{\partial J}{\partial x_j} = e\, w_i^{(5)} \left(1 - \left(y_i^{(4)}\right)^2\right) w_{ij}^{(4)} \tag{4.19}$$

Para obtener el error propagada a cada filtro basta con tomar los elementos x_j que correspondan a la salidas de cada filtros de la capa C_3. Para las capas convolucionales y de submuestreo, se tiene que modificar el algoritmo de backpropagation, ya que en las capas de submuestro solamente se reduce el tamaño de los datos de entrada y en las capas convolucionales se realiza una operación de convolución.

Capas de submuestreo

En estas capas, dado que no se realiza ninguna operación más que la de reducir la cantidad de datos, al gradiente del error debe aplicarsele una operación contraria, en general una operación *up*, la cual aumenta la dimensión del gradiente para coincidir con la dimensión de los datos de entrada. Al aplicar la operación maxpool, el gradiente se propaga en la dirección del elemento que proporcionó la salida de mayor magnitud en la capa anterior, y los demás elementos se colocan a cero. De esto se tiene:

$$\frac{\partial J}{\partial y^{(l-1)}} = up\left(\frac{\partial J}{\partial y^{(l)}}\right) \tag{4.20}$$

donde la función $up\,(\cdot)$ representa la operación para incrementar la dimensión de los datos para que concuerden con la capa anterior.

Capas convolucionales

Dado el proceso de backpropagation, llega el punto donde se necesita propagar el error a través de las capas convolucionales. Para ello es necesario obtener las expresiones matemáticas de los gradiente respecto a los valores de la filtros y el gradiente respecto a la salida de la capa anterior.

Aplicando la regla de la cadena, utilizando la misma notación que en el capítulo anterior, obtenemos $\partial J/\partial K_i^l$ de la capa actual l utilizando $\partial J/\partial y_i^l$, el cual ya se obtuvo mediante el proceso de propagación, por lo tanto tenemos:

$$\frac{\partial J}{\partial K_a^{(l)}} = \sum_{i=0}^{N-f_l} \frac{\partial J}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial K_i^{(l)}} \tag{4.21}$$

con N la dimensión del vector de entrada a la capa y siendo:

$$\frac{\partial y_i^{(l)}}{\partial x_i^{(l)}} = f\acute{}(x_i^{(l)}) \tag{4.22}$$

Dado que cada elemento de los filtros K_i, se aplican a varios elementos x_i, es por eso que es necesario realizar la sumatoria sobre los elementos con los que se opere el elemento del filtro. Refiriendose a (3.11), se puede observar que $\frac{\partial x_i^{(l)}}{\partial K_a^{(l)}} = y_{i+a}^{(l-1)}$. Con esto podemos calcular el gradiente del error con respecto a los elementos de los filtros:

$$\frac{\partial J}{\partial K_{h,a}^{(l)}} = \sum_{i=0}^{N-f_l} \left(\delta_{h,i}^{(l)}\right) y_{h,i+a}^{(l-1)} \tag{4.23}$$

donde:

$$\delta_{h,i}^{(l)} = \frac{\partial J}{\partial y_{h,i}^{(l)}} f\acute{}(x_{h,i}^{(l)}) \tag{4.24}$$

Una vez obtenido el error en la capa actual, es necesario propagarlo hacia la capa anterior, para ello, utilizando nuevamente la regla de la cadena se tiene una expresión muy parecida a (4.23):

$$\frac{\partial J}{\partial y_i^{(l-1)}} = \sum_{a=0}^{f_l-1} \frac{\partial J}{\partial x_{i-a}^{(l)}} \frac{\partial x_{i-a}^{(l)}}{\partial y_i^{(l-1)}} = \sum_{a=0}^{f_l-1} \frac{\partial J}{\partial x_{i-a}^{(l)}} K_a^{(l)} \tag{4.25}$$

Con estas expresiones se puede actualizar los valores de los filtros de las capas convolucionales y propagar el error hacaia capas anteriores. La ecuación (4.25), es equivalente a realizar una convolución de δ^l con el filtro K rotado 180°, de aquí se tiene:

$$\frac{\partial J}{\partial y_h^{(l-1)}} = \delta_h^{(l)} * rot180(K_h^{(l)}) \tag{4.26}$$

En ambos casos, el subíndice h indica el filtro sobre el cual se realiza la operación en la capa l. Para obtener la actualización de los filtros se utiliza la siguiente exresión:

$$K_h^{(l)}(k+1) = K_h^{(l)}(k) - \eta \frac{\partial J}{\partial K_h^{(l)}} \tag{4.27}$$

Desarrollando las expresiones anteriores para la red, se tiene que el gradiente del error respecto a los elementos de los filtros de la capa C_3, (4.23) esta dado por:

$$\frac{\partial J}{\partial K_{h,a}^{(3)}} = \sum_{i=0}^{p_2 - f_l} \left(\delta_{h,i}^{(3)}\right) y_{h,i+a}^{(2)} \tag{4.28}$$

De donde el segundo término corresponde a los elementos de la capa de submuestreo S_2. El primer término se obtiene a partir de (4.24) con:

$$\delta_{h,i}^{(3)} = \frac{\partial J}{\partial y_{h,i}^{(3)}} \acute{f}(x_{h,i}^{(3)}) = \frac{\partial J}{\partial y_{h,i}^{(3)}} \left(1 - \left(y_{h,i}^{(3)}\right)^2\right) \tag{4.29}$$

y $\frac{\partial J}{\partial y_{h,i}^{(3)}}$ se obtiene a partir de (4.24). Esto se realiza para elemento de todos los filtros de la capa C_3.

El error se sigue propagando hacia capas inferiores, y para la salida de la capa S_2 a partir de (4.26) se tiene:

$$\frac{\partial J}{\partial y_h^{(2)}} = \delta_h^{(3)} * rot180(K_h^{(3)}) \tag{4.30}$$

con lo cual se obtiene el vector del error para cada elemento de esta capa.

Dado que esta es una capa de submuestreo, la propagacion del error está dada por:

$$\frac{\partial J}{\partial y^{(1)}} = up\left(\frac{\partial J}{\partial y^{(2)}}\right) \tag{4.31}$$

el cual se ajusta al tamaño de la salida de la capa C_1 y solo en la dirección en la cual se tuvo la mayor respuesta en la etapa hacia adelante. Por último, la actualización de los pesos de la capa C_1 se realiza de acuerdo a las siguientes expresiones:

$$\begin{aligned} \frac{\partial J}{\partial K_{h,a}^{(1)}} &= \sum_{i=0}^{n_v+n_y+1-f_l} \left(\delta_{h,i}^{(1)}\right) x_{i+a}^{0} \\ \delta_{h,i}^{(1)} &= \frac{\partial J}{\partial y_{h,i}^{(1)}} \left(1 - \left(y_{h,i}^{(1)}\right)^2\right) \end{aligned} \tag{4.32}$$

Ejemplo

Para entender mejor el proceso de aprendizaje, se propone el siguiente ejemplo con valores númericos para observar la propagación del error atraves de la red. Considere una CNN de seis capas con la misma estructura propuesta en este capítulo. Considere ademas que el paso hacia adelante ya se realizó y tenemos todos los valores para empezar la propagación hacia atrás.

De (4.12) tenemos que:

$$w_1^{(5)}(2) = w_1^{(5)}(1) - \eta e\, y_1^{(4)} = 0{,}5 - (0{,}5)(-1{,}5)(-0{,}3) = 0{,}275 \tag{4.33}$$

los pesos restantes de esta capa se obtienen de manera similar, simplemente cambiando el peso a actualizar y el valor de salida de la capa anterior que afecta a ese peso.

Par la capa oculta, la actualización de los pesos esta definida por (4.16) y para el primer peso de capa se tiene:

$$\begin{aligned} w_{11}^{(4)}(2) &= w_{11}^{(4)}(1) - \eta\ e\, w_1^{(5)} \left(1 - \left(y_1^{(4)}\right)^2\right) x_1 \\ w_{11}^{(4)} &= 0{,}45 - (0{,}25)(-1{,}5)(,5)\left(1 - (-0{,}3)^2\right)(0{,}23) = 0{,}4892 \end{aligned} \tag{4.34}$$

Esto se realiza para todos los pesos de la capa oculta.

Para las capa convolucional C_3, se tiene que la actualización de un valor de un filtro esta dado por (4.27), numericamente:

$$K_1^{(3)}(2) = K_1^{(3)}(1) - \eta \frac{\partial J}{\partial K_1^{(3)}} \tag{4.35}$$

de donde para elemento del filtro, su dervida esta definida como (4.28):

$$\frac{\partial J}{\partial K_{1,0}^{(3)}} = \sum_{i=0}^{0} \left(\delta_{1,i}^{(3)}\right) y_{1,0}^{(2)} \tag{4.36}$$

Obteniendo $\delta_{1,0}^{(3)}$ a partir de (4.29) se tiene:

$$\delta_{1,0}^{(3)} = \frac{\partial J}{\partial y_{1,0}^{(3)}} \left(1 - \left(y_{1,0}^{(3)}\right)^2\right) = (0{,}24)\left(1 - (0{,}59)^2\right) = 0{,}1564 \tag{4.37}$$

entoces:

$$\frac{\partial J}{\partial K_{1,0}^{(3)}} = (0{,}1564)(0{,}845) = 0{,}1322 \tag{4.38}$$

de aquí se tiene que el filtro actualziado queda de la siguienete manera:

$$K_{1,0}^{(3)}(2) = K_{1,0}^{(3)}(1) - \eta \frac{\partial J}{\partial K_{1,0}^{(3)}} = 0{,}5212 - (0{,}3)(0{,}1322) = 0{,}4815 \tag{4.39}$$

Una vez actualizados los 3 elementos de los filtros de la capa C_3, se procede a obtener el gradiente del error de la capa anterior. Para ello se tiene el filtro $K_1^{(3)} = [\ 0{,}5212 \quad 0{,}2413 \quad 0{,}1241\ ]^T$ y $\delta_1^{(3)} = [\ 0{,}1564 \quad 0{,}3251 \quad 0{,}1268\ ]^T$. Siguiendo la ecuación (4.30) llegamos a la expresión:

$$\begin{aligned} \frac{\partial J}{\partial y_1^{(2)}} = \delta_1^{(3)} * rot180(K_1^{(3)}) &= \begin{bmatrix} 0{,}1564 \\ 0{,}3251 \\ 0{,}1268 \end{bmatrix} * rot180\left(\begin{bmatrix} 0{,}5112 \\ 0{,}2413 \\ 0{,}1241 \end{bmatrix}\right) \\ &= \begin{bmatrix} 0{,}1564 \\ 0{,}3251 \\ 0{,}1268 \end{bmatrix} * \begin{bmatrix} 0{,}1241 \\ 0{,}2413 \\ 0{,}5112 \end{bmatrix} \end{aligned} \tag{4.40}$$

Como ya se mencionó, se debe realizar un relleno con zeros al inicio y al final primer elemento para que se obtenga la operación correcta. Esto es:

$$\frac{\partial J}{\partial y_1^{(2)}} = \begin{bmatrix} 0 \\ 0{,}1564 \\ 0{,}3251 \\ 0{,}1268 \\ 0 \end{bmatrix} * \begin{bmatrix} 0{,}1241 \\ 0{,}2413 \\ 0{,}5112 \end{bmatrix} = \begin{bmatrix} 0{,}0781 \\ 0{,}1741 \\ 0{,}1968 \end{bmatrix} \tag{4.41}$$

Por ejemplo, 0,1741 = (0,1241)(0,1268) + (0,2413)(0,3151) + (0,5112)(0,1564).

La siguiente etapa corresponde a la capa de submuestro S_2 en la cual es necesario aumentar la dimensión de $\frac{\partial J}{\partial y_1^{(2)}}$ para coincidir con la capa anterior. Dadoq que se realizó una operación maxpool con reducción tres, el nuevo vector degradiente del error será de dimensión nueve, además de que solamente los elementos que aportaron la salida mas grande en la etapa hacia adelante se actualizarán y los demas se mantendrán.

$$\frac{\partial J}{\partial y_1^{(1)}} = up\left(\frac{\partial J}{\partial y_1^{(2)}}\right) = \begin{bmatrix} 0{,}0781 & 0 & 0 & 0 & 0 & 0{,}1741 & 0{,}1968 & 0 & 0 \end{bmatrix}^T \tag{4.42}$$

De la ecuación (4.42) se observa que solamente tres elementos del vector contienen valores diferentes de cero y en las posiciones que presentaron la salida mas grande en la etapa hacia adelante.

Para la capa C_1 el procedimiento es el mismo.

Con

$x^0 = \begin{bmatrix} 0{,}7781 & 0{,}7725 & 0{,}7682 & 0{,}7612 & 0{,}4312 & 0{,}3912 & 0{,}1688 & 0{,}1259 & 0{,}1127 & 0{,}1 \end{bmatrix}^T$

De (4.42) se tiene que para cada elemento de los filtros, su gradiente se calcula de la siguiente manera:

$$\frac{\partial J}{\partial K_{1,0}^{(1)}} = \sum_{i=0}^{7}\left(\delta_{1,i}^{(1)}\right)x_i^0 = (0{,}1241)(0{,}7781) + (-0{,}5412)(0{,}7725) + ... + (0{,}4363)(0{,}1688) = 0{,}35 \tag{4.43}$$

Finalmente, la actualización de los pesos de la capa C_1 es:

$$K_{1,0}^{(1)}(2) = K_{1,0}^{(3)}(1) - \eta\frac{\partial J}{\partial K_{1,0}^{(3)}} = 0{,}6123 - (0{,}3)(0{,}35) = 0{,}5073 \tag{4.44}$$

4.3. Simulaciones

4.3.1. Horno de gas

Los datos de prueba son obtenido de [56], los valores de η son los mismos para las simulaciones de esta planta, los cuales son:

$$\begin{aligned} \eta_{C_1} &= 0{,}15 \quad \eta_{C_3} = 0{,}15 \\ \eta_{F_5} &= 0{,}18 \quad \eta_{F_6} = 0{,}25 \end{aligned} \tag{4.45}$$

Los resultados utilizando backpropagation se muestran en la Figura 4.2. De esta figura podemos observar que a pesar del ruido en los datos de entrenamiento, la identificación de la planta se mantiene. En general el algoritmo BP presenta resultadods aceptables.

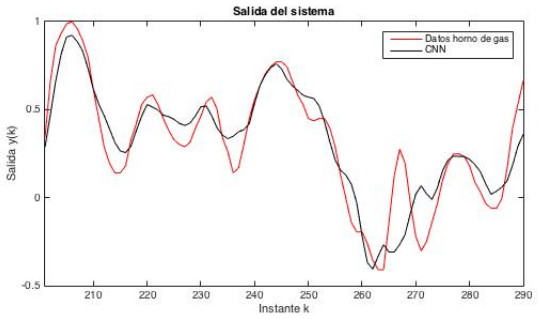


Figura 4.2: BP con ruido.

El siguiente resultado es utilizando métodos de mínimos cuadrados para obtener los pesos sinápticos de la capa de salida una vez aplicado el algoritmo BP. La Figura 4.3 muestra los resultados con ruido. Podemos notar que la identificación de la planta es adecuada aún en la presencia de ruido.

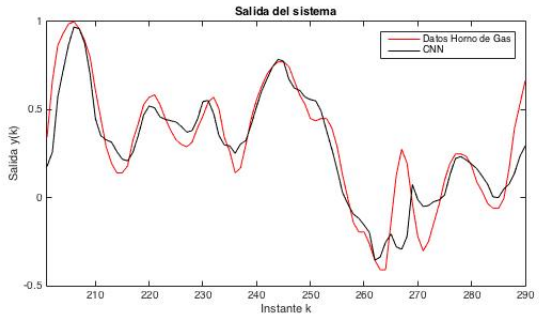


Figura 4.3: BP y mínimos cuadrados con ruido.

4.3.2. Sistema no lineal

Los datos de prueba son obtenido de [58], los valores de η son los mismos para las simulaciones de esta planta, los cuales son:

$$\begin{aligned} \eta_{C_1} &= 0{,}02 \quad \eta_{C_3} = 0{,}02 \\ \eta_{F_5} &= 0{,}33 \quad \eta_{F_6} = 0{,}3 \end{aligned} \tag{4.46}$$

Se presenta a continuación los resultados obtenidos con el algoritmo BP. En la Figura 4.4 se muestra el resultado sin ruido, la identificación es muy parecida a la del capítulo anterior. En la Figura 4.5 se presenta el resultado con ruido en los datos de entrenamiento, el cual tiene una respuesta casi parecida a los resultados sin ruido Figura 4.7.

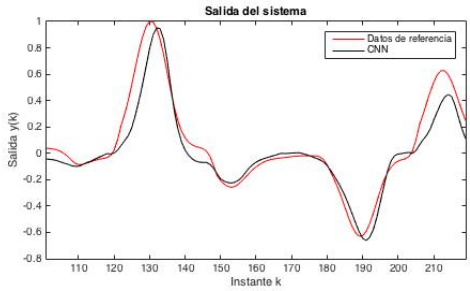


Figura 4.4: Backpropagation.

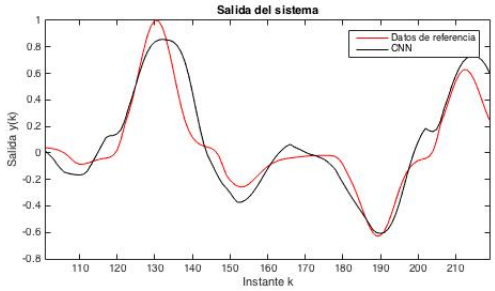


Figura 4.5: BP con ruido.

Ahora se presenta los resultados utilizando mínimos cuadrados. La Figura 4.6 muestra los resultados obtenidos cuando no hay ruido en los datos de entrada, se puede observar que la identificación es mejor que utilizando simplemente el BP, aun en la presencia de ruido.

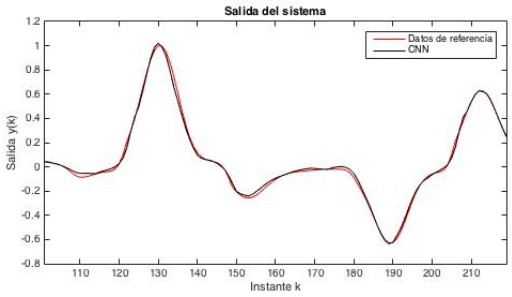


Figura 4.6: BP y mínimos cuadrados.

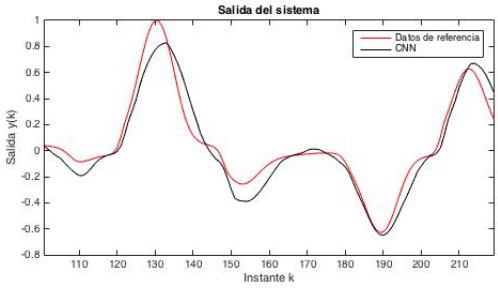


Figura 4.7: BP y mínimos cuadrados con ruido.

4.3.3. Datos Wiener-Hammerstein

Los datos de prueba son obtenido de [71], los valores de η son los mismos para las simulaciones de esta planta, los cuales son:

$$\begin{aligned} \eta_{C_1} &= 0{,}05 \quad \eta_{C_3} = 0{,}05 \\ \eta_{F_5} &= 0{,}3 \quad \eta_{F_6} = 0{,}34 \end{aligned} \tag{4.47}$$

Como en el capítulo anterior, solo se muestra una pequeña parte de la identificación debido a la gran cantidad de datos. Para esta planta en particular solamente se presentan los resultados obtenidos utilizando mínimos cuadrados con backpropagtion. Los resultados sin ruido y con ruido en los datos de entrenamiento se presentan en la Figura 4.8 y Figura 4.9 respectivamente. Aqui se observa que el ruido en los datos afecta el desempeño del algoritmo de aprendizaje pero aún asi mantiene la forma de la señal de referencia.

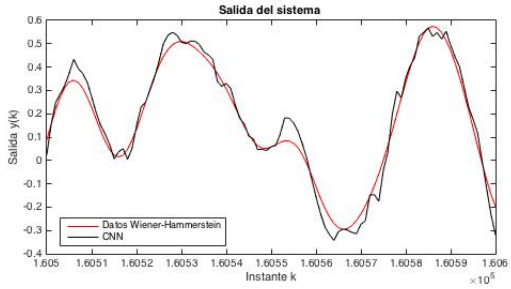


Figura 4.8: BP y mínimos cuadrados.

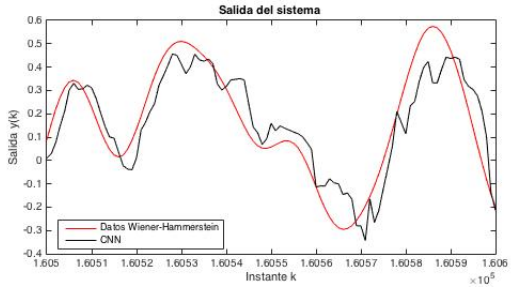


Figura 4.9: BP y mínimos cuadrados con ruido.

4.3.4. Cambio de estructura

En esta subsección se considera la modificación de la estructura de la CNN para realizar comparación en los resultados. La propuesta es aumentar la cantidad de filtros en las capas convolucionales, esto es, extraer una mayor cantidad de características de los datos de entrada del sistema, y alimentar las últimas dos capas con mayor información. La nueva estructura se presenta en la Figura 4.10, en la cual puede notarse que en las dos capas convolucionales puede haber un número variable de elementos.

En esta etapa el número de filtros en cada capa convolucional será de $h = 20$. Por consiguiente el número de elementos de las capas de submuestreo aumentarán en la misma cantidad. Además, en la capa de submuestro $S4$ se realizará una operación maxpool con $s4 = 3$, esto con la finalidad de disminuir los datos. En la estructura del capítulo anterior,

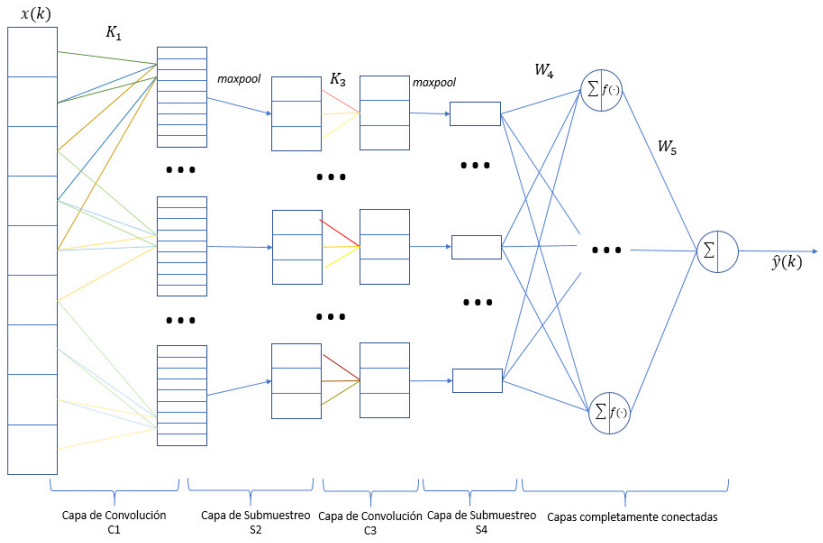


Figura 4.10: Estructura general de la CNN.

no era necesario realizar una operación *maxpool* en la capa de submuestro $S4$ ya que la dimensión de salida de la capa convolucional $C3$ era la adecuada para introducir los datos a las capas finales. Esto es:

$$\begin{aligned} y^{(4)} &= \text{máx}\, pool\left(y^{(3)}, s4\right) \\ X &= \left[\begin{array}{ccc} y_1^{(4)} & \ldots & y_h^{(4)} \end{array}\right]^T \in \Re^h \end{aligned} \tag{4.48}$$

Además, agregar esta capa de subuestreo modifca el algoritmo de aprendizaje. Por lo tanto, (4.19) hace referencia a la concatenación de los errores propagados de la salida de la capa de submuestreo S_4. Se agrega la siguiente expresión para incluir ela capa de submuestro en el aprendizaje:

$$\frac{\partial J}{\partial y^{(3)}} = up\left(\frac{\partial J}{\partial y^{(4)}}\right) \tag{4.49}$$

Se realizan simulaciones para las tres plantas, con y sin ruido pero ahora aplicando esta nueva estrucutra de la red. Para los datos Wiener-Hammerstein se realiza una simluación

extra, en la cual se utilizan 15 filtros adems de 20. Los resultados de simulación de cada planta se muestran en los apartados siguientes.

Horno de gas

Las primeras simulaciones se realizan utilizando el algoritmo de backpropagation para el aprendizaje, los resultados se presentan en la Figura 4.11, se puede notar que la identificación del sistema es muy parecida a la que se obtuvo cuando se utilizan 3 filtros en las capas convolucionales. Cuando hay presencia de ruido, la Figura 4.12 muestra que la red logra despreciar el ruido haciendo que la generalizacióntenga buenos resultados.

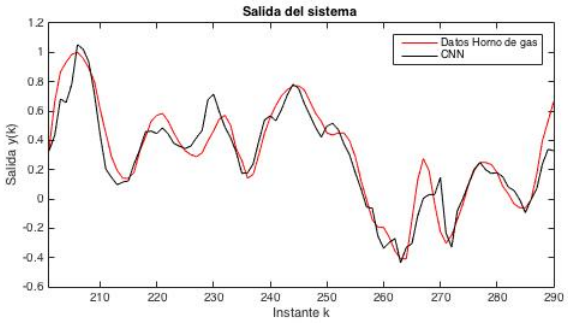


Figura 4.11: BP con 20 filtros.

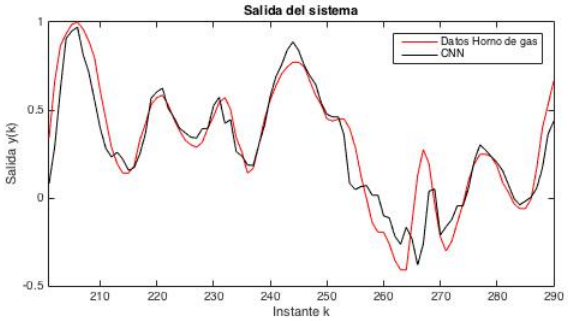


Figura 4.12: BP, 20 filtros y ruido.

La siguiente es aplicar un método de mínimos cuadrados despues del BP para actualizar los pesos sinápticos de la capa de salida, sin variar los demás parámetros de aprendizaje. Los resultados sin ruido en los datos se presentan en la Figura 4.13, se observa que la identificación la realiza mejor que solamente utilizando BP, sin embargo, sigue sin ser muy buena. Los resultados con ruido se presentan en la Figura 4.14.

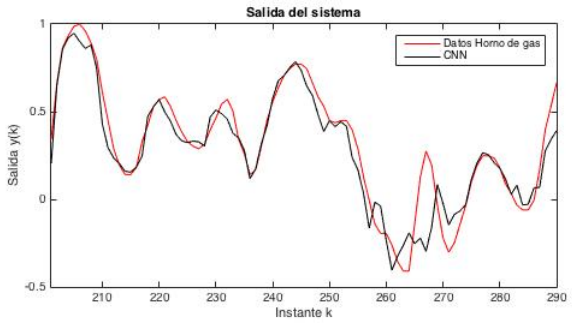


Figura 4.13: BP, mínimos cuadrados y 20 filtros.

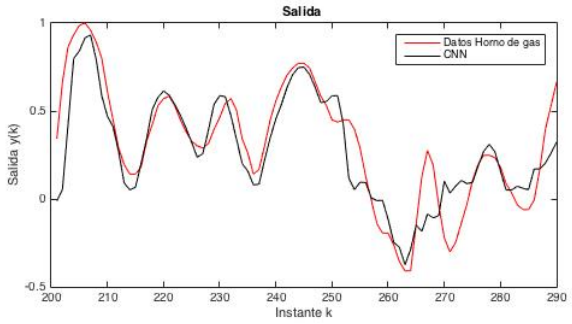


Figura 4.14: BP, mínimos cuadrados, 20 filtros y ruido.

Se hace una comparación de los errores de identificación con las dos configuraciones utilizando la misma técnica de aprendizaje. Se presentan las figuras con ruido en los datos. Se observa en la Figura 4.15 y Figura 4.16 utilizando BP y BP con mínimos cuadrados respectivamente.

Figura 4.15: Error de identificación con BP horno de gas.

Figura 4.16: Error de identificación con BP y mínimos cuadrados horno de gas.

En la tabla siguiente se presenta los errores medios cuadráticos para ambas simulaciones con 3 filtros y 20 filtros que se realizaron sobre los datos del horno de gas.

De la Tabla 4.1, se puede notar que la utilización de mínimos cuadrados reduce el error cuando no hay ruido en los datos, y la diferencia cuando hay ruido es muy pequeña en el caso de tres filtros en las capas convolucionales. Con 20 filtros, el error es muy parecido cuando no hay ruido, y el algorimto de backpropagation resulta un poco mejor cuando hay ruido.

En particular para este sistema, se obtuvieron mejores resultados utilizando solamente 3 filtros en cada capa convolucional, y utilizando mínimos cuadrados para atualizar los pesos

Tabla 4.1: Error medio cuadrático Horno de gas.

	Backpropagation	Backpropagation y mínimos cuadrados
3 Filtros		
Sin ruido	0.0061	0.0041
Ruido	0.0104	0.0104
20 filtros		
Sin ruido	0.0073	0.0072
ruido	0.0096	0.0107

de la capa de salida. Se puede notar que el error de generalización es menor para el caso donde tenemos los 3 filtros ya sea con o sin ruido.

Sistema no lineal

La simulación se realiza utilizando solamente BP. Se muestran los dos resultados obtenidos, con y sin ruido. La Figura 4.17 muestra que al utilizar este algoritmo, la identificación del sistema no es adecuada, al igual que con la presencia de ruido ver Figura 4.18. Aqui se observa que ante la presencia de ruido, aunque la identificaión no sea la mejor, la salida de la CNN no varía mucho ante él.

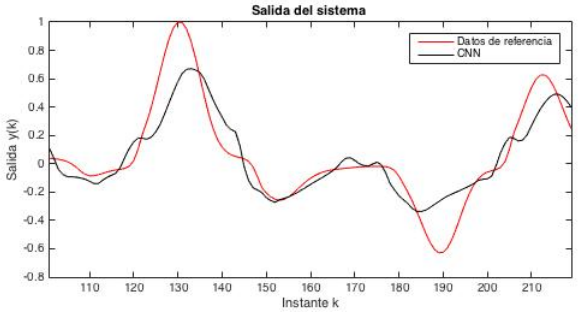


Figura 4.17: BP con 20 filtros.

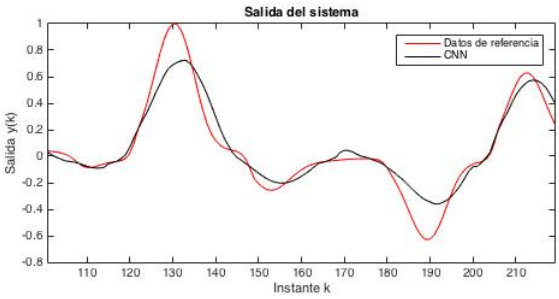


Figura 4.18: BP, 20 filtros y ruido.

La siguiente etapa consiste en utilizar mínimos cuadrados para actualizar la capa de salida una vez que se utilizó el algoritmo BP. Como era de esperarse, en la Figura 4.19 la utilización de mínimos cuadrados para la actualización de la capa de salida tiene mejores resultados que con BP unicamente. Al aplicar ruido en los datos, la señal se distorsiona un poco, pero mantiene la forma de la señal de referencia Figura 4.20.

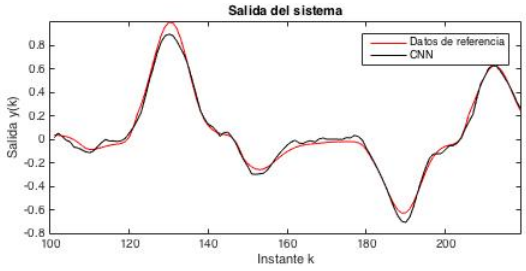


Figura 4.19: BP, mínimos cuadrados y 20 filtros.

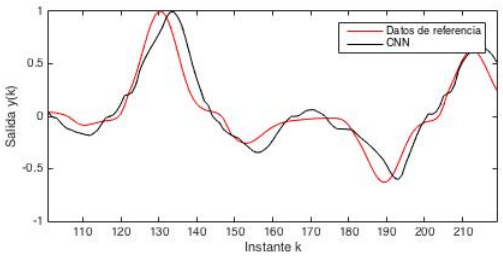


Figura 4.20: BP, mínimos cuadrados, 20 filtros y ruido.

La comparación de los errores para este sistema se presentan en la Figura 4.21 utilizando el algoritmo de BP solamente y ante la presencia de ruido. En la Figura 4.22 se realiza la comparación agregando un algoritmo de mínimos cuadrados a la etapa de aprendizaje.

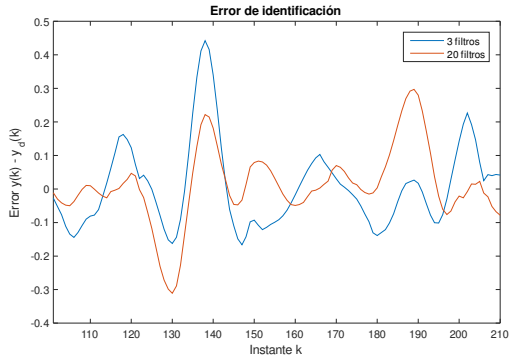


Figura 4.21: Error de identificación sistema no lineal.

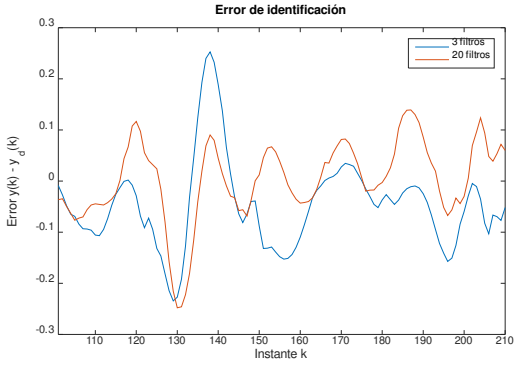


Figura 4.22: Error de identificación sistema no lineal con mínimos cuadrados.

En la Tabla 4.2 se presenta los errores medios cuadráticos para ambas simulaciones con 3 filtros y 20 filtros que se realizaron sobre los datos del sietema no lineal.

Tabla 4.2: Error medio cuadrático Sistema no lineal.

	Backpropagation	Backpropagation y mínimos cuadrados
3 Filtros		
Sin ruido	0.0070	$2.315\text{x}10^{-4}$
Ruido	0.0094	0.0052
20 filtros		
Sin ruido	0.0132	$7.3285\text{x}10^{-4}$
ruido	0.0062	0.0028

De la tabla anterior, se puede notar que la utilización de mínimos cuadrados reduce el error cuando no hay ruido en los datos, y de igual manera cuando hay ruido en el caso de tres filtros en las capas convolucionales. Para el caso de 20 filtros, ocurre lo mismo pero los valores obtenidos son mas grandes que en el caso de tres filtros.

Para esta planta, también se obtuvieron los mejores resultados con tres filtros en las capas convolucionales, como ya se ha mencionado. Ya sea utilizando solamente el algoritmo BP o utilzándolo con algoritmos de mínimos cuadrados. Aqui tampoco se puede concluir que a mayor cantidad filtros en las capas convolucionales se obtendrán mejores resultados. Los resultados también dependen de los valores de los parámetros de aprendizaje y de la inicialización de los filtros y pesos de la red.

Datos Wiener-Hammerstein

Para realizar estas simulaciones, se utilizarán 15 y 20 filtros para realizar una comparción extra de la cantidad de filtros y como afecta al desempeño de la red.

Simulaciones con 15 filtros

Primero se muestran las simulaciones con 15 filtros, utilizando BP y BP con mínimos cuadrados, con y sin ruido para ambos casos. En la Figura 4.23. se observa que la utilización de mas filtros mejora notoriamente la identificación, el error de generalización es menor aplicando más filtros en las capas convolucionales. Aunque con la presencia de ruido, la salida de la red neurnal convolucional si se ve afectada por él, Figura 4.24. El desempeño de la red es muy pobre comparandolo con el resultado obtenido con 3 filtros.

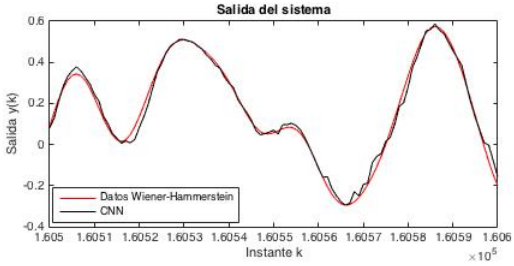


Figura 4.23: BP con 15 filtros.

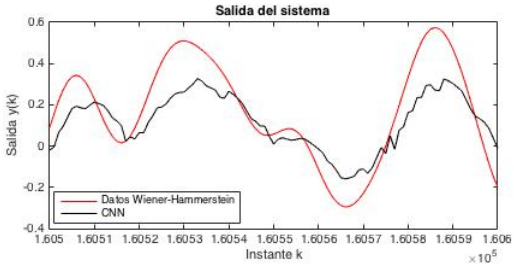


Figura 4.24: BP, 15 filtros y ruido.

Los siguientes resultados son obtenidos al aplicar mínimos cuadrados para actualizar los parámetros de la capa de salida. La figura 4.25 muestra que utilizar mínimos cuadrados en este caso no mejora la señal de salda de la red cuando no hay presencia de ruido. En cambio, cuando hay ruido en los datos Figura 4.26, se puede notar que tiene un mejor desempeño que el obtenido con menos filtros.

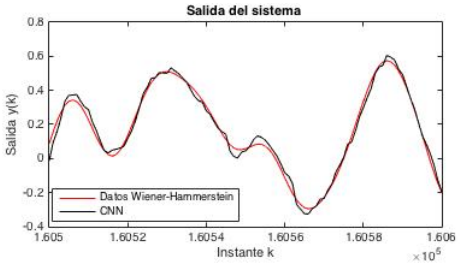


Figura 4.25: BP, mínimos cuadrados y 15 filtros.

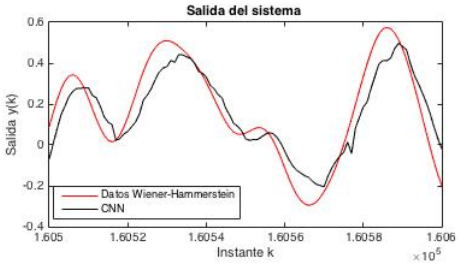


Figura 4.26: BP, mínimos cuadrados, 15 filtros y ruido.

Simulaciones con 20 filtros

Ahora se presenta los resultados de simulación utilizando 20 filtros en las capas convolucionales. Los resultados con BP se presentan en la Figura 4.27 y Figura 4.28. Como se observa, utilizar una mayor cantidad de filtros, empeora un poco los resultados cuando no hay ruido. Con la presencia de ruido, tiene un desempeño muy parecido a los anteriores resultados.

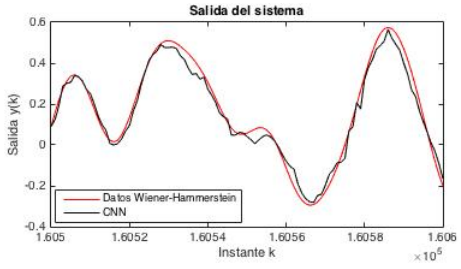


Figura 4.27: BP con 20 filtros.

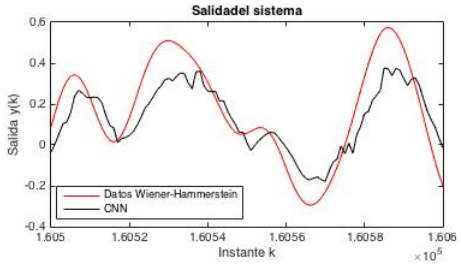


Figura 4.28: BP, 20 filtros y ruido.

Utilizando mínimos cuadrados, los resultados mejoran. Cuando no hay ruido presente, la Figura 4.29 muestra que la identificación del sistema es buena como en los casos anteriores. Ante la presencia de ruido, Figura 4.30, la identificación la realiza de mejor manera que solamente utilizando BP, pero no es muy buena.

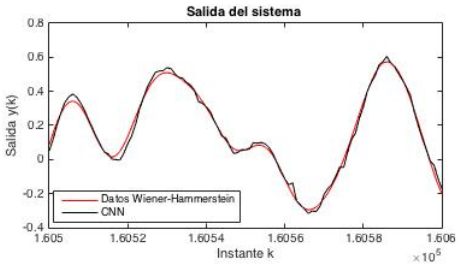


Figura 4.29: BP, mínimos cuadrados con 20 filtros.

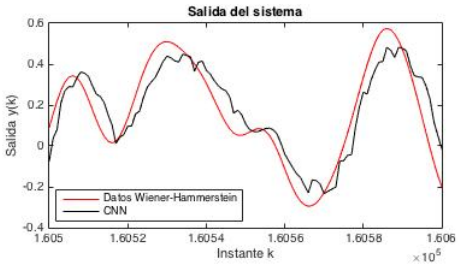


Figura 4.30: BP, mínimos cuadrados, 20 filtros y ruido.

Ahora se presenta la comparación del error de identifación utilizando BP con mínimos cuadrados para el aprendizaje y con ruido en los datos de entrenamiento. Se observa en la Figura 4.31 esta comparación, y se nota que la diferencia es muy pequeña entre ellas. La diferencia se puede apreciar mejor utilizando el error medio cuadrático que se presenta mas adelante.

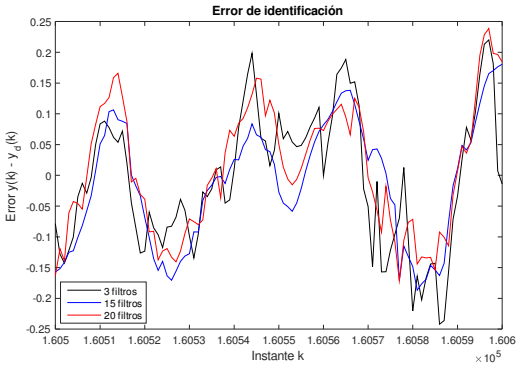


Figura 4.31: Error de identificación datos Wiener-Hammerstein.

Tabla 4.3: Error medio cuadrático Datos Wiener-Hammerstein.

	Backpropagation	Backpropagation y mínimos cuadrados
3 Filtros		
Sin ruido	-	0.011
Ruido	-	0.0095
15 filtros		
Sin ruido	$3.2868 \text{x} 10^{-4}$	$4.4914 \text{x} 10^{-4}$
Ruido	0.0062	0.0055
20 filtros		
Sin ruido	$9.9553 \text{x} 10^{-4}$	$3.0989 \text{x} 10^{-4}$
Ruido	0.0149	0.0058

En la Tabla 4.3 se puede apreciar la comparación de los errores medios cuadráticos para los datos Wiener-Hammerstein utilizando varias configuraciones de la red. Se observa que la diferencia de utilizar 3, 15 y 20 filtos es significativa, comparando cuando no hay ruido y se utiliza mínimos cuadrados, la red con tres filtros se queda con un error mucho mas alto que los otros dos, siendo mejor la red con 20 filtros.

En estas simulaciones, se obtuvieron mejores resultados utilizando 15 filtros, aunque los resultados cuando no hay ruido son muy parecidos, este se desempeña mejor cuando si lo hay. Como ya se ha mencionado, los resultados dependen de las inicializaciones de los parámetros de la red así como de los de aprendizaje.

Capítulo 5

Conclusiones

Este trabajo sirvió para mostrar ventajas de las redes neuronales convolucionales para la identificación de sistemas cuando hay ruido en los datos.

En el caso donde solamente se emplea la pseudoinversa generalizada de Moore-Penrose, comparando la CNN con un MLP, ambos resultados muy parecidos para los tres sistemas de prueba utilizados, sin embargo, cuando hay ruido es cuando la CNN supera al MLP. También debe considerarse que la CNN tiene mas capas ocultas pero menos unidades en la capa completamente conectada oculta que el MLP.

Para el aprendizaje supervisado, el algoritmo de backpropagrtion es una opción para entrenar redes neuronales convolucionales, solamente se requiere una pequeña modificación para considerar las capas convolucionales y de submuestreo, los resultados que se obtuvieron con esta metodología de aprendizaje son comparables con los resultados anteriores.

En cuanto a la estructura de la red, la cantidad de parámetros necesarios por capa (filtros) dependerá del sistema, ya que como se mostró en el capítulo anterior, aumentar la cantidad de filtros no mejoró el desempeño de la red en algunos sistemas.

Como se observó en este trabajo, las redes neuronales convolucioanles pueden emplearse para la identificación de sistemas y más aún, cuando hay ruido en los datos, dado las propiedades de la convolución y la forma en que esta operación se realiza dentro de la red neuronal.

Bibliografía

[1] E. Busseti, I. Osband, and S. Wong. Deep learning for time series modeling. *Technical report, Stanford University*, 2012.

[2] G. Horvath. Neural Networks in System Identification. Department of Measurement and Information Systems, Budapest University of Technology and Economics, 2007.

[3] M. Lin, Q. Chen, S. Yan. Network in network. in: ICLR, 2014.

[4] W. McCulloch, P. Walter. A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics. 5 (4): 115-133. 1943.

[5] J. L. McClelland, D. E. Rumelhart. Parallel Distributed Processing: Explorations in Microstructure of cognition. The MIT Press, 1986.

[6] B. W. Mel. Connectionist Robot Motion Planning. San Diego, CA: Academic Press, 1990.

[7] F. Rosenblatt. Principles of Neurodynamics. Spartan Books,Whasington DC, 1959.

[8] B. Krose, P. Smagt. An introduction to neural networks. The university of Amsterdam, Eighth edition, 1996.

[9] R. Manger, K. Puljić. Multilayer perceptrons and data compression. Computing and Informatics, Vol. 26, pp. 45-62, 2007.

[10] F. Rosenblatt. Principles of Neurodynamics: Perceptron and the Theory of Brain Mechanisms. Spartan Books, Whasington DC, 1961.

[11] D. E Rumerlhart, G. E Hinton, R. J Williams. Learning Internal Representations by error Propagation", David E. Rumelhart, James L McClelland, and the PDP research Group, Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: foundations, MIT Press, 1986.

[12] O. Nelles. Nonlinear System Identification: From classical Approaches to Neural Networks and Fuzzy models. Springer Science & Business Media, 2013.

[13] D.P. Bertsekas, J.N. Tsitsiklis. Neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. Athena Scientific p. 512, 1996.

[14] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. Advances in Neural Information Processing Systems (NIPS'06), pp.153-160, MIT Press, 2007.

[15] M. Minsky, S. Papert. Perceptrons: An Introduction to Computational Geometry. MIT Press. 1969.

[16] J. Sjöberg, Q. Sshang, L. Ljung, A. Benveniste, B. Deylon, P. Glorennec, H. Hjalmarsson, A. Judistsky. Nonlinear blackbox modeling in system identificaction; a unified overview. Automatica, Vol. 31, Issue 12, pp. 1691-1724, 1995.

[17] H. Khalil. Nonlinear Systems. Segunda edición, Prentice Hall, NJ, 1996.

[18] K. Sung, C. Riley. Applications of convolution in Image processing with MATLAB. University of Washington, 2013.

[19] S. Jayaraman, S. Esakkirajan, S. Veerakumar. Digital Image Processing. McGraw-Hill Education, 2011.

[20] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Comput. 1(4), 541-551. 1989.

[21] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. Gradient based learning applied to document recognition. *Proceeding of the IEEE*, 1998.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. Going Deeper with convolutions. in: CVPR, 2015.

[23] M. Matusugu, M. Katsuhiko, M. Yusuku, K. Yuji. Subject independent facial expression recognition with robust face detection using a convolutional neural network. Neural Networks 16(5): 555-559. 2003.

[24] D.H. Wiesel, T.N. Hubel. Receptive fields of single neurones in the cat's striate cortex. *J. Physio*l, 148:574-591, 1959.

[25] E.P. Simoncelli, D.J. Heeger. A model of neuronal responses in visual area. Vision research, 1998.

[26] *Journal of Machine Learning Research*, http://deeplearning.net/tutorial/lenet.html.

[27] K. Simonyan, A. Vedaldi, A. Zissserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. in: *Proc. ICLR*, 2014.

[28] Y. LeCun, R. Pfeiffer, Z. Chreter, F. Fogelman, L. Steels. Generalization and network design strategies. Connectionism in perspective, Eds. Elsevier Zurich, Switzerland, 1989.

[29] D. Hubel, T. Wiesel. Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London), 195, 215-243. 1968.

[30] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shua, T. Liu, X. Wang, G. Wang. Recent Advances in Convolutional Neural Networks. Interdisciplinary Graduate School, Nanyang Technological University, Singapore, 2017.

[31] D. Cireşan, U. Meier, J. Shcmihuber. Multi-column Deep neural networks for image calssifcation. *IEEE Conference on Computer Vision and Pattern Recognition*. New York, NY: Institute of Electrical and Electronics Engineers (IEEE): 3642-3649 , Junio 2012.

[32] K. Fukushima. Self-organization of a neural network which gives position-invariant response. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, pp.291-293, 1979.

[33] J. Bouvrie. Notes on Convolutional Neural Networks. MIT Press, 2006

[34] W. Zhang. Shift-invariant pattern recognition neural network and its optical architecture. *Proceedings of annual conference of the Japan Society of Aplied Physics*, 1988.

[35] P. Sermanet, Y. LeCun. Traffic Sign Recognition with Multi-scale Convolutional Networks. in: IJCNN, 2011.

[36] M.D. Zeiler. Hierarchical Convolutional Deep Learning in Computer Vision. Ph.D. Thesis, Nov. 8, 2013.

[37] D. Steinkraus, P. Simard. Using GPUs for Machine Learning Algorithms. 12th International Conference on Document Analysis and Recognition (ICDAR), pp. 115-119, 2005.

[38] K. Chellapilla, S. Puri, P. Simard. High Performance Convolutional Neural Networks for Document Processing. Lorette, Guy. Tenth International Workshop on Frontiers in Handwriting Recognition, 2006.

[39] G.E. Hinton, S. Osindero, Y.W. Teh. A fast learning algorithm for deep belief nets, Neural Computation. Neural Computation, 18 (7), 2006.

[40] J. Kruger, R. Westernmann. Linear Operators for GPU Implementations of numerical algorithms. Proceedings of SIGGRAPH, San Diego, pp. 908-916, 2013.

[41] A. Krizhevsky, I. Sutskever, G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. in: NIPS. 2012.

[42] M.D. Zeiler, R. Fergus. Visualizing and understanding convolutional networks. in: ECCV, 2014.

[43] R.E. Lopez. Matematicas, análisis de datos y python. http://relopezbriega.github.io/blog/2016/05/29/machine-learning-con-python-sobreajuste/

[44] D. Scherer, A. Muller, S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. in *International Conference on Artificial Neural Network*, 2010.

[45] K. He, X. Zhang, S. Ren, J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. in: ECCV, 2014.

[46] S. Singh, A. Gupta, A. Efros. Unsupervised discovery of mid-level discriminative patches. in: ECCV, 2012.

[47] K. Fukushima, S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. Pattern Recognition, vol. 15, pp 455-469, 1982.

[48] I. Goodfellow, D. Warde-Farley, M Mirza, A. Courville, Y. Bengio. Maxout networks. in: *ICML*, 2013.

[49] D.A. Clevert, T. Unterthiner, S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). in: ICLR, 2016.

[50] E. de la Rosa, W. Yu. Randomized Algorithms for nonlinear System Identification with Deep Learning Modification. Information Sciences, Vo. 364-365, pp. 197-212, 2016.

[51] M- Ranzato, F. Huang, Y. Boureau, Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. in *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007.

[52] Y. Bengio. Deep learning of representations: Looking forward. in: SLSP, 2013.

[53] H. Lee, R. Groose, R. Ranganath, and A. Y. Ng. Connvolutional deep belief networks for scalable unsupervised learning of hierarchical representations. in: *ICML*, 2009.

[54] J. C. A. Barata, M. S. The Moore–Penrose pseudoinverse: A tutorial review of the theory. Brazilian Journal of Physics, 42(1), 146-165, 2012.

[55] I. Gelink, Y.H. Pao B. Stochastic Choice of Basis Functions in Adaptive Function Approximation and the Functional-Link Net. *IEEE Transactions on Neural Networks* 6 (2) (1995) 1320-1329.

[56] G. Box, G. Jenkins, G. Reinsel. Time Series Analysis: Forecasting and Control. 4th Ed, Wiley, 2008.

[57] J. Schoukens, J. Suykens, L. Ljung. Wiener-Hammerstein benchmark. 5th IFAC Symposium on System Identification, Saint-Malo, France, 2009.

[58] K.S. Narendra, K. Parthasarathy. Gradient methods for optimization of dynamical systems containing neural networks. *IEEE Transactions on Neural Networks* 3 (2) (1991) 252-262.

[59] Y. LeCun, L. Bottou, G. Orr, K. Muller. Efficient BackProp. in: *Neural Networks: Tricks of the trade*. G. Orr andK. Mullet (eds.), Springer 1998.

[60] G. E. Hinton, N. Srivastava. Improving neural networks by preventing co-adaptation of feature detectors. in: CoRR, 2012.

[61] E. Polak. Computational Methods in Optimization. New York: Academic Press, 1971.

[62] D. O. Hebb. The Organization of Behavior. New York: Wiley, 1949.

[63] L. Bottou. Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT*, 2010.

[64] E: Thibodeau-Laufer, G. Alain, J. Yosinski. Deep generative stochastic networks trainable by backprop. in: JMLR, 2014.

[65] D. Kingma, J. Ba, Adam: A method for stochastic optimization. in: ICLR, 2015.

[66] P. Simard, D. Steinkraus, J. C. Platt Best practices for convolutional neural networks applied to visual document analysis. in: ICDAR, 2003.

[67] UFLDL tutorial. http://ufldl.stanford.edu.tutorial.

[68] G. Gwardys. Convolutional Neural Networks backpropagation: from intuition to derivation. https://grzegorzgwardys.wordpress.com/2016/04/22/8/, 2016.

[69] J. Johnson, A. Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. Notes of Stanfrod CS class.

[70] I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, http://www.deeplearningbook.org, 2016.

[71] K. de Brabater, P. Dreesen, P. Karmakers, K. Pelckmans, J. de Brabater, J.A.K Suykens, and B. de Moor. Fixed-size LS-SVM applied to the Wiener-Hammerstein benchmark. in: *Proceedings of the 15th IFAC Symposium on System Identification*, pp. 826-83, Sain-Malo, France, 2009.