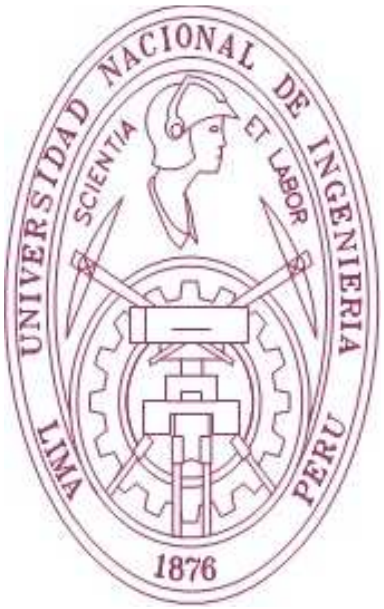
UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERÍA CIVIL



PREDICCIÓN DE LA RESPUESTA SÍSMICA DE MUROS DE ALBAÑILERÍA CONFINADA EMPLEANDO REDES NEURONALES ARTIFICIALES

TESIS

PARA OPTAR EL GRADO DE MAESTRO EN CIENCIAS CON MENCIÓN EN INGENIERÍA ESTRUCTURAL

ELABORADO POR

MELQUIADES DÁMASO DAMIÁN JARA

ASESOR

DR. CARLOS A. ZAVALA TOLEDO

LIMA – PERÚ

2013

PREDICCIÓN DE LA RESPUESTA SÍSMICA DE MUROS DE ALBAÑILERÍA CONFINADA EMPLEANDO REDES NEURONALES

Melquiades Dámaso Damián Jara

Presentado a la Sección de Posgrado de la Facultad de Ingeniería Civil en cumplimiento parcial de los requerimientos para el grado de:

MAESTRO EN CIENCIAS CON MENCIÓN EN INGENIERÍA ESTRUCTURAL

DE LA

UNIVERSIDAD NACIONAL DE INGENIERÍA

2013

Autor : Melquiades Dámaso Damián Jara

Recomendado : Dr. Carlos A. Zavala Toledo
Asesor de Tesis

Aceptado por : PhD. Jorge E. Alva Hurtado
Jefe de la Sección de Posgrado

Dedicatoria:

A Dios
Por haberme dado la vida y guiar mi desarrollo personal y profesional.

A mis Padres
Por su infinito amor, soporte constante y enseñanza de valores.

A mis hermanas
Por su cariño, apoyo y consejos.

Agradecimientos:

A Dr. Carlos Zavala
Por su confianza y apoyo constate durante el desarrollo de esta tesis.

A PhD. Hugo Scaletti y PhD. Javier Piqué
Por su valioso tiempo dedicado a la revisión de esta tesis.

A mis Amigos
Por haberme motivado a continuar trabajando en esta tesis.

Índice General

Resumen

En la presente tesis se emplea una Red Neuronal Artificial (RNA) para determinar la respuesta a una acción/desplazamiento lateral en muros de albañilería confinada. Para ello se preparan los datos de entrada al modelo, correspondiente al desplazamiento lateral y la carga vertical del muro confinado; así como la información de salida de la red, modelando el agrietamiento del muro como una secuencia de ceros y unos (0: no agrietado, 1: agrietado), además de la respuesta/fuerza lateral del muro que corresponde al patrón de agrietamiento.

Luego se diseña la arquitectura; una Red Neuronal feedforward con propagación del error hacia atrás (Backpropagation), un algoritmo de entrenamiento de tipo gradiente descendente con momento y aprendizaje variable, y una capa oculta con 33140 neuronas. La red se entrena para aprender los agrietamientos y las fuerzas laterales, logrando que reproduzca los datos aprendidos con aceptable precisión.

Summary

In this thesis, it is employed an Artificial Neural Network (ANN) to determine the response to a lateral force/displacement in confined masonry walls. For that, the model input data are prepared, corresponding to the lateral displacement and the confined wall vertical load as well as the network output, modeling the confined wall cracking as a sequence of zeros and ones (0: non cracked, 1: cracked), besides the wall lateral response/force that corresponds to the cracking pattern.

Afterwards, it is designed the architecture, a feedforward Neural Network with Backpropagation algorithm, a gradient descent training algorithm with variable moment and learning rate, and a hidden layer of 33140 neurons. The network is trained to learn the cracks and lateral forces, by making it reproduces the learnt data with acceptable precision.

Lista de Figuras

Lista de Tablas

Lista de Siglas y Símbolos

AASHTO	Asociación Americana de Normas para Carreteras Estatales y Transporte
ACI	Instituto Americano del Concreto
AISC	Instituto Americano de la Construcción en Acero
ASC	Escuelas Asociadas de Construcción
BP	BackPropagation
CISMID	Centro Peruano Japonés de Investigaciones Sísmicas y Mitigación de Desastres
EDPs	Ecuaciones Diferenciales Parciales
LRFD	Diseño por Factores de Carga y Resistencia
MATLAB	Laboratorio de Matrices
NIST	Instituto Nacional de Estándares y Tecnología
RNA	Red Neuronal Artificial
TIA	Tecnología de Inteligencia Artificial
UBC	Código Uniforme de Construcción
W	Matriz de Pesos
$\boldsymbol{b}$	Vector de umbrales
$\boldsymbol{e}$	Vector de errores
$\boldsymbol{x}$	Vector de entrada
$\boldsymbol{y}$	Vector de Salida
$\boldsymbol{z}$	Vector Objetivo

Introducción

Con el avance de las computadoras personales y la difusión de las tecnologías de Inteligencia Artificial (TIAs) tal como las Redes Neuronales Artificiales (RNAs), es posible predecir la respuesta de estructuras sometidas a desplazamientos laterales, modelar su comportamiento y predecir el agrietamiento. Además, en años recientes se ha demostrado que la utilización de RNAs puede ayudar a predecir mejor la respuesta de estas estructuras que los métodos convencionales.

En la presente tesis se propone una metodología de trabajo para entrenar las RNAs, con datos de muros de albañilería confinada ensayados en el laboratorio del CISMID, para replicar patrones de fisuras producidos por desplazamientos laterales.

El primer capítulo corresponde a los fundamentos del comportamiento de muros de albañilería confinada. Se explican los diferentes tipos de falla que pueden ocurrir en un muro confinado bajo las diversas condiciones de esfuerzo y los mecanismos de confinamiento que lo caracterizan.

El segundo capítulo muestra el estado del arte de las RNAs en la ingeniería estructural, sus aportes al diseño y análisis de estructuras, y recientes aplicaciones en los métodos experimentales en ingeniería civil.

El tercer capítulo corresponde a la selección de la arquitectura de la RNA a aplicar en la presente investigación. En este capítulo se detallan los diversos modelos y arquitecturas de redes neuronales más usadas, cuyas formulaciones son presentadas en formato matricial de acuerdo a las nuevas tendencias del software para cálculo científico.

El cuarto capítulo va dirigido a la aplicación del modelo y arquitectura de la red neuronal diseñada para la predicción de grietas en un muro de albañilería confinada sometida a desplazamientos laterales.

Finalmente, en el capítulo cinco se plasman las conclusiones y recomendaciones producto del presente estudio.

1. Comportamiento de Muros de Albañilería Confinada

La albañilería confinada es el sistema más empleado en la construcción de viviendas, oficinas y hoteles en las zonas urbanas del Perú.

Este sistema está constituido por un muro de albañilería simple enmarcado por elementos de concreto armado, vaciados con posterioridad a la construcción del muro. El marco de concreto armado, sirve principalmente para darle ductilidad al sistema. Adicionalmente funciona como elemento de arriostre cuando la albañilería se ve sujeta a acciones perpendiculares a su plano.

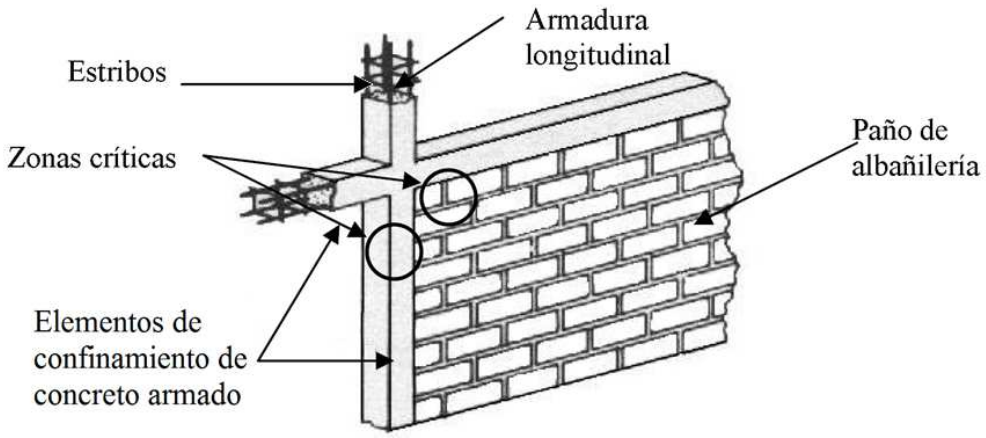


Figura 1.1: Partes de un muro de albañilería confinada

Como el objetivo de la tesis es simular el comportamiento de muros de albañilería confinada, sometidos a acciones/desplazamientos horizontales, en la sección 1.1 se explicará los tipos de falla que se presentan en muros de albañilería confinada.

1.1. Tipos de Falla en Muros de Albañilería Confinada

Existen varios tipos de fallas que pueden presentarse en este tipo de muros:

- Falla de deslizamiento por corte.- Este modo de falla se produce por un deslizamiento a lo largo de la junta horizontal del mortero debido a un problema en la adherencia por corte en la junta. Este deslizamiento produce un mecanismo de columna corta (ver figura 1.2)

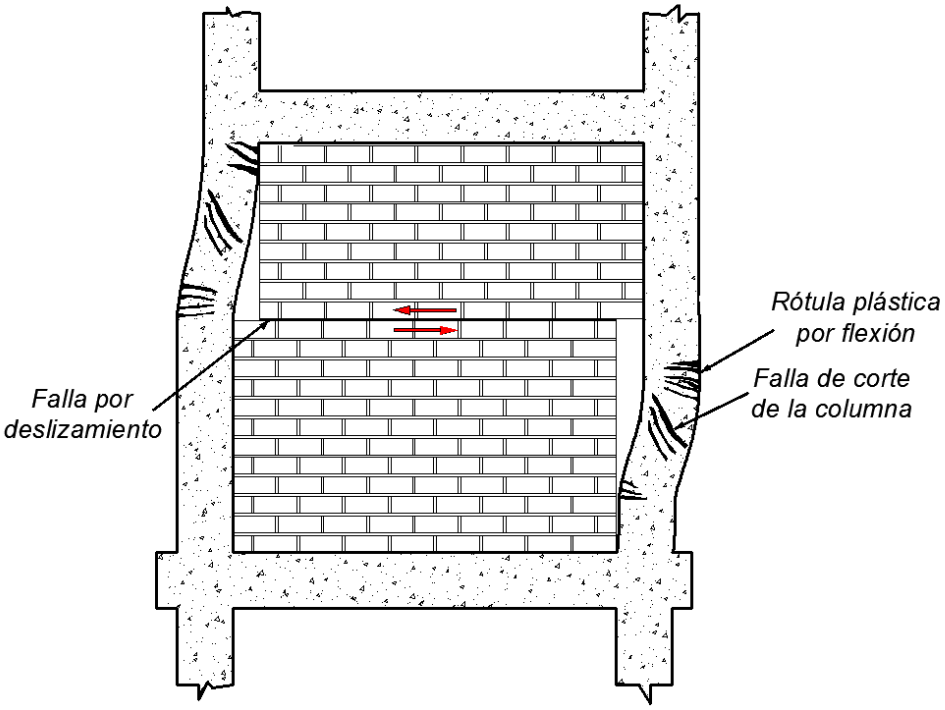


Figura 1.2: Falla de deslizamiento por corte

- Falla por corte.- Esta falla se caracteriza por un agrietamiento diagonal del paño de albañilería, como se muestra en la figura 1.3, y es consecuencia de las tensiones de tracción diagonal que se producen en el paño.

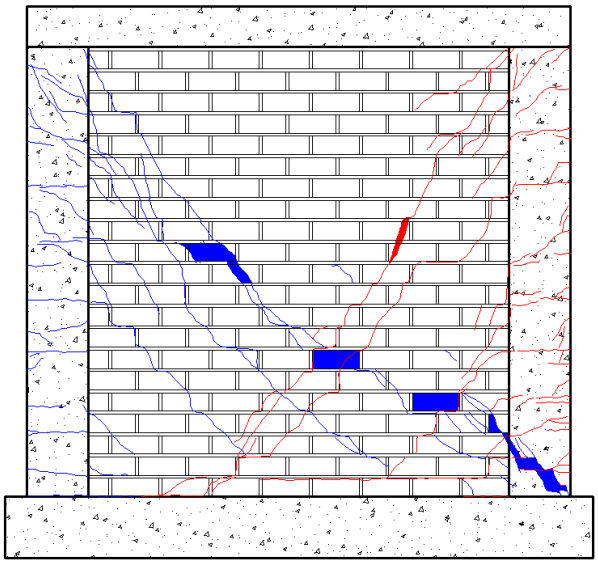


Figura 1.3: Falla por corte en el paño

- Falla por aplastamiento por compresión diagonal.- Esta falla es producto de la separación del paño de ladrillos de los elementos de confinamiento, generándose de esta manera un esfuerzo diagonal. Esto genera grandes esfuerzos de compresión en las esquinas, las que provocan falla por aplastamiento si el material es de baja calidad.

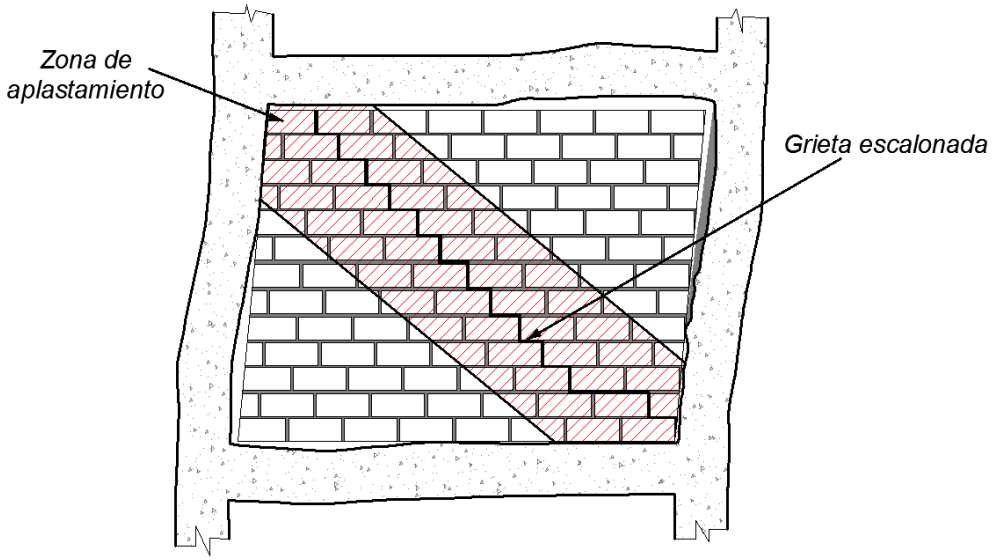


Figura 1.4: Falla por aplastamiento por compresión diagonal

- Falla por flexión.- Este tipo de falla se puede presentar en muros esbeltos, sobre todo cuando se generan grandes tracciones en las columnas, produciéndose de esta manera la fluencia de los aceros longitudinales y una falla por trituración de los talones flexocomprimidos.

2. Estado del Arte de las Redes Neuronales Artificiales

Las RNAs se emplean para resolver problemas de naturaleza lineal y no lineal en los diferentes campos de la Ingeniería Civil (estructuras, construcción, transporte, hidráulica, geotecnia y otros). Para ilustrar este punto, en este capítulo se presentan aplicaciones correspondientes a la Ingeniería Estructural.

2.1. Posicionamiento de Cargas Vivas en Pilares de Puentes

El análisis estructural de puentes y las cimentaciones de sus pilares [1] es un tema complejo. A diferencia de la mayoría de los diseños de edificación, los diseños de puentes vehiculares deben considerar la incertidumbre de la variabilidad de las cargas y de su aplicación. En particular, la aplicación de las cargas vivas vehiculares no es directa. En cualquier momento, los vehículos pueden atravesar el puente con velocidades y trayectorias desconocidas produciendo diferentes efectos. Afortunadamente, se ha realizado la documentación de la aplicación correcta de las cargas del vehículo a la superestructura del puente por la American Association of State, Highway, and Transportation Officials (AASHTO) así como por otras instituciones de investigación. Sin embargo, la aplicación subsecuente de estas cargas vivas a los pilares de apoyo del puente todavía no es bien comprendida y sólo es abordada muy brevemente por las especificaciones de diseño de la AASHTO-LRFD.

Una situación similar se presenta cuando se determinan los efectos de las fuerzas en la superestructura y la cimentación del pilar. La aplicación de las cargas vivas vehiculares a la superestructura para lograr los máximos efectos de fuerza no necesariamente produce los máximos efectos de fuerza en la cimentación del pilar. Es decir, una aplicación de carga viva totalmente diferente puede producir los máximos efectos de fuerza en la cimentación del pilar.

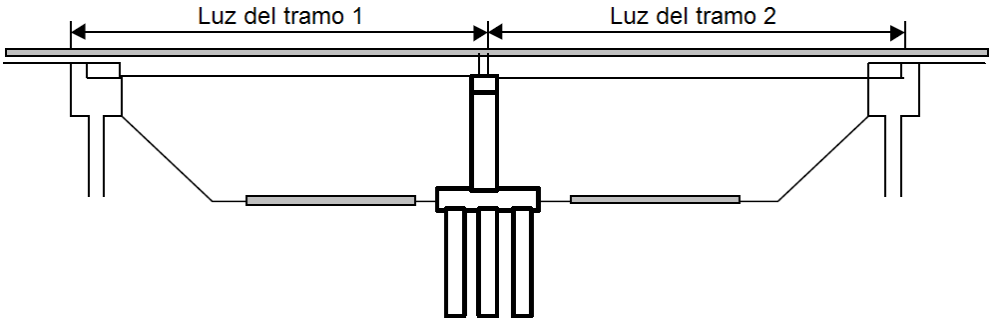


Figura 2.1: Sistema completo del puente

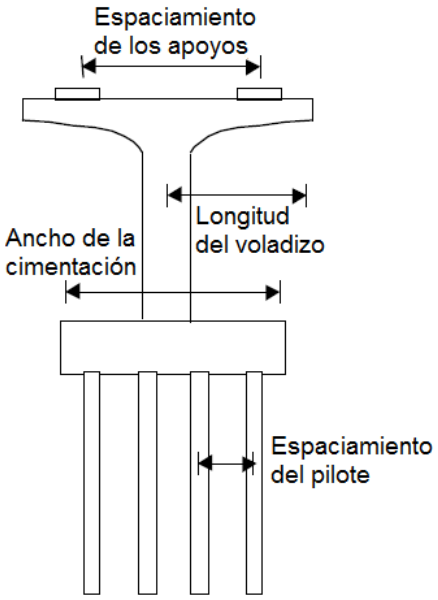


Figura 2.2: Configuración del pilar

Un estudio exhaustivo de las combinaciones de la posición de la carga viva a través del tablero del puente puede producir miles de posibles cargas de diseño. Las posiciones de carga viva más críticas pueden ser determinadas por consiguiente estudiando los resultados de dichas combinaciones.

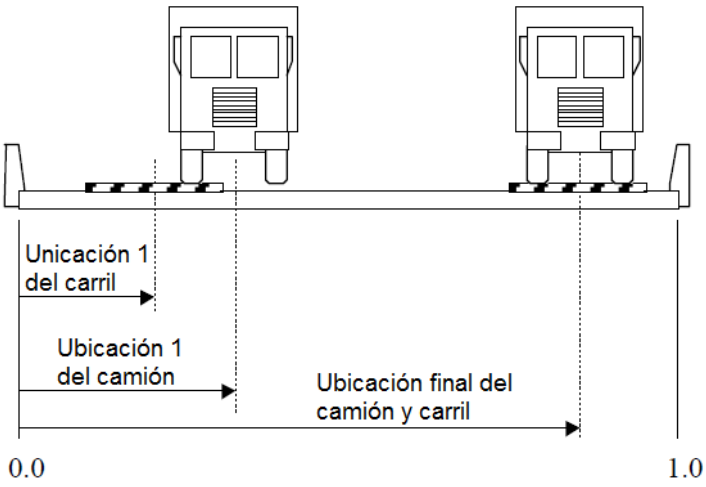


Figura 2.3: Ubicación normalizada de las carga vivas en la sección del puente

En la investigación de Williams [1], se emplean las RNAs para la aplicación de cargas vivas en puentes vehiculares. En particular, las RNAs son desarrolladas para predecir la posición de las cargas vivas en cada carril de tráfico que produzca los mayores

efectos en el apoyo interior del pilar. Para lograr la predicción de las posiciones de cargas vivas, se deben especificar los parámetros de entrada de la red. En esta aplicación particular de RNAs, los parámetros de entrada deberán describir la geometría de la superestructura del puente y del apoyo interior del pilar. Una exitosa aplicación de las RNAs debe producir una relación entre los parámetros geométricos de entrada y su salida correspondiente a la predicción de las posiciones de las cargas vivas. Como el problema involucra la interacción de varias variables de diseño, es improbable que la solución sea obvia por simple inspección visual. Esta deficiencia se resuelve por medio de la funcionalidad de las RNAs, las cuales son superiores a cualquiera de las técnicas de regresión estadística de reconocimiento de patrones.

Debido a que existen diferentes efectos máximos de fuerzas para cada componente estructural del pilar, se desarrollan diferentes RNAs para predecir las posiciones críticas de las cargas para cada efecto de fuerza. En la investigación de Williams [1] se desarrolló un total de 8 RNAs, que corresponden a los cuatro efectos máximos de fuerza identificados en los componentes estructurales para dos casos: primero para pilares de una columna y luego para pilares con varias columnas (ver figura 2.4 y 2.6). Estos cuatro efectos máximos de fuerza son identificados como: la combinación que produzca las máximas fuerzas en los pilotes y las columnas de los pilares así como el máximo esfuerzo de corte y de momento flector en la cimentación del pilar. Estos 4 efectos máximos de fuerza controlan el diseño de pilares de puente por carga viva.

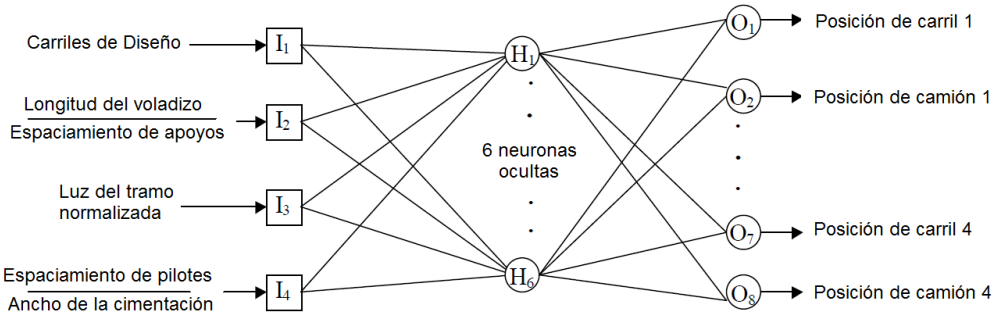


Figura 2.4: Arquitectura de red para pilares de una columna

El caso de pilares de varias columnas no es tan simple como el caso de una columna, ya que envuelve la interacción de varios parámetros de diseño. La incorporación de más parámetros es el resultado directo de la necesidad de tener un apoyo

más amplio para más carriles. La figura 2.5 muestra la configuración del pilar de varias columnas.

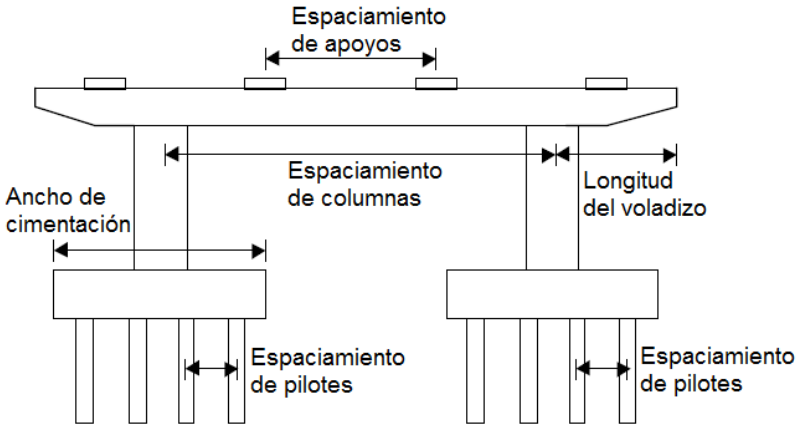


Figura 2.5: Configuración del pilar

Al igual que la red anterior, en este caso se predicen 4 pares de posiciones para lograr la máxima fuerza axial en el pilote y columna, así como el máximo momento flector y fuerza cortante en la cimentación del pilar (ver figura 2.6).

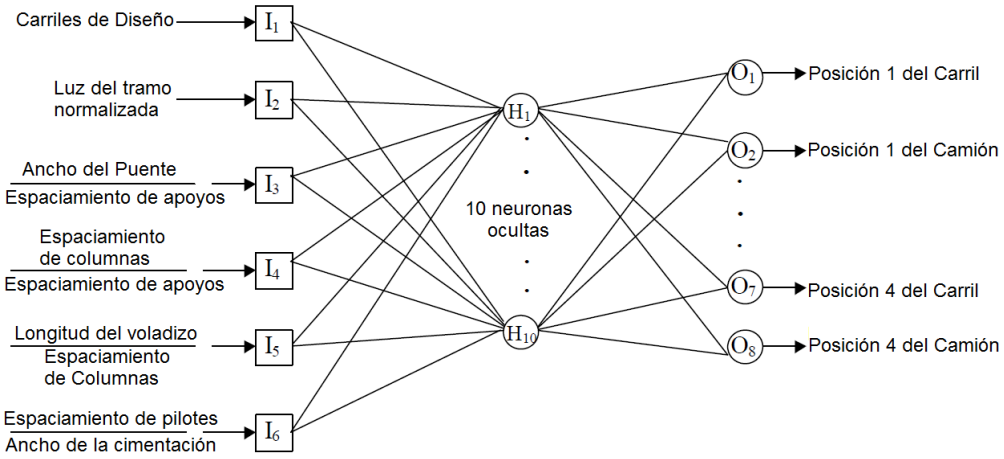


Figura 2.6: Arquitectura de red para pilares de varias columnas

Los resultados obtenidos luego de entrenar las redes fueron muy alentadores, aunque el autor añade que todavía es posible realizar mejoras en las redes.

2.2. Diseño de Mezclas de Concreto

El diseño de mezclas consiste en determinar las cantidades relativas de los materiales que forman parte de una estructura de concreto. La proporcionalidad se puede basar en datos obtenidos por experiencia práctica e investigaciones de ensayos que

pueden resultar de diversas dosificaciones para la obtención del concreto. Los materiales que forman parte del concreto son los agregados gruesos y finos, así como el agua y el cemento.

Existen diversos métodos de diseño seguidos a nivel mundial y son esencialmente similares, excepto que cada país posee su propio conjunto de tablas y gráficos para el cálculo de la densidad, agua requerida para la trabajabilidad y resistencia, basado en la disponibilidad local del tipo de agregado y cemento. Las variaciones son pequeñas en el proceso de seleccionar las proporciones de mezclas con los diferentes métodos de diseño. Algunos de los más comunes son:

- Método de diseño de mezclas del ACI
- Método de diseño de mezclas del USBR
- Método de diseño de mezclas Británico

En el estudio de Garg [2] se considera el ACI [3] como método de diseño de mezclas de concreto y se usa una RNA para predecir la proporción del agregado fino y grueso, dado como datos de entrada el esfuerzo a la compresión (f'c), el módulo de finura, relación de agregado grueso (10mm, 20mm), contenido de agua y relación de agua/cemento (ver figura 2.7).

En la selección de los datos de entrada se tuvo en cuenta que no necesariamente se deben ingresar un gran número de muestras, ya que podría sobreentrenar la red y eso no garantizaría obtener las mejores soluciones. Teniendo en cuenta esto, se usaron valores de contenido de agua, módulo de finura, relación de agregado grueso y esfuerzo de compresión correspondiente a los 28 días con relaciones de agua cemento de 0.42, 0.44, 0.46, 0.48 y 0.50.

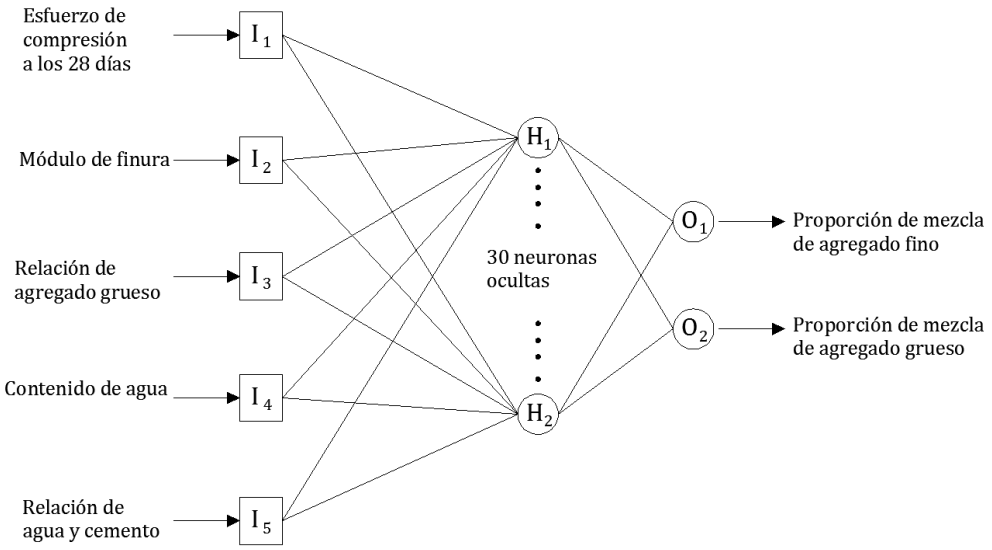


Figura 2.7: Arquitectura para la obtención de proporción de agregados

Los resultados obtenidos en las predicciones de la proporción de agregado grueso y fino se hicieron con las relaciones de agua cemento de 0.40 y 0.52 considerando concreto de resistencia a los 28 días y se compararon con los datos obtenidos experimentalmente, pudiéndose comprobar que existe un margen aceptable de error del 5%.

2.3. Presión Lateral en Muros de Contención

Las presiones laterales en muros de contención debido a cargas distribuidas en la superficie han sido investigadas por Yildiz [4] considerando el comportamiento no lineal de esfuerzo-deformación del suelo por análisis de elementos finitos. Los datos obtenidos a partir de análisis de elementos finitos fueron usados para entrenar redes neuronales con el fin de obtener una solución para evaluar el empuje lateral total y su punto de aplicación en muros de contención debido a una carga distribuida. La figura 2.8 muestra un esquema de la geometría del problema.

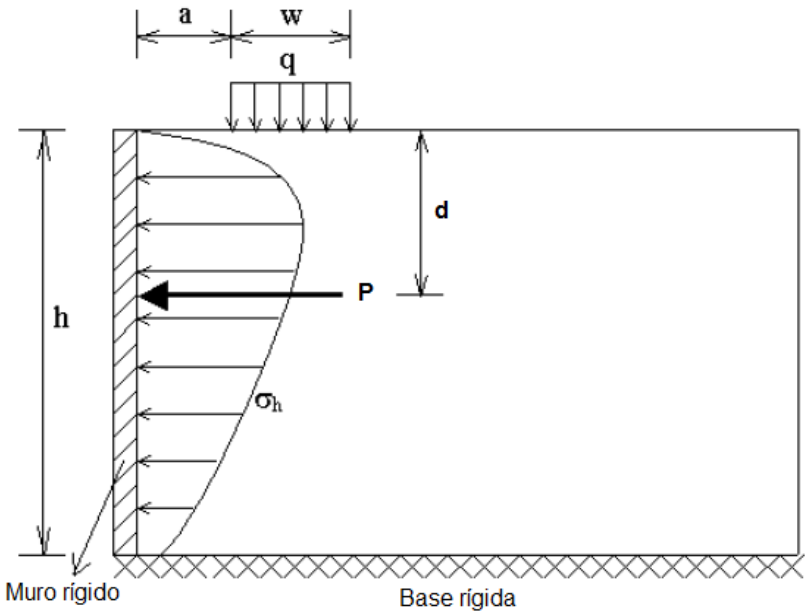


Figura 2.8: Geometría del problema

De la figura 2.8 en la investigación realizada por [4] se determinan los parámetros de entrada y salida de la red neuronal propuesta (ver tabla 2.1).

Entrada	Descripción
h	Altura del muro
a	Distancia a la carga distribuida
q	Magnitud de la carga distribuida
c	Cohesión
Φ	Ángulo de fricción
w	Ancho de la carga distribuida
Salida	Descripción
P	Empuje lateral en el muro debido a la carga distribuida
d	Distancia entre el punto de aplicación de P y la superficie del suelo

Tabla 2.1: Parámetros del muro de contención

Para la solución del problema propuesto, se planteó una red de dos capas, en cuya primera capa (oculta) se usó la función de transferencia sigmoidal y en la segunda capa (salida) la función de transferencia lineal. Los parámetros de entrada y de salida son mostrados en la figura 2.9. Esta primera red fue probada usando directamente los parámetros de la tabla 2.1, pero no se logró alcanzar una aceptable solución debido a que los resultados eran muy diferentes de los valores reales.

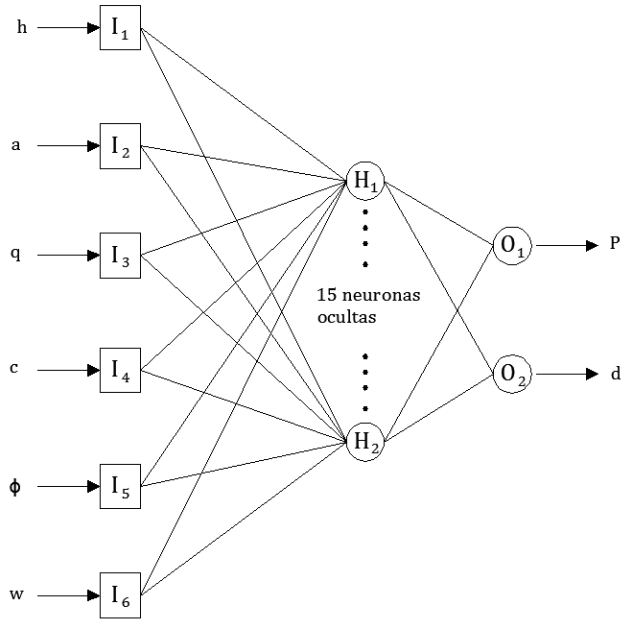


Figura 2.9: RNA inicial para el muro de contención

La idea del autor fue realizar algunas modificaciones en los parámetros, normalizando el parámetro a, w, d al dividirlos por el valor h. Se normaliza también el valor P dividiéndolo por el valor q (ver figura 2.10).

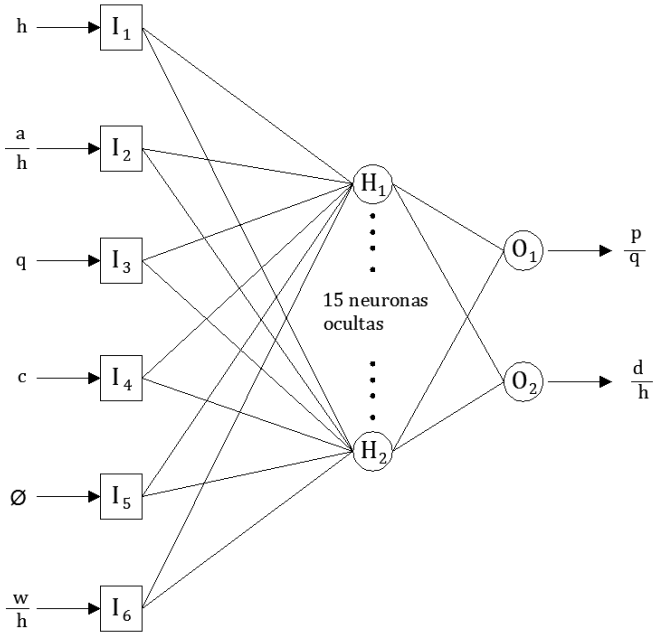


Figura 2.10: Arquitectura para la obtención de la fuerza lateral

Nuevamente se realizaron los entrenamientos y los resultados fueron mucho mejores que los obtenidos con la red 2.9, es decir, hablando específicamente de los parámetros. En este caso se obtuvo que el error máximo no superaba el 0.25%.

En la investigación realizada por [4] se pudo concluir que los parámetros podían influir en la solución del problema, ya que la normalización ayudó a brindar una mayor capacidad de generalización de la red.

2.4. Evaluación de Uniones Viga-Columna

Las uniones estructurales juegan un rol fundamental en la respuesta global de estructuras de acero. Se ha investigado el comportamiento real de una unión estructural a través de varias pruebas experimentales descritas por múltiples autores. El principal objetivo de estas pruebas fue determinar los parámetros físicos y geométricos que influyen en el comportamiento estructural de las uniones.

A partir de datos de ensayos, las uniones pueden ser clasificadas de acuerdo a sus momentos de flexión y a sus capacidades de rotación asociadas. Generalmente, las uniones son clasificadas como rígidas o flexibles. Sin embargo, esta clasificación no es precisa, ya que la mayoría de las uniones estructurales de acero no coinciden con ninguna de estas dos simplificaciones. A pesar de este hecho, el diseño tradicional de pórticos sin traslación usualmente supone juntas flexibles. Desafortunadamente, cuando se requiere diseñar pórticos con traslación, tienen que utilizarse uniones rígidas. Por otro lado, las uniones rígidas tienen los más altos costos de fabricación y dan lugar a un gran número de cuestionamientos acerca de su comportamiento estructural real. Para superar estas dificultades, las uniones semi-rígidas encajan como una solución natural, reduciendo el costo final y produciendo un comportamiento estructural más realista.

Lima y colaboradores [5] propusieron el uso de las redes neuronales para predecir la resistencia a la flexión y la rigidez inicial de las uniones semi-rígidas de viga-columna. Este problema de ingeniería estructural se caracteriza por la influencia de varios parámetros físicos y geométricos y por la gran dificultad de generar nuevos datos basados en pruebas experimentales. Esta fue la principal motivación para usar redes neuronales artificiales. En el estudio realizado por [5] se tomaron en cuenta 3 tipos de uniones: la unión con plancha empernada, unión soldada y unión empernada con ángulo (ver figuras 2.11, 2.12 y 2.13).

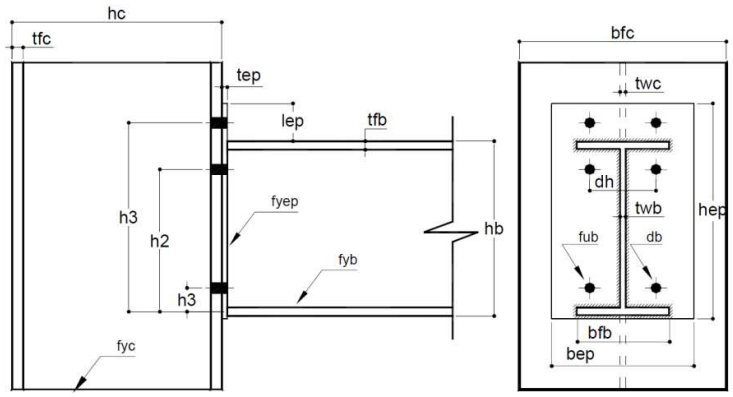


Figura 2.11 Unión con plancha empernada

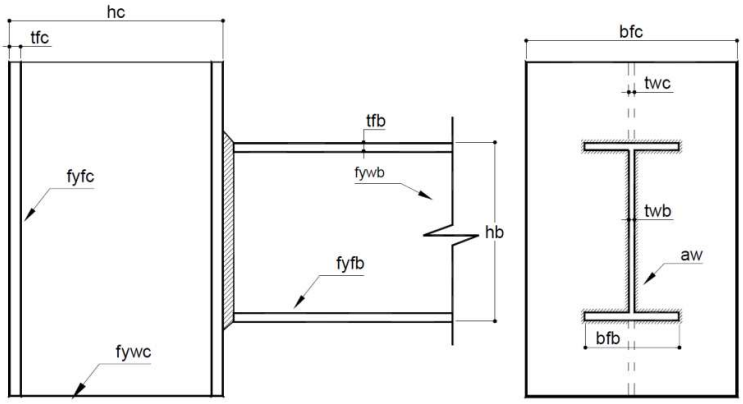


Figura 2.12: Unión soldada

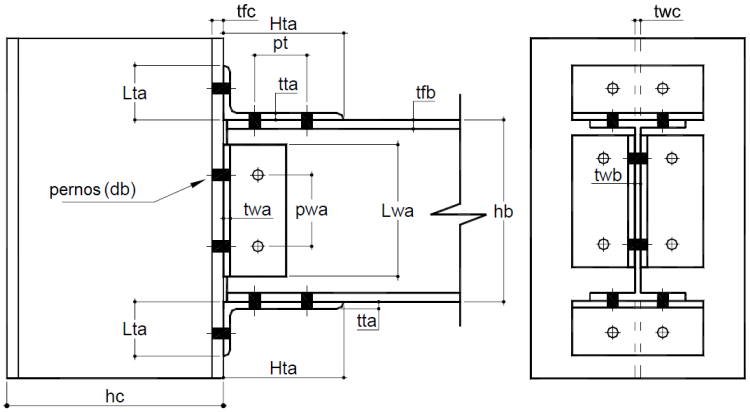


Figura 2.13: Unión con ángulo empernado

La tabla 2.2 muestra los parámetros usados para la creación de la red neuronal basada en las características geométricas y físicas de las uniones.

Entrada	Descripción
b_{ep}	Ancho de la plancha
b_{fb}	Ancho del ala de la viga
b_{fc}	Ancho del ala de la columna
d_b	Diámetro del perno
d_h	Distancia horizontal entre pernos
f_{ub}	Esfuerzo último del perno
f_{yb}	Esfuerzo de fluencia de la viga
f_{yc}	Esfuerzo de fluencia de la columna
f_{yep}	Esfuerzo de fluencia de la plancha
h_1	Altura de la primera fila de pernos
h_2	Altura de la segunda fila de pernos
h_3	Altura de la tercera fila de pernos
h_b	Altura de la viga
h_c	Altura de la columna
h_{ep}	Altura de la plancha
l_{ep}	Distancia desde la parte superior del ala de la viga hasta el borde libre de la plancha
t_{ep}	Espesor de la plancha
t_{fb}	Espesor del ala de la viga
t_{fc}	Espesor del ala de la columna
t_{wb}	Espesor del alma de la viga
t_{wc}	Espesor del alma de la columna
Salida	Descripción
$M_{j,Rd}$	Momento Resistente de la unión
$S_{j,ini}$	Rigidez inicial de la unión

Tabla 2.2: Parámetros de las uniones

Se crearon un total de 6 redes neuronales, 2 por cada tipo de conexión (una para el momento resistente y otra para la rigidez inicial). Es decir, las entradas fueron extraídas de la tabla 2.2, escogiendo las propiedades geométricas y físicas de acuerdo al tipo de conexión y luego fueron asociadas a una sola salida de las dos disponibles.

Los resultados de la red para todos los tipos de conexión fueron satisfactorios salvo por los resultados obtenidos para la rigidez inicial, lo cual mostró la necesidad de incorporar nuevos datos experimentales.

2.5. Diagnóstico de Fallas de Vigas Agrietadas en Voladizo

La dinámica de estructuras agrietadas ha sido estudiada intensamente en las dos últimas décadas. Las frecuencias naturales y formas de modo sufren variaciones debido a la presencia de grietas. Las desviaciones de las frecuencias naturales y de las formas de modo principalmente dependen de la ubicación y la intensidad de la grieta. La medida de las vibraciones flexionales de una viga en voladizo de sección rectangular con una fractura transversal superficial que se extiende uniformemente a través del ancho de la viga y sus resultados analíticos se usan para relacionar los modos de vibración medidos para la ubicación de la grieta y su profundidad.

En la investigación realizada por Cas y Parhi [6] se ha realizado la predicción de la ubicación de fracturas y sus profundidades desarrollando: análisis analítico (numérico), experimental y finalmente una técnica de RNA.

La red neuronal propuesta (figura 2.14) tiene 6 parámetros de entrada, dos parámetros de salida (tabla 2.3) y 8 capas ocultas. Este tipo de red es poco usual, ya que generalmente se usan de una a dos capas ocultas para representar la mayor complejidad a problemas de naturaleza no lineal.

Entrada	Descripción
fnf	Primera frecuencia natural
snf	Segunda frecuencia natural
tnf	Tercera frecuencia natural
fmd	Primera forma de modo
smd	Segunda forma de modo
tmd	Tercera forma de modo
Salida	Descripción
rcl	Ubicación relativa de la grieta
rcd	Profundidad relativa de la fractura

Tabla 2.3: Parámetros de la red para el diagnóstico de fallas

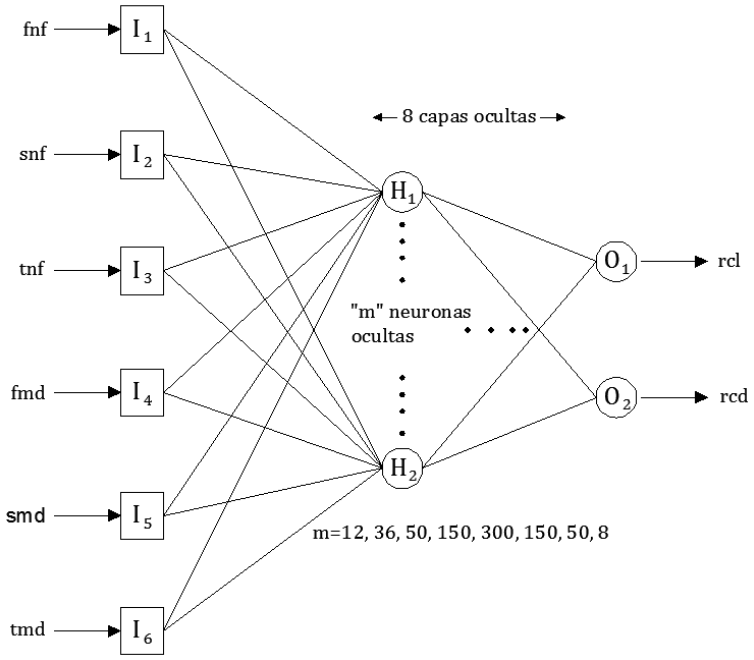


Figura 2.14: Red Neuronal para el diagnóstico de fallas

Se encontró una gran similitud entre los resultados obtenidos al predecir el resultado de 10 muestras para la viga en estado agrietado y no agrietado. Con ese fin, se usaron 800 patrones de entrenamiento para la red neuronal.

Finalmente, cabe mencionar que la detección exitosa de la fractura y su intensidad en la viga en voladizo demuestra que la técnica desarrollada en el estudio realizado por [6] puede ser usado de manera eficiente y efectiva en la detección de fracturas en diferentes estructuras del tipo viga y puede extenderse a los diferentes tipos de estructuras que se encuentren sometidas a vibraciones.

2.6. Capacidad Sísmica de Elementos Estructurales

La determinación cuantitativa de la resistencia y la capacidad de desempeño de elementos estructurales es de vital importancia para la evaluación de la vulnerabilidad de edificaciones existentes, así como para el diseño efectivo de nuevas edificaciones resistentes a terremotos.

Stanic y colaboradores [7] fueron motivados debido a la gran incertidumbre en la estimación de la capacidad sísmica de muros y columnas. A pesar de los extensos estudios experimentales todavía hay una falta de comprensión en la dependencia del

comportamiento observado de variables tales como la forma de la sección de corte, cantidad de refuerzo vertical y horizontal, compresión axial e historia de cargas.

La evaluación de la capacidad de desempeño de muros y columnas basada en las propiedades de esfuerzo-deformación no representa fácilmente el comportamiento real de muchos parámetros desconocidos (deslizamiento del refuerzo en los nudos, aplastamiento y descascarado del concreto). El enfoque empírico parece ser más apropiado, ya que están incluidos muchos parámetros impredecibles.

El principal objetivo de este trabajo es hacer una contribución a la determinación cuantitativa de la capacidad de desempeño de elementos estructurales verticales específicos, que posean una muy buena resistencia a la carga lateral. Su desempeño, expresado en términos de capacidad de resistencia al corte y deformación es de vital importancia para la evaluación del desempeño sísmico de estructuras existentes así como para el diseño de las nuevas edificaciones de concreto armado resistentes a terremotos.

En este caso se explica la red neuronal presentada para la predicción del rendimiento sísmico de una columna. La base de datos usada en este estudio es obtenida del PEER Structural Performance Database (http://www.ce.washington.edu/~peera1/). Esta base de datos es construida en el trabajo previo del National Institute Standards and Technology (NIST). En el momento del estudio, esta base de datos contó con 107 pruebas de columnas rectangulares y 92 pruebas de columnas circulares (zunchadas) de concreto reforzados, pero para este estudio se usaron 91 columnas rectangulares.

La tabla 2.4 muestra los parámetros de entrada y salida requeridas para esta red neuronal.

Entrada	Descripción
f'c	Resistencia a la compresión del concreto
P	Carga axial
B	Ancho de columna
H	Profundidad de columna
L	Longitud equivalente del volado
$\emptyset_L$	Diámetro del refuerzo longitudinal
n_L	Número de barras del refuerzo longitudinal
a	Recubrimiento
rhol	Cuantía del refuerzo longitudinal
fyl	Resistencia de fluencia del acero longitudinal

$\emptyset_T$	Diámetro de la barra del refuerzo transversal
rhot	Cuantía del refuerzo transversal
fyt	Resistencia de fluencia del acero transversal
Salida	Descripción
Fy	Fuerza de corte de fluencia
dy	Desplazamiento de fluencia
Fu	Fuerza de corte última
du	Desplazamiento último
Tipo de falla	Flexión 1; Corte 2; Flexión y Corte 3

Tabla 2.4: Parámetros de capacidad sísmica

Tal como se muestra en la tabla 2.4 el objetivo aquí es obtener los valores de Fy, dy, Fu, du y el tipo de falla. Se creó una red neuronal para cada salida de la tabla 2.4, es decir, se obtuvieron 5 redes neuronales cada una con 13 datos de entrada. La calidad de cada una de las redes neuronales fue probada con datos que no estuvieron en la base de datos original. Los resultados de las redes y los datos experimentales estuvieron razonablemente cerca, cosa que se puede verificar al referirse al estudio de Stanic et al. [7].

3. Selección de la Arquitectura de la Red Neuronal a Emplear

Para seleccionar la arquitectura adecuada se debe primero revisar los modelos, arquitecturas, así como las reglas de aprendizaje que más se ajusten al problema, lo cual es motivo de la presente tesis. Primero se explica lo relacionado a las redes monocapa que son los pilares de las diversas arquitecturas que se vienen empleando en el campo de investigación experimental y que están dando muy buenos resultados. Luego, se revisa la red multicapa más ampliamente aceptada que es la red de propagación hacia atrás (backpropagation), la cual finalmente se presentará con una formulación netamente matricial y será acondicionada para resolver el problema propuesto.

3.1. Redes Monocapa

En esta sección se cubre las redes que son de interés histórico, aunque actualmente se utilicen menos que las redes multicapa otorgan una base sólida para la comprensión de estas últimas. Además, cabe mencionar el aporte de la notación matricial a cada una de las redes con sus respectivos códigos en MATLAB [8].

La red perceptrón es una red de una sola capa cuyos pesos y umbrales pueden ser entrenados para producir un correcto vector objetivo cuando se presenta con el correspondiente vector de entrada. La regla del perceptrón fue el primer algoritmo de entrenamiento desarrollado para las redes neuronales. El libro original del perceptrón es presentado por Rosenblatt [9].

La red de Hopfield es usada para almacenar uno o más vectores de equilibrio. Estos vectores de equilibrio pueden ser vistos como los recuerdos que la red hará volver cuando ésta se provea con vectores similares que actúen como una señal a la memoria de la red.

3.1.1. El Perceptrón Simple

El perceptrón es la red de aprendizaje más sencilla para realizar clasificaciones de patrones a través de un hiperplano.

3.1.1.1. Modelo

A continuación en la figura 3.1 se muestra una neurona de perceptrón, la cual usa la función escalón unitario como función de transferencia.

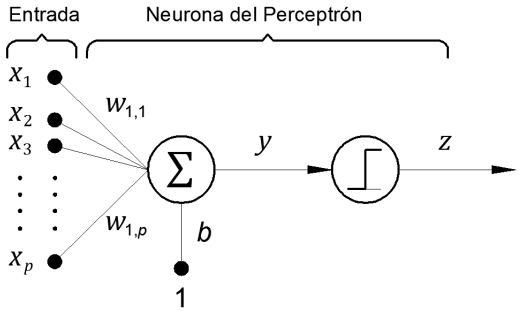


Figura 3.1: Esquema del perceptrón simple binario

Donde:

p = es el número de elementos del vector de entrada.

Se pondera cada una de las entradas externas con un peso w_i apropiado, y se envía la suma de las entradas ponderadas a la función de transferencia escalón unitario, incluyendo aquella que posee un valor de entrada 1 ponderada por el umbral. La figura 3.2 muestra la función escalón unitario de acuerdo a las convenciones adoptadas en la presente tesis:

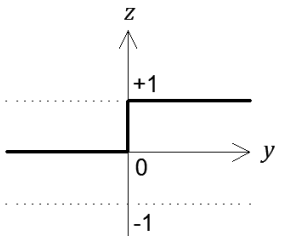


Figura 3.2: Función escalón unitario

$$y = \sum_{i=1}^{p} x_i w_{1,i} + \mathrm{b} = W\boldsymbol{x} + b \tag{3.1}$$

Donde $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \in \mathbb{R}^p$ es el vector de las entradas, $y \in \mathbb{R}$ es la salida,

$W = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,p} \end{bmatrix}^T \in \mathbb{R}^p$ y $b \in \mathbb{R}$.

x: es el vector columna que contiene las entradas.

y: es un escalar que representa la salida de la neurona.

w: es el vector de pesos de una fila y p columnas.

b: es el umbral de la entrada unitaria.

La función de transferencia escalón unitario brinda a un perceptrón la habilidad de clasificar los vectores de entrada dividiendo el espacio de entrada en dos regiones. Específicamente, las salidas serán 0 si la entrada y de la red es menor que 0, o 1 si la entrada y de la red es mayor o igual a 0. La figura 3.3 muestra el espacio de entrada de una neurona escalón unitario de 2 entradas con los pesos $w_1 = -1$, $w_2 = 1$ y un umbral $b = 1$.

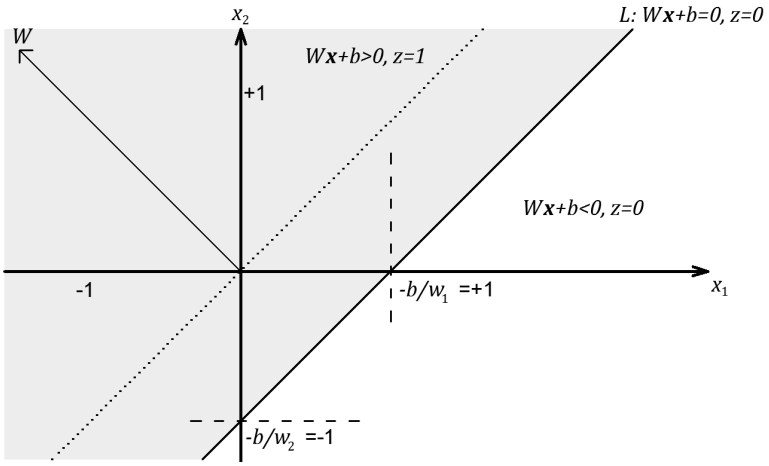


Figura 3.3: Regiones con valores binarios determinada por la recta L

Donde:

$$w_1 = -1$$

$$w_2 = +1$$

$$b = +1$$

La línea L define dos regiones de clasificación de contorno de decisión en $W\boldsymbol{x} + b = 0$. Esta línea es perpendicular al vector de pesos W y es afectada de acuerdo al umbral b. Los vectores de entrada encima y a la izquierda de la línea L resultan en un ingreso a la red mayor que 0 y, por lo tanto, causan que la neurona escalón unitario produzca una salida de 1. Los vectores de entrada debajo y a la derecha de la línea L causan una salida de la neurona de 0. Es posible escoger los valores de los pesos y el

umbral para orientar y desplazar la línea de división tal que se clasifique el espacio de entrada como se desee.

Las neuronas escalón unitario sin un umbral siempre tienen una línea de clasificación que pasa a través del origen. Añadir un umbral permite a la neurona resolver los problemas donde los dos conjuntos de vectores de entrada no están ubicados en diferentes lados del origen, como se muestra en la figura 3.3.

3.1.1.2. Arquitectura

La red perceptrón consiste de una sola capa de m neuronas conectadas a p entradas a través de un conjunto de pesos $w_{i,j}$, como lo mostrado en la figura 3.4. Al igual que antes, los índices de la red i y j indicarán que $w_{i,j}$ es la fuerza de conexión desde la entrada j-ésima a la i-ésima neurona.

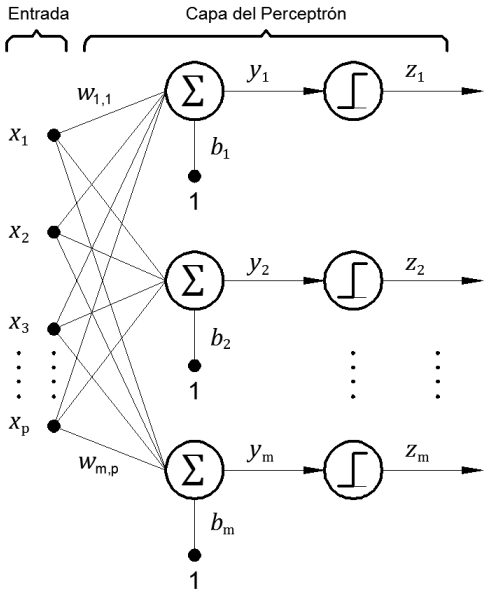


Figura 3.4: Red de una capa del perceptrón binario

La regla de aprendizaje del perceptrón descrita brevemente es capaz de entrenar solamente una capa. En consecuencia aquí sólo se considerarán las redes de una sola capa. Esta restricción establece las limitaciones que puede llevar a cabo un perceptrón en el cálculo.

Representación de la arquitectura del perceptrón de acuerdo al Toolbox de MATLAB:

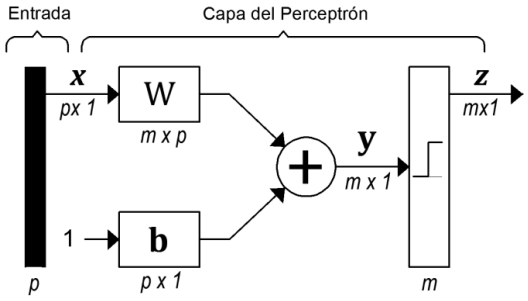


Figura 3.5: Diagrama funcional de la red perceptrón binario en Matlab

Los perceptrones son entrenados con ejemplos deseados de comportamiento. El comportamiento deseado puede ser totalizado por un conjunto de pares de entrada y salidas.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

$$\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\boldsymbol{z} = f(\boldsymbol{y}) = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

$$\boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{bmatrix}$$

Cada entrada deberá ser ponderada por sus respectivos pesos, los cuales pueden ser representados en su forma matricial:

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,p} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,p} \end{bmatrix}$$

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

El objetivo es reducir el error $\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{z}$, lo cual conduce a un problema de optimización en el cual se deberá minimizar la siguiente expresión:

$$\frac{1}{2}\|\boldsymbol{t} - f(W\boldsymbol{x} + \boldsymbol{b})\|^2 \tag{3.2}$$

El factor 1/2 es añadido por conveniencia notacional y no cambiará el proceso de minimización. La regla de aprendizaje del perceptrón calcula los cambios deseados a los pesos y umbrales del perceptrón, dando un vector de entrada p y el error asociado e. El vector objetivo t debe contener valores de 0 o 1, porque los perceptrones sólo pueden arrojar esos valores.

$$W(k+1) = W(k) + \Delta W(k) \tag{3.3}$$
$$\boldsymbol{b}(k+1) = \boldsymbol{b}(k) + \alpha \cdot \boldsymbol{e}(k) \tag{3.4}$$

Donde:

$$\Delta W(k) = \alpha \cdot \boldsymbol{e}(k)\boldsymbol{x^T}$$

$$\boldsymbol{e}(k) = \boldsymbol{t} - f(W(k) \cdot \boldsymbol{x} + \boldsymbol{b}(k)) \tag{3.5}$$

y α es el factor de aprendizaje.

Cada vez que se ajusten los pesos, el perceptrón poseerá una mejor oportunidad de producir las salidas correctas. Se brinda la regla del perceptrón para converger en una solución en un número finito de iteraciones, en caso de existir una.

Si no se usara el umbral, el algoritmo de aprendizaje trabajaría para encontrar una solución alterando sólo los pesos de la matriz *W* que apunte hacia los vectores de entrada para ser clasificados como 1 y alejados de los vectores a ser clasificados como 0 (no se alteraría el vector $\boldsymbol{b}$). Esto resultará en un contorno de decisión que sea ortogonal a *W* y que propiamente clasifique los vectores de entrada.

3.1.2. Red Lineal

Este tipo de red es usada frecuentemente como salidas en las diversas arquitecturas de aprendizaje de las redes neuronales, porque permite obtener resultados sin necesidad de realizar una desnormalización de los datos.

3.1.2.1. Modelo

En la figura 2.6 se muestra una neurona lineal con *p* entradas.

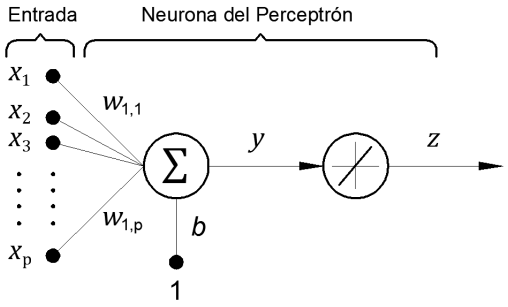


Figura 3.6: Perceptrón simple lineal

Esta red posee la misma estructura básica que el perceptrón. La única diferencia es que la neurona usa una función de transferencia lineal (ver figura 3.7).

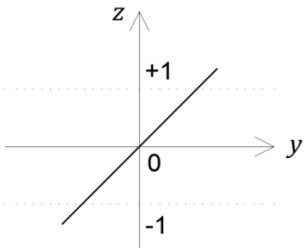


Figura 3.7: Función lineal

Esta neurona puede ser entrenada para aprender una función afín a sus entradas, o encontrar una aproximación lineal a una función no lineal. Una red de este tipo no puede, de hecho, ser creada para llevar cabo un cálculo no lineal.

Así como el perceptrón, la red lineal posee un contorno de decisión que es determinado por los vectores de entrada para lo cual la entrada de la red *y* es cero. Para *y* = 0 la ecuación $W\boldsymbol{x} + b = 0$ especificará un límite de decisión, como se muestra a continuación:

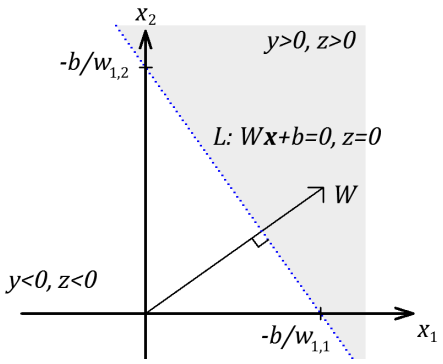


Figura 3.8: Regiones divididas por la recta L para salidas negativas y positivas

Los vectores de entrada en la parte superior derecha del área gris conducen a una salida mayor que 0. Los vectores de entrada en la parte inferior izquierda del área blanca conducen a una salida menor que 0. Por lo tanto, la red lineal puede ser usada para clasificar objetos dentro de dos categorías. Sin embargo, sólo es posible clasificar de esta manera si los objetos son linealmente separables. En consecuencia, la red lineal posee la misma limitación que el perceptrón.

3.1.2.2. Arquitectura

La red lineal mostrada en la figura 3.9 posee una capa de *m* neuronas conectada a *p* entradas a través de una matriz de pesos *W*.

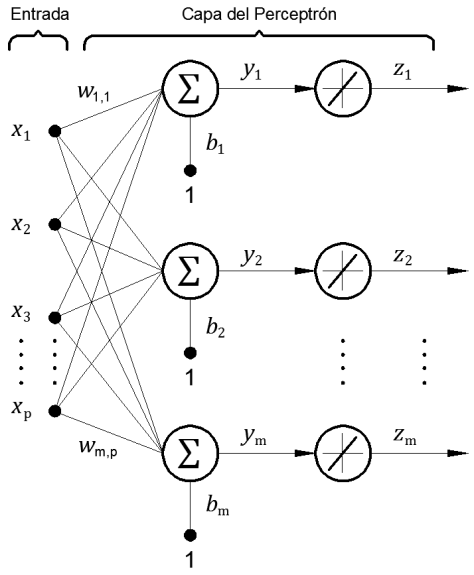


Figura 3.9: Red de una capa del perceptrón lineal

Representación de la arquitectura de la red lineal de acuerdo al Toolbox de MATLAB:

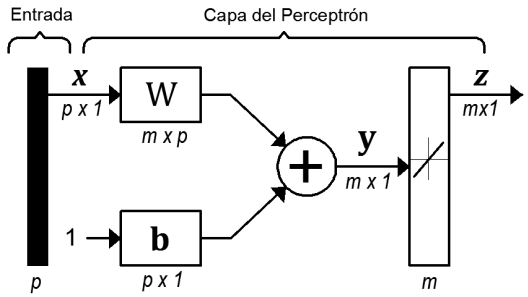


Figura 3.10: Diagrama funcional de la red perceptrón lineal en Matlab

Note que la figura 3.10 define el vector z de salida de longitud m.

Se muestra una red lineal de una sola capa. Sin embargo, esta red es justo tan capaz como las redes lineales multicapa. Para toda red lineal multicapa, existe una red lineal de una sola capa equivalente.

3.1.3. Hopfield

Esta red es muy parecida al perceptrón, pero presenta una característica adicional en las neuronas de la capa media, y es que éstas presentan conexiones de salida hacia otras neuronas de la capa media.

3.1.3.1. Modelo

El objetivo es diseñar una red que almacene un conjunto específico de puntos de equilibrio, tal que cuando se provea de una condición inicial, la red eventualmente se detenga en un punto de diseño. La red será recursiva, ya que la salida es retroalimentada a la entrada, una vez que la red está en operación. Con optimismo, la salida de la red se establecerá en uno de los puntos de diseño originales.

El método de diseño presentado no es perfecto ya que la red diseñada puede tener puntos de equilibrio no deseados que sean falsos, además de los deseados. Sin embargo, el número producido de estos puntos no deseados es tan pequeño como sea posible por el método de diseño. Además, cabe mencionar que el dominio de atracción de los puntos de equilibrio diseñados es lo más grande posible.

El método de diseño está basado en un sistema de ecuaciones diferenciales lineales ordinarias de primer orden que son definidas en un hipercubo cerrado del espacio de estado. Las soluciones existen en el contorno del hipercubo. Estos sistemas poseen la estructura básica del modelo de Hopfield, pero son más fáciles de entender y diseñar que el modelo de Hopfield.

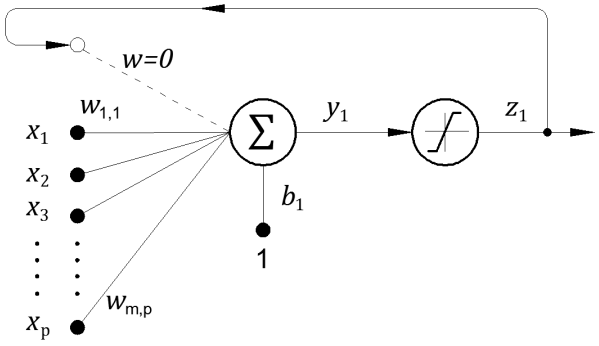


Figura 3.11: Modelo recurrente de una neurona

3.1.3.2. Arquitectura

La arquitectura de la red de Hopfield se muestra a continuación:

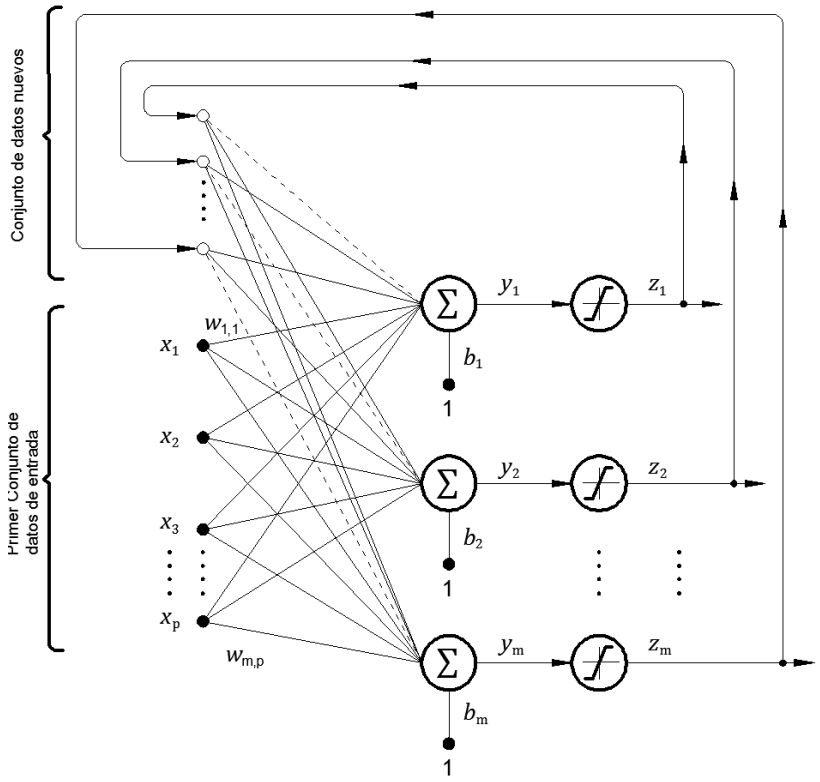


Figura 3.12: Red recurrente de Hopfield

La representación de la arquitectura en MATLAB es:

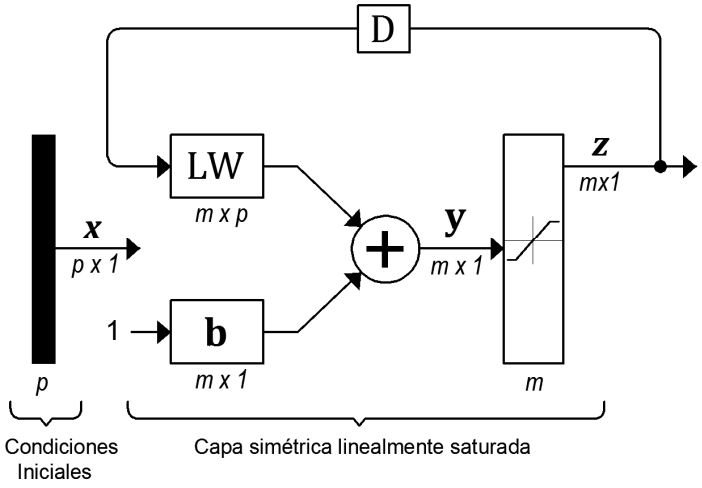


Figura 3.13: Diagrama funcional de la red Hopfield en Matlab

Como se nota, la entrada p para esta red simplemente suministra las condiciones iniciales. La red de Hopfield usa la función de transferencia lineal saturada.

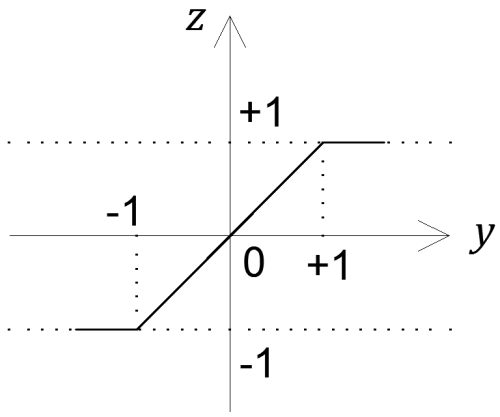


Figura 3.14: Función lineal saturada

Para las entradas menores que -1 la función lineal saturada producirá -1. Para las entradas en el rango de -1 a +1 ésta simplemente retornará el valor de entrada. Para las entradas mayores que +1 se producirá +1.

Esta red podrá ser probada con uno o más vectores de entrada que sean presentadas como las condiciones iniciales de la red. Después de que se den las condiciones iniciales, la red producirá una salida que sea por consiguiente retroalimentada para convertirse en la entrada. Este proceso será repetido una y otra vez hasta que la salida se estabilice. Con optimismo otra vez, cada vector de salida eventualmente convergerá a uno de los vectores del punto de equilibrio de diseño que sea más cercano a la entrada que lo provocó.

3.2. Redes Multicapa: Backpropagation

3.2.1. Modelos de Neurona

Las redes multicapa frecuentemente usan la función de transferencia sigmoidal (o logística) en las capas ocultas y una función lineal en la capa de salida. Es posible aplicar la función logística como capa de salida, por ejemplo, si se requiere que los resultados se encuentren en valores porcentuales; de otro modo, se usará la función lineal para que los resultados se presenten con sus magnitudes reales (kg, m3, Newtons, N/m2, etc.)

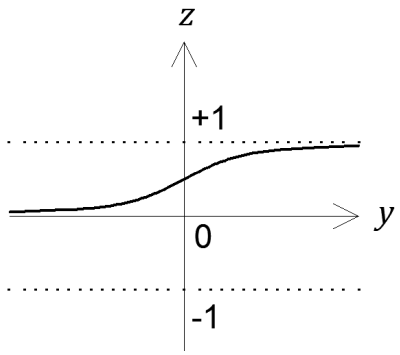


Figura 3.15: Función sigmoidal o logística

La función sigmoidal genera salidas entre 0 y 1, ya que los valores de entrada a la red de la neurona van desde $[-\infty, +\infty]$.

Alternativamente, las redes multicapa pueden usar la función de transferencia tangente hiperbólica, ya que es una función continua y derivable.

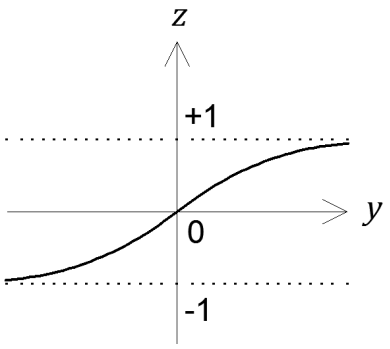


Figura 3.16: Función tangente hiperbólica

Ocasionalmente, es posible usar la función de transferencia lineal en la redes de propagación hacia atrás.

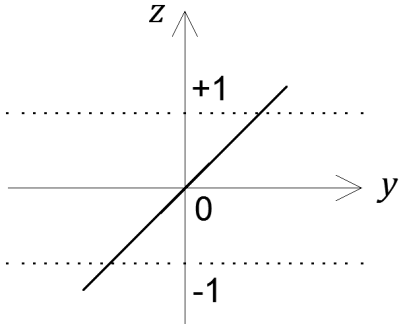


Figura 3.17: Función lineal

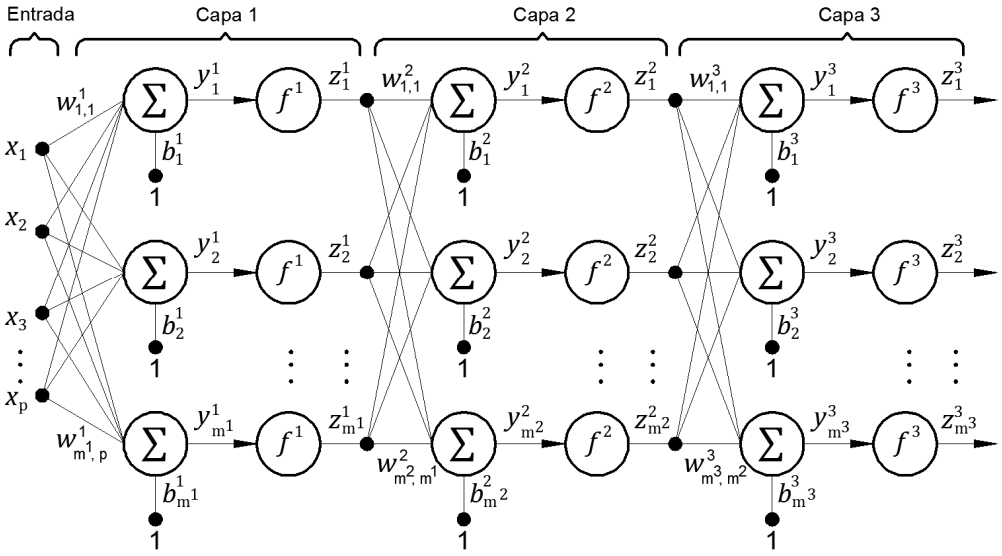
Si la última capa de una red multicapa tiene neuronas sigmoidales, entonces las salidas de la red se verán limitadas a un rango muy pequeño. Si se usan las neuronas de salidas lineales, las salidas de la red podrían tomar cualquier valor.

3.2.2. Arquitectura

Cada capa de una red neuronal puede tener su propia función de transferencia. La entrada acepta señales del mundo exterior y redistribuye esas señales a todas las neuronas en las capas ocultas. Por otro lado, la última capa acepta señales de salida, en otras palabras un patrón de estímulo desde la capa oculta y establece el patrón de salida de toda la red.

Con una capa oculta, es posible representar cualquier función continua de las señales de entrada, y con dos capas es posible representar incluso las funciones discontinuas, aunque esto pueda representar un mayor esfuerzo computacional.

Algunos programas comerciales de redes neuronales incorporan una o dos capas ocultas. Cada una de estas capas puede contener entre 10 a 1000 neuronas. Las redes neuronales experimentales pueden llegar a tener de tres o cuatro capas y utilizar millones de neuronas, pero las aplicaciones más prácticas usan solo una capa oculta, porque cada capa adicional incrementaría el esfuerzo computacional de manera exponencial.



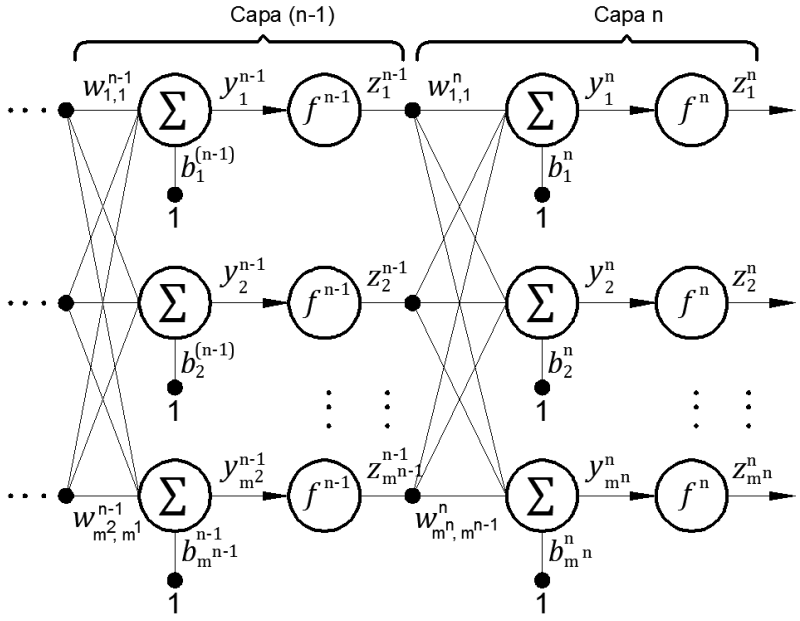


Figura 3.18: Red multicapa de neuronas sigmoidales

La figura 3.19 muestra la nomenclatura matricial para la red de la figura 3.18, de acuerdo a un esquema funcional en Matlab, que puede ser empleado incluso en la construcción de algoritmos recursivos para casos que contemplen dicha naturaleza.

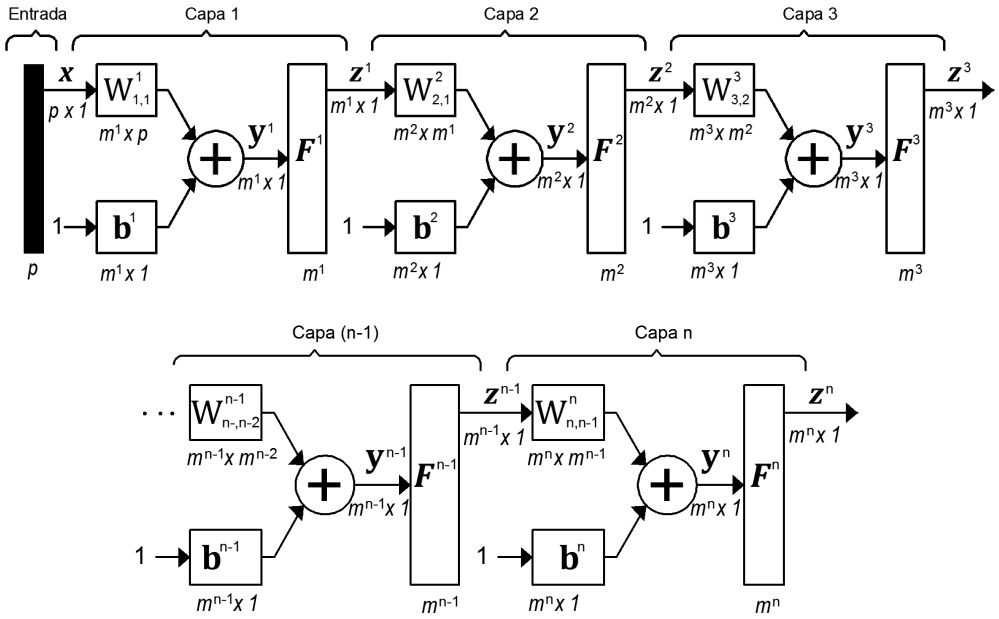


Figura 3.19: Arquitectura de la red multicapa de neuronas sigmoidales

3.2.3. Regla de Aprendizaje

Demostración realizada por Lara [10]:

Haciendo: $m^0 = p$

Realizando la propagación hacia delante:

$$y_1^1 = w_{1,1}^1 y_1^0 + w_{1,2}^1 y_2^0 + \cdots + w_{1,m^0}^1 y_{m^0}^0 + b_1^1$$

$$y_2^1 = w_{2,1}^1 y_1^0 + w_{2,2}^1 y_2^0 + \cdots + w_{2,m^0}^1 y_{m^0}^0 + b_2^1$$

...

$$y_{m^1}^1 = w_{m^1,1}^1 y_1^0 + w_{m^1,2}^1 y_2^0 + \cdots + w_{m^1,m^0}^1 y_{m^0}^0 + b_{m^1}^1$$

Matricialmente:

$$y^1 = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & \cdots & w_{1,m^0}^1 \\ w_{2,1}^1 & w_{2,2}^1 & \cdots & w_{2,m^0}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{m^1,1}^1 & w_{m^1,2}^1 & \cdots & w_{m^1,m^0}^1 \end{bmatrix} \begin{bmatrix} y_1^0 \\ \vdots \\ y_{m^0}^0 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_{m^1}^1 \end{bmatrix}$$

Por lo tanto:

$$\boldsymbol{y}^1 = W^1 \boldsymbol{y}^0 + \boldsymbol{b}^1$$

Generalizando:

$$\boldsymbol{y}^q = W^q \boldsymbol{y}^{q-1} + \boldsymbol{b}^q$$

$$z_1^1 = f^1(y_1^1)$$

$$z_2^1 = f^1(y_2^1)$$

...

$$z_{m^1}^1 = f^1(y_{m^1}^1)$$

$\begin{bmatrix} z_1^1 \\ z_2^1 \\ \vdots \\ z_{m^1}^1 \end{bmatrix} = \begin{bmatrix} f_1^1(y_1^1) \\ f_2^1(y_2^1) \\ \vdots \\ f_{m^1}^1(y_{m^1}^1) \end{bmatrix}$, haciendo $F^1(y^1) = \begin{bmatrix} f_1^1(y_1^1) \\ f_2^1(y_2^1) \\ \vdots \\ f_{m^1}^1(y_{m^1}^1) \end{bmatrix}$, queda $\boldsymbol{z}^1 = F^1(\boldsymbol{y}^1)$

Generalizando:

$$\boldsymbol{z}^q = F^q(\boldsymbol{y}^q), \quad q = 1 \dots n$$

$$e_1 = d_1 - z_1^n$$

$$e_2 = d_2 - z_2^n$$

...

$$e_{m^n} = d_{m^n} - z_{m^n}^n$$

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{m^n} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{m^n} \end{bmatrix} - \begin{bmatrix} z_1^n \\ z_2^n \\ \vdots \\ z_{m^n}^n \end{bmatrix}$$

$e_l = d_l - z_l^n$, $l = 1 \dots m^n$, en general $\boldsymbol{e} = \boldsymbol{d} - \boldsymbol{z}^n$

El error medio cuadrático:

$$\epsilon = (e_1)^2 + (e_2)^2 + \cdots + (e_{m^n})^2 = \sum_{i=1}^{m^n} (e_i)^2 = \boldsymbol{e}^T \boldsymbol{e}$$

Propagación hacia adelante:

Escalarmente:

$$w_{i,j}^n(k+1) = w_{i,j}^n(k) - \alpha \frac{\partial \epsilon(k)}{\partial w_{i,j}^n(k)}$$

$$b_i^n(k+1) = b_i^n(k) - \alpha \frac{\partial \epsilon(k)}{\partial b_i^n(k)}$$

Donde k indica el número de iteración.

$$\begin{bmatrix} w_{1,1}^{n}(k+1) & w_{1,2}^{n}(k+1) & \dots & w_{1,m^0}^{1}(k+1) \\ w_{2,1}^{n}(k+1) & w_{2,2}^{n}(k+1) & \dots & w_{2,m^n}^{nc}(k+1) \\ \vdots & \vdots & \ddots & \vdots \\ w_{m^n,1}^{n}(k+1) & w_{m^n,2}^{n}(k+1) & \dots & w_{m^{n-1},m^n}^{nc}(k+1) \end{bmatrix}$$

$$= \begin{bmatrix} w_{1,1}^{n}(k) & w_{1,2}^{n}(k) & \dots & w_{1,m^n}^{n}(k) \\ w_{2,1}^{n}(k) & w_{2,2}^{n}(k) & \dots & w_{2,m^n}^{n}(k) \\ \vdots & \vdots & \ddots & \vdots \\ w_{m^n,1}^{n}(k) & w_{m^n,2}^{n}(k) & \dots & w_{m^{n-1},m^n}^{n}(k) \end{bmatrix}$$

$$- \alpha \begin{bmatrix} \frac{\partial \epsilon(k)}{\partial w_{1,1}^{n}(k)} & \frac{\partial \epsilon(k)}{\partial w_{1,2}^{n}(k)} & \dots & \frac{\partial \epsilon(k)}{\partial w_{1,m^n}^{n}} \\ \frac{\partial \epsilon(k)}{\partial w_{2,1}^{n}(k)} & \frac{\partial \epsilon(k)}{\partial w_{2,2}^{n}(k)} & \dots & \frac{\partial \epsilon(k)}{\partial w_{2,m^n}^{n}(k)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \epsilon(k)}{\partial w_{m^n,1}^{n}(k)} & \frac{\partial \epsilon(k)}{\partial w_{m^n,2}^{n}(k)} & \dots & \frac{\partial \epsilon(k)}{\partial w_{m^{n-1},m^n}^{n}(k)} \end{bmatrix}$$

Es decir:

$$W^n(k+1) = W^n(k) - \alpha \frac{\partial \epsilon(k)}{\partial W^n(k)}$$

Y escalarmente:

$$\frac{\partial \epsilon}{\partial w_{i,j}^{n}} = \frac{\partial \epsilon}{\partial e_i} \frac{\partial e_i}{\partial y_i^n} \frac{\partial y_i^n}{\partial z_i^n} \frac{\partial z_i^n}{\partial w_{i,j}^{n}} = \frac{\partial \epsilon}{\partial z_i^n} \frac{\partial z_i^n}{\partial w_{i,j}^{n}}$$

Matricialmente:

$$\frac{\partial \epsilon}{\partial W^n} = \frac{\partial \epsilon}{\partial \boldsymbol{z}^n} \frac{\partial \boldsymbol{z}^n}{\partial W^n}$$

$\frac{\partial \epsilon}{\partial \boldsymbol{z}^n}$ = derivada parcial de una función escalar con respecto a un vector.

$\frac{\partial \boldsymbol{z}^n}{\partial W^n}$ = derivada de una función vectorial de argumento matricial con respecto a una matriz.

Escalarmente:

$$\frac{\partial \epsilon}{\partial z_i^n} = \frac{\partial \epsilon}{\partial e_i} \frac{\partial e_i}{\partial y_i^n} \frac{\partial y_i^n}{\partial z_i^n}$$

Matricialmente se debe tener en cuenta que si f es una función de p, la cual es una función de y, que a su vez es una función del vector $\boldsymbol{x}$, entonces:

$$\frac{df}{dx} = \frac{dy}{dx}\frac{\partial p}{\partial y}\frac{\partial f}{\partial p}$$

Teniendo en cuenta lo anterior:

$$\frac{\partial \epsilon}{\partial \boldsymbol{z}^2} = \frac{\partial \boldsymbol{y}^n}{\partial \boldsymbol{z}^n}\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{z}^2}\frac{\partial \epsilon}{\partial \boldsymbol{e}} \rightarrow \frac{\partial \epsilon}{\partial W^2} = \frac{\partial \boldsymbol{y}^n}{\partial \boldsymbol{z}^n}\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{y}^n}\frac{\partial \epsilon}{\partial \boldsymbol{e}} * \frac{\partial \boldsymbol{z}^n}{\partial W^n}$$

Teniendo en cuenta que:

$$\frac{d(Ax)}{dA} = x^T \rightarrow \frac{\partial \boldsymbol{z}^n}{\partial W^n} = \frac{\partial (W^n \boldsymbol{y}^{n-1} + \boldsymbol{b}^n)}{\partial W^n} = (y^{n-1})^T$$

En general:

$$\frac{\partial \boldsymbol{z}^q}{\partial W^q} = (\boldsymbol{y}^{q-1})^T$$

$\frac{\partial \boldsymbol{y}^n}{\partial \boldsymbol{z}^n} = \begin{bmatrix} \frac{\partial y_1^n}{\partial z_1^n} & \frac{\partial y_2^n}{\partial z_1^n} & \cdots & \frac{\partial y_{m^n}^n}{\partial z_1^n} \\ \frac{\partial y_1^n}{\partial z_2^n} & \frac{\partial y_2^n}{\partial z_2^n} & \cdots & \frac{\partial y_{m^n}^n}{\partial z_2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1^n}{\partial z_{m^n}^n} & \frac{\partial y_2^n}{\partial z_{m^n}^n} & \cdots & \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} \end{bmatrix}$, ya que $y_i^n = f_i^n(z_i^n)$, entonces, $\frac{\partial y_i^n}{\partial z_j^n} = 0$, para $\forall i \neq j$

y $\frac{\partial y_i^n}{\partial z_j^n} = f_i^{n\prime}(z_i^n)$

$\frac{\partial \boldsymbol{y}^n}{\partial \boldsymbol{z}^n} = \begin{bmatrix} \frac{\partial y_1^n}{\partial z_1^n} & 0 & \cdots & 0 \\ 0 & \frac{\partial y_2^n}{\partial z_2^n} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} \end{bmatrix}$, como $y_i^n = f_i^n(z_i^n) \rightarrow \frac{\partial y_i^n}{\partial z_i^n} = f_i^{n\prime}(z_i^n)$

$\boldsymbol{y}^n = F^n(\boldsymbol{z}^n) \rightarrow$ se puede hacer $F^{n\prime}(\boldsymbol{z}^n) = \frac{\partial \boldsymbol{y}^n}{\partial \boldsymbol{z}^n}$

$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{y}^n} = \begin{bmatrix} \frac{\partial e_1}{\partial y_1^n} & \frac{\partial e_2}{\partial y_1^n} & \cdots & \frac{\partial e_{m^n}}{\partial y_1^n} \\ \frac{\partial e_1}{\partial y_2^n} & \frac{\partial e_2}{\partial y_2^n} & \cdots & \frac{\partial e_{m^n}}{\partial y_2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_1}{\partial y_{m^n}^n} & \frac{\partial e_2}{\partial y_{m^n}^n} & \cdots & \frac{\partial e_{m^n}}{\partial y_{m^n}^n} \end{bmatrix}$, ya que $e_i = d_i - y_i^2$, entonces $\frac{\partial e_i}{\partial y_j^n} = 0$, para $\forall i \neq j$

y $\frac{\partial e_i}{\partial y_i^n} = -1$

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{y}^n} = \begin{bmatrix} \frac{\partial e_1}{\partial y_1^n} & 0 & 0 & 0 \\ 0 & \frac{\partial e_2}{\partial y_2^n} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{\partial e_{m^n}}{\partial y_{m^n}^n} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\frac{\partial \varepsilon}{\partial \boldsymbol{e}} = \frac{\partial}{\partial \boldsymbol{e}}(\boldsymbol{e}^T \boldsymbol{e}) = 2e = 2\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{m^n} \end{bmatrix}$$

Acoplando todos estos resultados:

$$\frac{\partial \varepsilon}{\partial W^n} = \begin{bmatrix} \frac{\partial y_1^n}{\partial z_1^n} & 0 & 0 & 0 \\ 0 & \frac{\partial y_2^n}{\partial z_2^n} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} \end{bmatrix} \begin{bmatrix} \frac{\partial e_1}{\partial z_1^n} & 0 & 0 & 0 \\ 0 & \frac{\partial e_2}{\partial z_2^n} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{\partial e_{m^n}}{\partial z_{m^n}^n} \end{bmatrix} * 2\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{m^n} \end{bmatrix}$$

$$* \begin{bmatrix} y_1^{n-1} & y_2^{n-1} & \cdots & y_{m^{n-1}}^{n-1} \end{bmatrix}$$

$$\frac{\partial \varepsilon}{\partial W^n} = 2\begin{bmatrix} -e_1 \frac{\partial y_1^n}{\partial z_1^n} \\ -e_2 \frac{\partial y_2^n}{\partial z_2^n} \\ \vdots \\ -e_{m^n} \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} \end{bmatrix} \begin{bmatrix} y_1^{n-1} & y_2^{n-1} & \cdots & y_{m^{n-1}}^{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} -2e_1 \frac{\partial y_1^n}{\partial z_1^n} y_1^{n-1} & -2e_1 \frac{\partial y_1^n}{\partial z_1^n} y_2^{n-1} & \cdots & -2e_1 \frac{\partial y_1^n}{\partial z_1^n} y_{m^{n-1}}^{n-1} \\ -2e_2 \frac{\partial y_2^n}{\partial z_2^n} y_1^{n-1} & -2e_2 \frac{\partial y_2^n}{\partial z_2^n} y_2^{n-1} & \cdots & -2e_2 \frac{\partial y_2^n}{\partial z_2^n} y_{m^{n-1}}^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ -2e_{m^n} \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} y_1^{n-1} & -2e_{m^n} \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} y_2^{n-1} & \cdots & -2e_{m^n} \frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} y_{m^{n-1}}^{n-1} \end{bmatrix}$$

Si se hace

$$\boldsymbol{\delta}^n = -\frac{\partial \varepsilon}{\partial z^n} = 2\begin{bmatrix} -\frac{\partial y_1^n}{\partial z_1^n} & 0 & 0 & 0 \\ 0 & -\frac{\partial y_2^n}{\partial z_2^n} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & -\frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} \end{bmatrix}\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{m^n} \end{bmatrix} = \begin{bmatrix} 2e_1\frac{\partial y_1^n}{\partial z_1^n} \\ 2e_2\frac{\partial y_2^n}{\partial z_2^n} \\ \vdots \\ 2e_{m^n}\frac{\partial y_{m^n}^n}{\partial z_{m^n}^n} \end{bmatrix}$$

$$= \begin{bmatrix} 2f_1^{n\prime}(z_1^n)e_1 \\ 2f_2^{n\prime}(z_1^n)e_2 \\ \vdots \\ 2f_{m^n}^{n\prime}\left(z_{m^n}^n\right)e_{m^n} \end{bmatrix} = \begin{bmatrix} \delta_1^n \\ \delta_2^n \\ \vdots \\ \delta_{m^n}^n \end{bmatrix}$$

Matricialmente

$$\boldsymbol{\delta}^n = 2F^{2\prime}(\boldsymbol{z}^n)e \rightarrow donde\ F^{n\prime}(\boldsymbol{z}^n) = \begin{bmatrix} f_1^{n\prime}(z_1^n) & 0 & 0 & 0 \\ 0 & f_2^{n\prime}(z_2^n) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & f_{m^n}^{n\prime}(z_{m^n}^n) \end{bmatrix}$$

Entonces,

$$\frac{\partial \varepsilon}{\partial W^n} = -\delta^n (y^{n-1})^T$$

Recordando que

$$W^n(k+1) = W^n(k) - \alpha\frac{\partial \varepsilon(k)}{\partial W^n(k)}$$

$$W^n(k+1) = W^n(k) - \alpha\delta^n(n)\left(y_{(k)}^{n-1}\right)^T$$

Ahora

$$b^n(k+1) = b^2(k) - \alpha\frac{\partial \varepsilon(k)}{\delta b^n(k)}$$

$$\frac{\partial \varepsilon}{\partial b^n} = \frac{\partial z^n}{\partial b^n}\frac{\partial \varepsilon}{\partial z^n} = -\frac{\partial z^n}{\partial b^n}\delta^n$$

Pero $z^n = W^n y^{n-1} + b^n$ recuérdese que $y^n = W^{n-1}z^{n-1} + b^{n-1}$

$$\frac{\partial z^n}{\partial b^n} = \begin{bmatrix} \frac{\partial z_1^n}{\partial b_1^n} & \frac{\partial z_2^n}{\partial b_1^n} & \cdots & \frac{\partial z_{m^n}^n}{\partial b_1^n} \\ \frac{\partial z_1^n}{\partial b_2^n} & \frac{\partial z_2^n}{\partial b_2^n} & \cdots & \frac{\partial z_{m^n}^n}{\partial b_2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_1^n}{\partial b_{m^n}^n} & \frac{\partial z_2^n}{\partial b_{m^n}^n} & \cdots & \frac{\partial z_{m^n}^n}{\partial b_{m^n}^n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}$$

$$\frac{\partial \varepsilon}{\partial b^n} = -\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \delta_1^n \\ \delta_2^n \\ \vdots \\ \delta_{m^n}^n \end{bmatrix} = -\begin{bmatrix} \delta_1^n \\ \delta_2^n \\ \vdots \\ \delta_{m^n}^n \end{bmatrix} = -\delta^n$$

Por lo tanto

$$b^n(k+1) = b^n(k) + \alpha\delta^n$$

En general, para los pesos y el BIAS de la última capa se tiene

$$\delta^n = 2F^{n\prime}(z^n)e$$

$$\frac{\partial \varepsilon}{\partial W^n} = -\delta^n (y^{n-1})^T$$

$$W^n(k+1) = W^n(k) + \alpha\delta^n(k)\left(y_{(k)}^{n-1}\right)^T$$

$\frac{\partial \varepsilon}{\delta b^n} = -\delta^n$ y $b^n(k+1) = b^n(k) + \delta_{(k)}^n$ donde n es el número de capas de la red.

Ahora es necesario hallar la fórmula para la actualización de W^{n-1} y b^{n-1}

$$W^{n-1}(k+1) = W^{n-1}(k) + \alpha \frac{\partial \varepsilon(k)}{\partial W^{n-1}(k)}$$

$\frac{\partial \varepsilon}{\partial W^{n-1}} = \frac{\partial \varepsilon}{\partial z^{n-1}} \frac{\partial z^{n-1}}{\partial W^{n-1}}$ y $\frac{\partial \varepsilon}{\partial z^{n-1}} = \frac{\partial y^{n-1}}{\partial z^{n-1}} \frac{\partial z^n}{\partial y^{n-1}} \frac{\partial y^n}{\partial z^n} \frac{\partial \varepsilon}{\partial y^n} \frac{\partial e}{\partial e} = \frac{\partial y^{n-1}}{\partial z^{n-1}} \frac{\partial z^n}{\partial y^{n-1}} \frac{\partial \varepsilon}{\partial z^n} = \frac{\partial z^n}{\partial z^{n-1}} \frac{\partial \varepsilon}{\partial z^n}$

$$\frac{\partial y^{n-1}}{\partial z^{n-1}} = \begin{bmatrix} \frac{\partial y_1^{n-1}}{\partial z_1^{n-1}} & \frac{\partial y_2^{n-1}}{\partial z_1^{n-1}} & \cdots & \frac{\partial y_{m^{n-1}}^{n-1}}{\partial z_1^{n-1}} \\ \frac{\partial y_1^{n-1}}{\partial z_2^{n-1}} & \frac{\partial y_2^{n-1}}{\partial z_2^{n-1}} & \cdots & \frac{\partial y_{m^{n-1}}^{n-1}}{\partial z_2^{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1^{n-1}}{\partial z_{m^{n-1}}^{n-1}} & \frac{\partial y_1^{n-1}}{\partial z_{m^{n-1}}^{n-1}} & \cdots & \frac{\partial y_{m^{n-1}}^{n-1}}{\partial z_{m^{n-1}}^{n-1}} \end{bmatrix}$$

Recordando que $y_i^{n-1} = f_i^{n-1}\left(z_i^{n-1}\right) \rightarrow \frac{\partial y_i^{n-1}}{\partial z_j^{n-1}} = 0,\ \forall i \neq j \ \rightarrow \frac{\partial y_i^{n-1}}{\partial S_i^{n-1}} = f^{n-1\prime}(z_i^{n-1})$

$$\frac{\partial y^{n-1}}{\partial z^{n-1}} = \begin{bmatrix} \frac{\partial y_1^{n-1}}{\partial z_1^{n-1}} & 0 & \dots & 0 \\ 0 & \frac{\partial y_1^{n-1}}{\partial z_2^{n-1}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\partial y_{m^{n-1}}^{n-1}}{\partial z_{m^{n-1}}^{n-1}} \end{bmatrix}$$

$$= \begin{bmatrix} f_1^{n-1\prime}(z_1^{n-1}) & 0 & \dots & 0 \\ 0 & f_2^{n-1\prime}(z_2^{n-1}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f_{m^{n-1}}^{n-1\prime}(z_{m^{n-1}}^{n-1}) \end{bmatrix} = F^{n-1\prime}(z^{n-1})$$

$$\frac{\partial z^n}{\partial y^{n-1}} = \frac{\partial}{\partial y^{n-1}}(W^n y^{n-1} + b^n) = (W^n)^T$$

Es decir

$$\frac{\partial z^n}{\partial y^{n-1}} = \begin{bmatrix} \frac{\partial z_1^n}{\partial y_1^{n-1}} & \frac{\partial z_2^n}{\partial y_1^{n-1}} & \dots & \frac{\partial z_{m^n}^n}{\partial y_1^{n-1}} \\ \frac{\partial z_1^n}{\partial y_2^{n-1}} & \frac{\partial z_2^n}{\partial y_2^{n-1}} & \dots & \frac{\partial z_{m^n}^n}{\partial y_2^{n-1}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial z_1^n}{\partial y_{m^{n-1}}^{n-1}} & \frac{\partial z_2^n}{\partial y_{m^{n-1}}^{n-1}} & \dots & \frac{\partial z_{m^n}^n}{\partial y_{m^{n-1}}^{n-1}} \end{bmatrix}$$ recordando que

$$z_1^n = W_{i,1}^n y_1^{n-1} + W_{i,2}^n y_2^{n-1} + \cdots + W_{i,m^{n-1}}^n y_{m^{n-1}}^{n-1} + b_i^n \ \rightarrow \frac{\partial z_i^n}{\partial y_j^{n-1}} = W_{i,j}^n$$

$$\frac{\partial z^n}{\partial y^{n-1}} = \begin{bmatrix} W_{1,1}^n & W_{2,1}^n & \dots & W_{m^n,1}^n \\ W_{1,2}^n & W_{2,2}^n & \dots & W_{m^n,2}^n \\ \vdots & \vdots & \ddots & \vdots \\ W_{1,m^{n-1}}^n & W_{2,m^{n-1}}^n & \dots & W_{m^n,m^{n-1}}^n \end{bmatrix} = (W^n)^T$$

Recordando que $\frac{\partial \epsilon}{\partial z^n} = -\delta^n$

$$\frac{\partial \epsilon}{\partial z^{n-1}} = -\begin{bmatrix} \frac{\partial y_1^{n-1}}{\partial z_1^{n-1}} & 0 & \cdots & 0 \\ 0 & \frac{\partial y_2^{n-1}}{\partial z_2^{n-1}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial y_{m^{n-1}}^{n-1}}{\partial z_{m^{n-1}}^{n-1}} \end{bmatrix} \begin{bmatrix} W_{1,1}^n & W_{2,1}^n & \dots & W_{m^n,1}^n \\ W_{1,2}^n & W_{2,2}^n & \dots & W_{m^n,2}^n \\ \vdots & \vdots & \ddots & \vdots \\ W_{1,m^{n-1}}^n & W_{2,m^{n-1}}^n & \dots & W_{m^n,m^{n-1}}^n \end{bmatrix} \begin{bmatrix} \delta_1^n \\ \delta_2^n \\ \vdots \\ \delta_{m^n}^n \end{bmatrix}$$

$$\frac{\partial \epsilon}{\partial z^{n-1}} = -F^{n-1\prime}(z^{n-1})(W^n)^T\delta^n$$

$$\frac{\partial \varepsilon}{\partial z^{n-1}} = -\delta^{n-1}$$

$$\frac{\partial z^{n-1}}{\partial W^{n-1}} = \frac{\partial (W^{n-1}y^{n-2} + b^{n-1})}{\partial W^{n-1}} = (y^{n-2})^T$$

En general

$$\frac{\partial z^n}{\partial W^n} = (y^{n-1})^T$$

$$\frac{\partial \varepsilon}{\partial W^{n-1}} = \frac{\partial \varepsilon}{\partial z^{n-1}}\frac{\partial z^{n-1}}{\partial W^{n-1}} = -\delta^{n-1}(y^{n-2})^T$$

En general

$$\frac{\partial \varepsilon}{\partial W^n} = \frac{\partial \varepsilon}{\partial z^n}\frac{\partial z^n}{\partial W^n}$$

$$\frac{\partial \varepsilon}{\partial z^n} = \frac{\partial z^{n+1}}{\partial z^k}\frac{\partial \varepsilon}{\partial z^{n+1}}$$

$$\frac{\partial z^{n+1}}{\partial z^n} = \frac{\partial y^n}{\partial z^n}\frac{\partial z^{n+1}}{\partial y^n} \rightarrow \frac{\partial y^n}{\partial z^n} = F^{n\prime}(z^n)$$

$$\frac{\partial z^{n+1}}{\partial y^n} = \frac{\partial (W^{n+1}y^n + b^{n+1})}{\partial y^n} \rightarrow \frac{\partial z^{n+1}}{\partial y^n} = (W^{n+1})^T$$

$$\frac{\partial z^{n+1}}{\partial y^n} = F^{n\prime}(z^n)(W^{n+1})^T$$

$$\frac{\partial \varepsilon}{\partial z^{k+1}} = -\delta^{n+1} \rightarrow \frac{\partial \varepsilon}{\partial z^n} = -F^{n\prime}(z^n)(W^{n+1})^T\delta^{n+1}$$

Si se generaliza

$$\delta^n = -\frac{\partial \varepsilon}{\partial z^n} \rightarrow \delta^n = F^{n\prime}(z^n)(W^{n+1})^T\delta^{n+1}$$

$\delta^n = \left(W^{n+1}F^{n\prime}(z^n)\right)^T\delta^{n+1}$ ya que $F^{n\prime}(z^n)$ es simétrica

$$\frac{\partial \varepsilon}{\partial W^n} = \frac{\partial \varepsilon}{\partial z^n}\frac{\partial z^n}{\partial W^n} = \delta^n(y^{n-1})^T$$

$$W^{n-1}(k+1) = W^{n-1}(k) + \alpha\delta^{n-1}_{(k)}\left(y^{n-2}_k\right)^T \tag{3.6}$$

En general

$W^l(k+1) = W^l(k) + \alpha\delta^l_{(k)}\left(y^{l-1}_k\right)^T$ para $l = n \ldots 1$

Ahora se determina la fórmula de actualización para b^{n-1}

$$b^{n-1}(k+1) = b^{n-1}(k) - \alpha\frac{\partial\varepsilon(k)}{\partial b^{n-1}(k)}$$

$$\frac{\partial z^{n-1}}{\partial b^{n-1}} = \begin{bmatrix} \frac{\partial z^{n-1}_1}{\partial b^{n-1}_1} & \frac{\partial z^{n-1}_2}{\partial b^{n-1}_1} & \cdots & \frac{\partial z^{n-1}_{m^{n-1}}}{\partial b^{n-1}_1} \\ \frac{\partial z^{n-1}_1}{\partial b^{n-1}_2} & \frac{\partial z^{n-1}_2}{\partial b^{n-1}_2} & \cdots & \frac{\partial z^{n-1}_{m^{n-1}}}{\partial b^{n-1}_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z^{n-1}_1}{\partial b^{n-1}_{m^{n-1}}} & \frac{\partial z^{n-1}_2}{\partial b^{n-1}_{m^{n-1}}} & \cdots & \frac{\partial z^{n-1}_{m^{n-1}}}{\partial b^{n-1}_{m^{n-1}}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = [I]_{kk(n-1)}$$

En general

$$\frac{\partial z^n}{\partial b^n} = [I]_{kk(n)}$$

$$\frac{\partial\varepsilon}{\partial b^n} = -[I]_{kk(n)}\delta^n = -\delta^n$$

$$b^{n-1}(k+1) = b^{n-1}(k) + \alpha\delta^{n-1}(k)$$

$$b^l(k+1) = b^l(k) + \alpha\delta^l(k) \tag{3.7}$$

3.2.4. Entrenamiento de la red

El algoritmo de entrenamiento Backpropagation es una técnica de aprendizaje supervisado. Es decir, la red deberá ser presentada con un conjunto de datos de entrada con sus correspondientes salidas, de esta manera la red podrá comparar los resultados deseados contra los resultados obtenidos por la red y calcular el error correspondiente a una época y detenerse en caso de cumplir con una tolerancia o propagar el error hacia atrás capa por capa y continuar con la siguiente época.

Se estima que el 85% de las aplicaciones de redes neuronales aplican alguna forma de entrenamiento con Backpropagation (Wasserman [11]). El redescubrimiento del entrenamiento Backpropagation por Rumelhart et al. [12] en 1986 aceleró el crecimiento de la investigación en el campo de las redes neuronales que años atrás se había abandonado por los excesivos requerimientos computacionales de la época. A continuación se muestra el algoritmo de entrenamiento para una red Backpropagation típica de dos capas:

Paso 1: Inicialización

Se establecen todos los pesos y umbrales de la red con valores aleatorios uniformemente distribuidos dentro de un pequeño rango [0 1] o [-1 1].

Paso 2: Activación

Activar la red neuronal Backpropagation aplicando las entradas $x_1, x_2, \dots, x_p$ o su equivalente $y_1^0, y_2^0, \dots, y_p^0$, y sus salidas deseadas $d_1, d_2, \dots, d_{m^{(2)}}$.

(a) Calcular las salidas actuales de las neuronas en la capa oculta:

$$z_j^{(1)}(k) = sigmoide\left[\sum_{i=1}^{p} x_i(k) \cdot w_{j,i}^{(1)}(k) - b_j^{(1)}\right] \forall\, j = 1 \dots m^{(1)}$$

Donde p es el número de entradas de la neurona j en la capa oculta y $sigmoide$ es la función de activación.

(b) Calcula las salidas actuales de las neuronas en la capa de salida:

$$z_j^{(2)}(k) = sigmoide\left[\sum_{i=1}^{m^{(1)}} z_i^{(1)}(k) \cdot w_{j,i}^{(2)}(k) - b_j^{(2)}\right] \forall\, j = 1 \dots m^{(2)}$$

Donde $m^{(1)}$ es el número neuronas en la capa oculta que servirán de entradas a la neurona j en la capa de salida y $m^{(2)}$ el número de salidas.

(*) El superíndice representa el número de capa actual.

Paso 3: Entrenamiento de los pesos

Actualizar los pesos en la red Backpropagation propagando los errores hacia atrás asociados con sus neuronas de salida.

(a) Calcular la gradiente del error para las neuronas en la capa de salida:

$$\delta_l^{(2)}(k) = z_l^{(2)}(k) \cdot \left[1 - z_l^{(2)}(k)\right] \cdot e_l(k) \quad \forall\, l = 1..m^{(2)}$$

Donde

$$e_l(k) = d_l - z_l^{(2)}(k)$$

Cálculo de las correcciones de los pesos:

$$\Delta w_{l,j}^{(2)}(k) = \alpha \cdot z_j^{(1)}(k) \cdot \delta_l^{(2)}(k) \quad \forall\, j = 1..m^{(1)}$$

Actualizar los pesos en las neuronas de salida:

$$w_{l,j}^{(2)}(k+1) = w_{l,j}^{(2)}(k) + \Delta w_{l,j}^{(2)}(k)$$

(b) Cálculo del gradiente de error para las neuronas en la capa oculta:

$$\delta_j^{(1)}(k) = z_j^{(1)}(k) \cdot \left[1 - z_j^{(1)}(k)\right] \cdot \sum_{l=1}^{m^{(2)}} \delta_l^{(2)}(k) \cdot w_{l,j}^{(2)}(k) \qquad \forall\, j = 1..m^{(1)}$$

Cálculo de la corrección de pesos:

$$\Delta w_{j,i}^{(1)}(k) = \alpha \cdot x_i(k) \cdot \delta_j^{(1)}(k)$$

Actualizar los pesos de las neuronas ocultas:

$$w_{j,i}^{(1)}(k+1) = w_{j,i}^{(1)}(k) + \Delta w_{j,i}^{(1)}(k) \qquad \forall\, i = 1..p$$

Paso 4: Iteración

Se incrementa la iteración k en uno, se regresa al paso 2 y se repite el proceso hasta el la tolerancia del error sea alcanzada.

3.2.4.1. Aprendizaje adaptativo

Teniendo en cuenta que la superficie de error bajo el algoritmo Backpropagation es relativamente plana con cañones empinados. La elección del tamaño de paso para el proceso de optimización es muy importante, esto debido a que el método de Backpropagation estándar no ajusta el tamaño de paso durante el proceso de optimización, es posible que se requieran demasiadas iteraciones para cruzar una superficie relativamente plana si se usa un paso pequeño. De la misma forma, si se elige un tamaño de paso grande puede fallar al momento de ubicar el mínimo saltándose

cualquier caída en la superficie de error. El aprendizaje adaptativo busca corregir ambos problemas modificando el tamaño de paso durante el proceso de optimización.

La manera más prominente de aprendizaje adaptativo fue propuesto por Jacobs [13]. La forma original del método, conocido como la regla delta-delta, introdujo tasas de aprendizajes separadas para cada peso en la red. Esta tasa de aprendizaje actúa como un factor para los cambios de los pesos. Por lo tanto, un pequeño factor hará que se ralentice el proceso de ajuste de los pesos y de este modo ralentizará el proceso de aprendizaje. De la misma forma, un factor grande magnificará los cambios en los pesos. Bajo la regla delta-delta, las tasas de aprendizajes son actualizadas de forma lineal de acuerdo a la pendiente de la superficie de error durante cada ciclo de entrenamiento. Si la derivada de la superficie de error con respecto al peso actual posee el mismo signo para iteraciones consecutivas, la tasa de aprendizaje se incrementará para acelerar la convergencia. Si la derivada cambia de signo durante las iteraciones, la tasa de aprendizaje será reducida para evitar oscilaciones en el proceso de solución. Ambos escenarios se muestran en la figura 3.20.

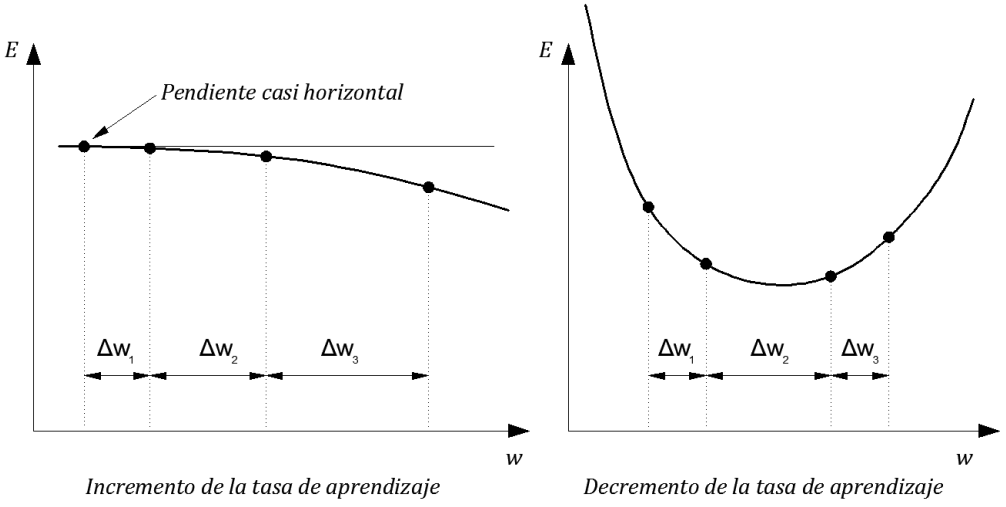


Figura 3.20: Variación de la tasa de aprendizaje

Adicionalmente es posible hacer una modificación en la regla delta-delta para producir la regla delta-bar-delta. Esta nueva regla incrementa la tasa de aprendizaje linealmente y la decrementa en forma exponencial. El decremento exponencial penaliza la tasa de aprendizaje más severamente que uno lineal para controlar mejor las oscilaciones alrededor del mínimo.

3.2.4.2. Momento

Una técnica sencilla de mejorar la tasa de convergencia durante el entrenamiento de la red es añadir un momento a la fórmula de la gradiente descendente. Este término momento brinda un efecto de inercia al movimiento a través del espacio de pesos y suaviza las oscilaciones en el mínimo local [14]. Este efecto se logra reteniendo el cambio de peso anterior durante el proceso de entrenamiento. Dependiendo del signo del anterior cambio de peso, el peso actual cambiará ya sea incrementándose o reduciéndose con el término momento suplementario.

La ventaja más significante de añadir un término momento a la rutina de Backpropagation es que previene innecesarias oscilaciones alrededor del mínimo local. A menos que se elija exactamente el paso, la rutina estándar del Backpropagation oscilará alrededor del mínimo, convergiendo lentamente al punto de error mínimo. Debido a que el requerimiento de que la gradiente en la superficie de error debe ser cero en el mínimo, la convergencia en este punto puede ser bastante lenta para pendientes empinadas alrededor del mínimo. Este fenómeno se muestra en la figura 3.21. Con la adición del término momento sin embargo la solución procede rápidamente al punto de error mínimo.

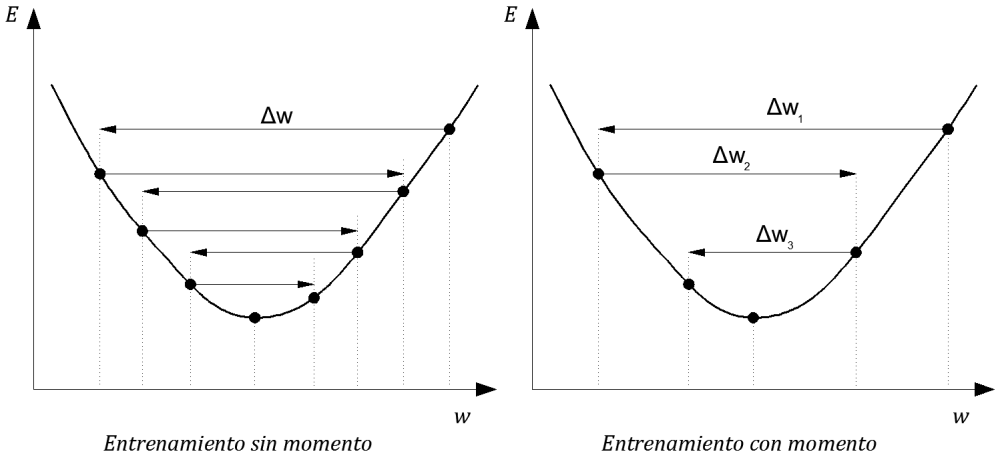


Figura 3.21: Soluciones de oscilación y momento

La segunda ventaja de un término momento es evitar el entrampamiento en mínimos locales pocos profundos. Para soluciones sencillas de gradiente descendente con un paso pequeño, el método se detendrá en el primer punto de mínimo error. Algunas veces este punto es un punto poco profundo al costado de un mínimo global más profundo. Como se mencionó, el término momento provee una inercia de movimiento a la rutina de gradiente descendente. En ciertos casos esta inercia es suficiente para presionar el movimiento fuera del mínimo local poco profundo en un esfuerzo por alcanzar un

mínimo más profundo como se muestra en la figura 3.22. Este efecto se presenta como un bono con la inclusión del momento al proceso de entrenamiento.

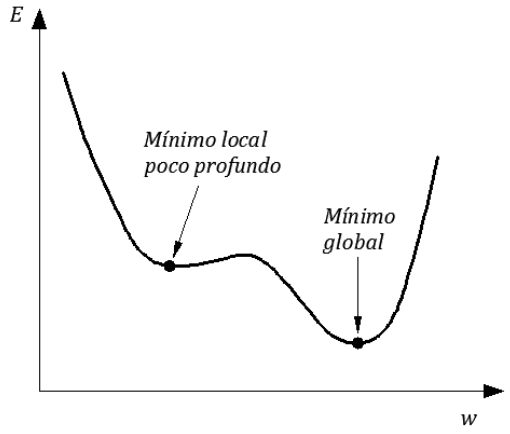


Figura 3.22: Escapa de zonas poco profundas

4. Aplicación de la Red Neuronal Artificial para conocer la Respuesta Sísmica de Muros de Albañilería

4.1. Especímenes de Muros Usados en los Ensayos

Los muros ensayados por Zavala y colaboradores [15] en el CISMID presentan las dimensiones mostradas en la siguiente figura:

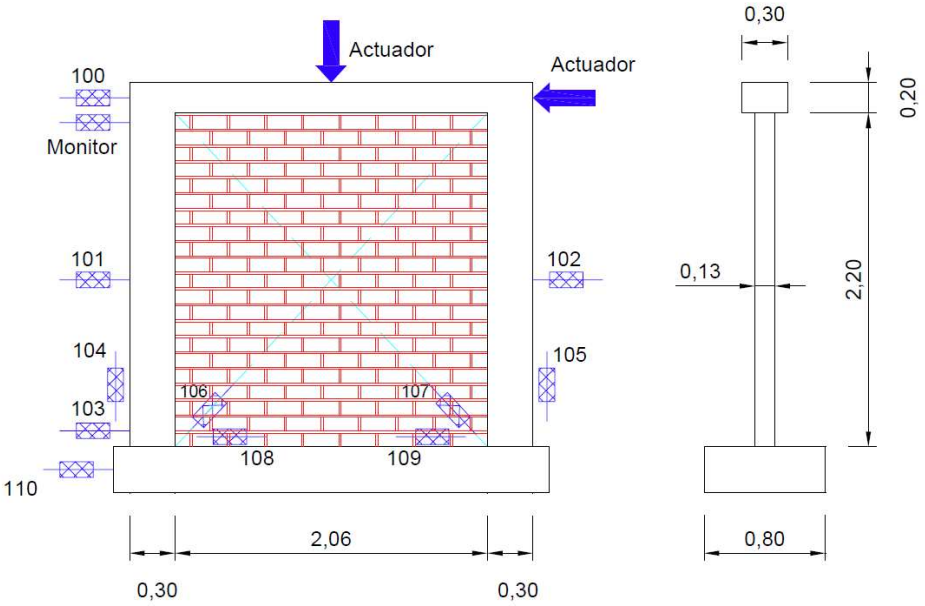


Figura 4.1: Dimensiones del muro patrón

Las características de los refuerzos están definidas de acuerdo a la tabla:

Espécimen	Ancho (cm)	Largo (cm)	Alto (cm)	Refuerzo Vertical	Refuerzo Hor.	Estribos	Obs.
MURO-01	13	260	240	4□1/2”	4□1/2”	□ ¼”@20	Patrón con 4 varillas dúctiles #4
MURO-02	13	260	240	4□3/8”	4□3/8”	□ ¼”@20	Patrón con 4 varillas dúctiles #3
MURO-03	13	260	240	8□8.3mm	8□8.3mm	22□ 5.5@20	Electrosoldado con refuerzo equivalente a 4 varillas dúctiles #4
MURO-04	13	260	240	8□8.3mm	8□8.3mm	22□ 5.5@20	Repetición
MURO-05	13	260	240	8□8.3mm	8□8.3mm	22□ 5.5@20	Repetición

MURO-06	13	260	240	4□8.5mm	4□8.5mm	22□ 5.5@20	Electrosoldado con refuerzo equivalente a 4 varillas dúctiles #3
MURO-07	13	260	240	4□8.5mm	4□8.5mm	22□ 5.5@20	Repetición
MURO-08	13	260	240	4□8.5mm	4□8.5mm	22□ 5.5@20	Repetición

Tabla 4.1: Especímenes de muros usados en los ensayos

MURO-01, MURO-02 son muros de albañilería confinada típicos. MURO-03 al MURO-08 son muros de albañilería armada con mallas electrosoldadas en sus elementos de confinamiento.

4.2. Generación del Modelo Numérico del Muro

Para la presente tesis se dispuso de la información del archivo de dibujo llamado grietasMuros.dwg, en el cual se encuentran las grietas para cada espécimen de muro ensayado sometido a cargas cíclicas.

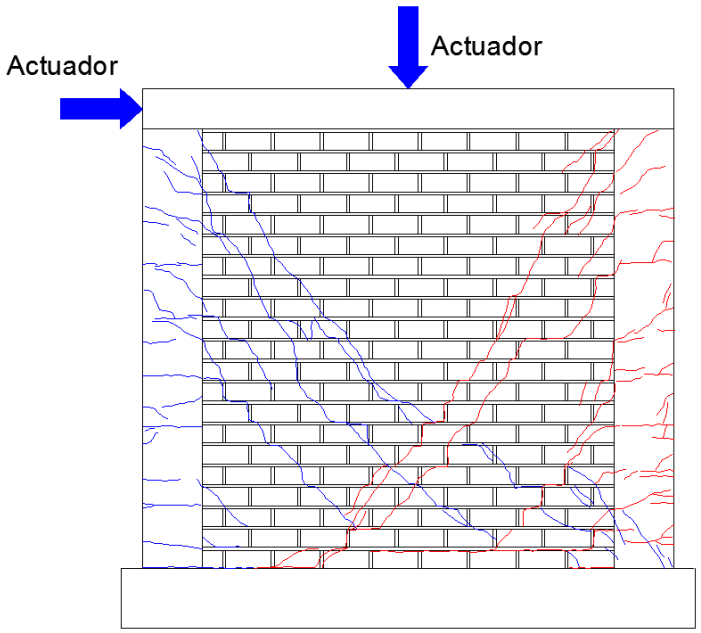


Figura 4.2: Modelo de muro usado durante los ensayos

Para recuperar las coordenadas de las grietas de cada muro que se encuentran en formato vectorial (*.dwg) se realizó un programa en AutoLISP que extrajo la información de la base de datos gráfica y que posteriormente almacenó en un archivo de texto llamado muro_grietas??.txt.

```
(defun c:extraer_data()
(setvar "cmdecho" 0)
(setq estado (getvar "osmode"))
(setvar "osmode" 0)
(setq pto (getpoint "Seleccione la esquina del muro:"))
(command "ucs" "move" pto)(terpri)
(setq nummuro (getstring "Ingrese nro. Muro:"))
(setq parte (getstring "Frontal o Posterior (F/P):"))
(setq nombremuro "d:\\datos de muros\\muro_grietas")
(setq nombremurototal (strcat nombremuro "-" parte "-" nummuro
".txt"))
(setq tipo "LWPOLYLINE")
(setq ruta nombremurototal)
(setq arch (open ruta "w"))
(setq objs (ssget))
(setq n (sslength objs))
(setq i 0)
    (repeat n
      (setq nomb (ssname objs i))
      (setq lista (entget nomb))
        (if (= (cdr (nth 1 lista)) tipo)
           (progn (foreach elem lista
                (if (= (car elem) 10)
                  (progn
                  (setq x (car (cdr elem)))
                    (setq y (caddr elem))
                  (setq pto (trans (list x y) 0 1))
                  (setq x (car pto))
                  (setq y (cadr pto))
                    (write-line (strcat (rtos x 2 6) " " (rtos y
2 6)) arch)
                  );;fin del if
                  )
                );; fin del foreach
             (write-line "" arch)
          )
       );;fin del if tipo
      (setq i (+ i 1))
    );; fin del repeat
(setvar "osmode" estado)
(close arch)
(setvar "cmdecho" 1)
(prin1)
)
```

Posteriormente se realizó otro programa en MATLAB para reproducir las grietas leídas del archivo de texto y finalmente transformar el muro con las grietas a una matriz de ceros y unos que pueda ser mostrada como una imagen de mapa de bits (ver figura 4.3) y procesada por la red neuronal.

A continuación se presentan los códigos fuente de las rutinas que permitirán reproducir las grietas en formatos de ceros y unos.

Primero, se presenta código en MATLAB que permite conectar dos pixeles dentro de una imagen de mapa de bits:

```
function Img = DibujarLinea(Img, X0, Y0, X1, Y1, nG)
Img(X0, Y0) = nG;
Img(X1, Y1) = nG;
if abs(X1 - X0) <= abs(Y1 - Y0)
   if Y1 < Y0
      k = X1; X1 = X0; X0 = k;
      k = Y1; Y1 = Y0; Y0 = k;
   end
   if (X1 >= X0) & (Y1 >= Y0)
      dy = Y1-Y0; dx = X1-X0;
      p = 2*dx; n = 2*dy - 2*dx; tn = dy;
      while (Y0 < Y1)
         if tn >= 0
            tn = tn - p;
         else
            tn = tn + n; X0 = X0 + 1;
         end
         Y0 = Y0 + 1; Img(X0, Y0) = nG;
      end
   else
      dy = Y1 - Y0; dx = X1 - X0;
      p = -2*dx; n = 2*dy + 2*dx; tn = dy;
      while (Y0 <= Y1)
         if tn >= 0
            tn = tn - p;
         else
            tn = tn + n; X0 = X0 - 1;
         end
         Y0 = Y0 + 1; Img(X0, Y0) = nG;
      end
   end
else if X1 < X0
      k = X1; X1 = X0; X0 = k;
      k = Y1; Y1 = Y0; Y0 = k;
   end
   if (X1 >= X0) & (Y1 >= Y0)
      dy = Y1 - Y0; dx = X1 - X0;
      p = 2*dy; n = 2*dx-2*dy; tn = dx;
      while (X0 < X1)
         if tn >= 0
            tn = tn - p;
         else
            tn = tn + n; Y0 = Y0 + 1;
         end
         X0 = X0 + 1; Img(X0, Y0) = nG;
      end
   else
      dy = Y1 - Y0; dx = X1 - X0;
      p = -2*dy; n = 2*dy + 2*dx; tn = dx;
      while (X0 < X1)
         if tn >= 0
            tn = tn - p;
         else
            tn = tn + n; Y0 = Y0 - 1;
```

```
          end
          X0 = X0 + 1; Img(X0, Y0) = nG;
       end
    end
end
```

Por consiguiente, si se requiere dibujar varias líneas, unidas entre sí por pixeles, que representen en conjunto el agrietamiento en un muro, entonces se puede hacer uso del código anterior dentro de un **script** para modelar un modelo numérico en el cual cada pixel sea de 2cms x 2cms tal como se resuelve en el siguiente código fuente:

```
ntotal_arch=6;
%%Abrir archivo de coordenadas para lectura
for i=1:ntotal_arch
    result=zeros(120+2,133+2);
        nomb_archivo_total= ...
        strcat('d:\\Tesis\\Datos de Muros\\muro_grietas', ...
        '-',int2str(i),'.txt');
        fp=fopen(nomb_archivo_total,'r');contador=0;
        parametro=fgets(fp);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        while ~feof(fp)
          if contador==0
            [pInicialX pInicialY]= transforma(parametro);
          end
          contador=contador+1;
              aux=fgets(fp);
              if length(aux)>2
               [pFinalX pFinalY] = transforma(aux);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

result=DibujarLinea(result,round(pInicialY*50)+1,...
               round(pInicialX*50)+1,round(pFinalY*50)+1,...
               round(pFinalX*50)+1,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               pInicialX=pFinalX;
               pInicialY=pFinalY;
              elseif ~feof(fp)
               [pInicialX pInicialY]= transforma(fgets(fp));
              end%%fin del if-else
         end%%fin del while
        temp=reshape(flipud(result),16470,1);
        fclose(fp);
    if i==1
    T=temp;
    else
        T=[T temp];
    end
    imshow(flipud(result));pause(2);close;
    P=load('-ascii','d:\\Tesis\\Datos de Muros\\data.txt');
    clear result;
    clear temp;
end%% fin del for i
```

La figura 4.3 muestra el resultado de pasar el formato vectorial de la izquierda al formato de ceros y unos de la derecha usando la rutina anterior.

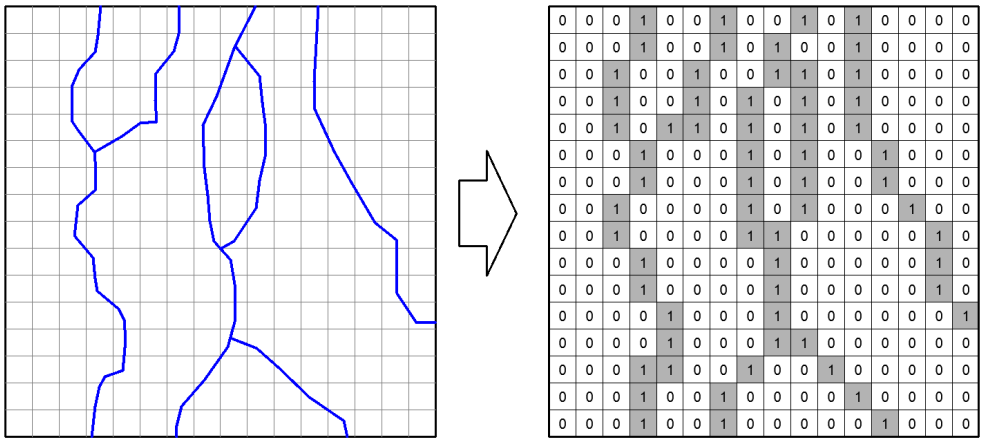


Figura 4.3: Transformación del formato vectorial al formato matricial de ceros y unos. Izquierda: Agrietamiento obtenido del ensayo Derecha: Modelo numérico del agrietamiento

La figura 4.4 muestra un ejemplo del muro que será procesado por el programa presentado en MATLAB.

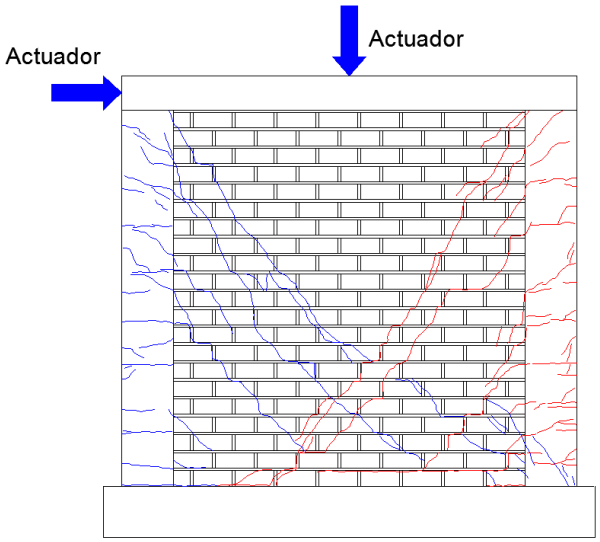


Figura 4.4: Ejemplo de muro agrietado a representar numéricamente

El resultado de transformar las grietas a una imagen de mapa de bits se muestra en la figura 4.5 en el formato que la computadora podrá interpretarla para el aprendizaje.

>> script

```
>> imshow(reshape(T,122,135))
```

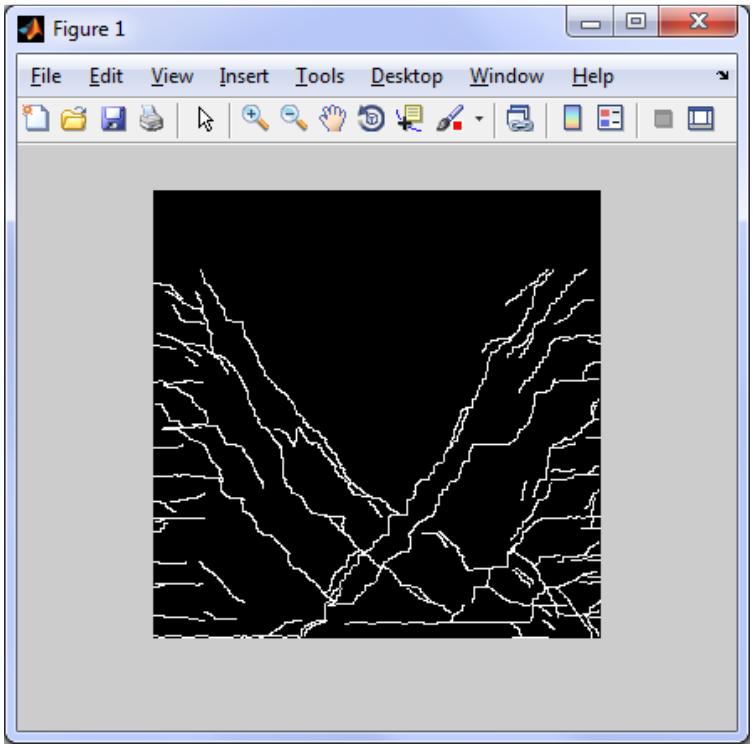


Figura 4.5: Muro en imagen de mapa de bits

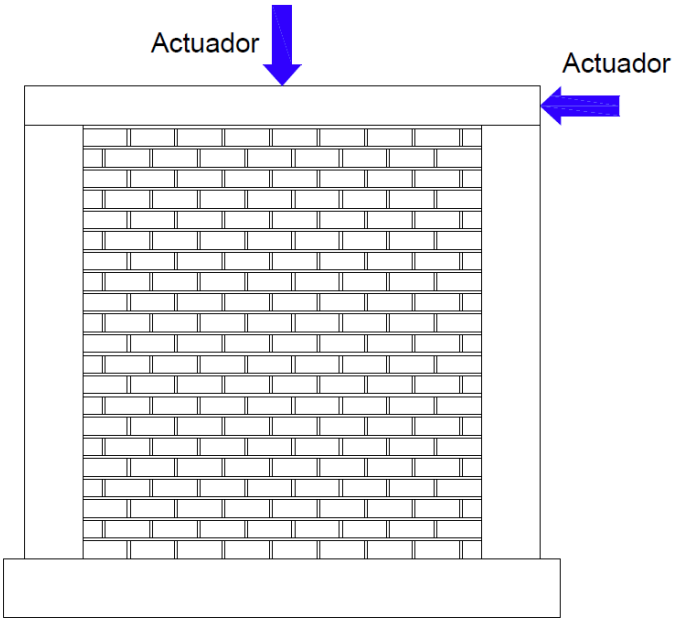
Posteriormente este archivo de mapa de bits será usado como el patrón de aprendizaje en la red neuronal *Backpropagation*.

4.3. Datos de Entrenamiento del Muro Patrón I

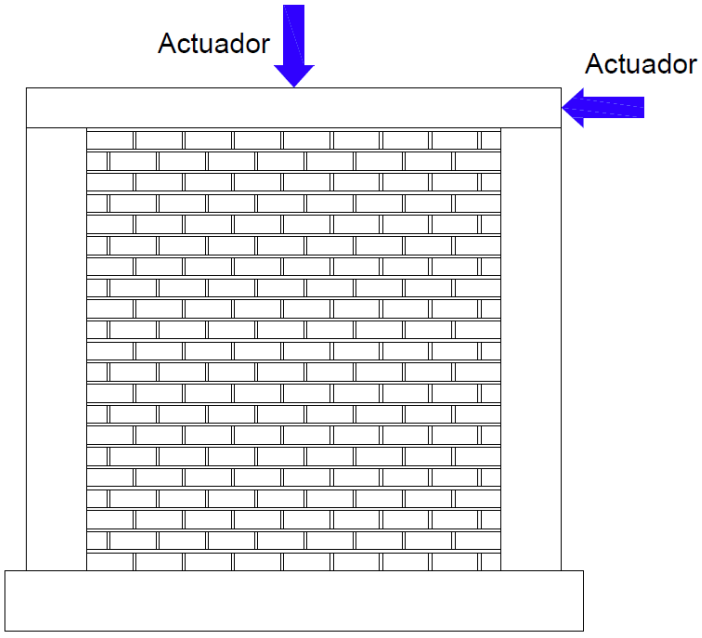
Se ha optado por tomar los datos del muro patrón I para el entrenamiento de la red, debido a las limitaciones de la memoria RAM para más datos de entrenamiento, que supongan el uso de más capas ocultas o de neuronas en la capa oculta de la red.

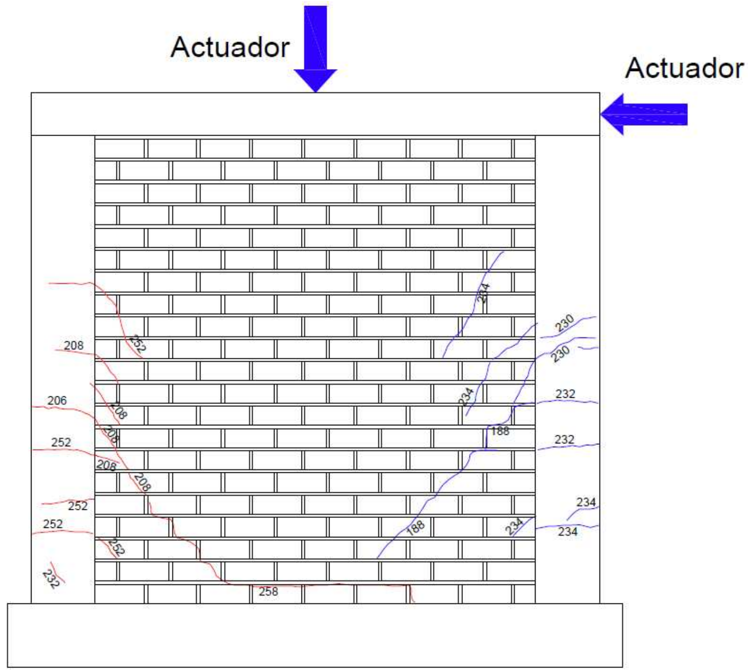
El procedimiento del ensayo consiste en someter un muro a desplazamientos cíclicos laterales controlados y determinar la respuesta del muro a estos desplazamientos, a su vez que se van registrando los agrietamientos. Las figuras 4.6 muestran la evolución del agrietamiento conforme se incrementan los desplazamientos laterales.

Muro I
Distorsión 1/2700
Cara Frontal

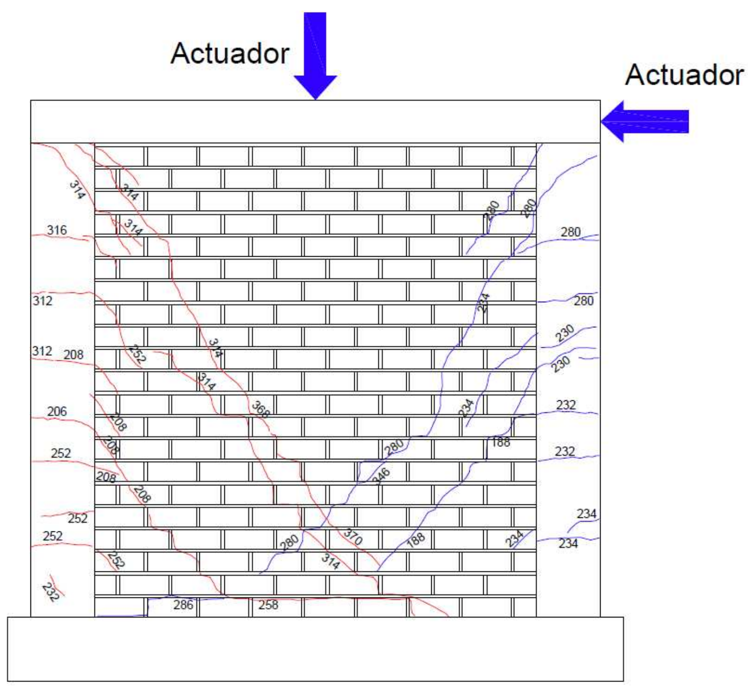


Muro I
Distorsión 1/1350
Cara Frontal





Muro I
Distorsión 1/675
Cara Frontal



Muro I
Distorsión 1/350
Cara Frontal

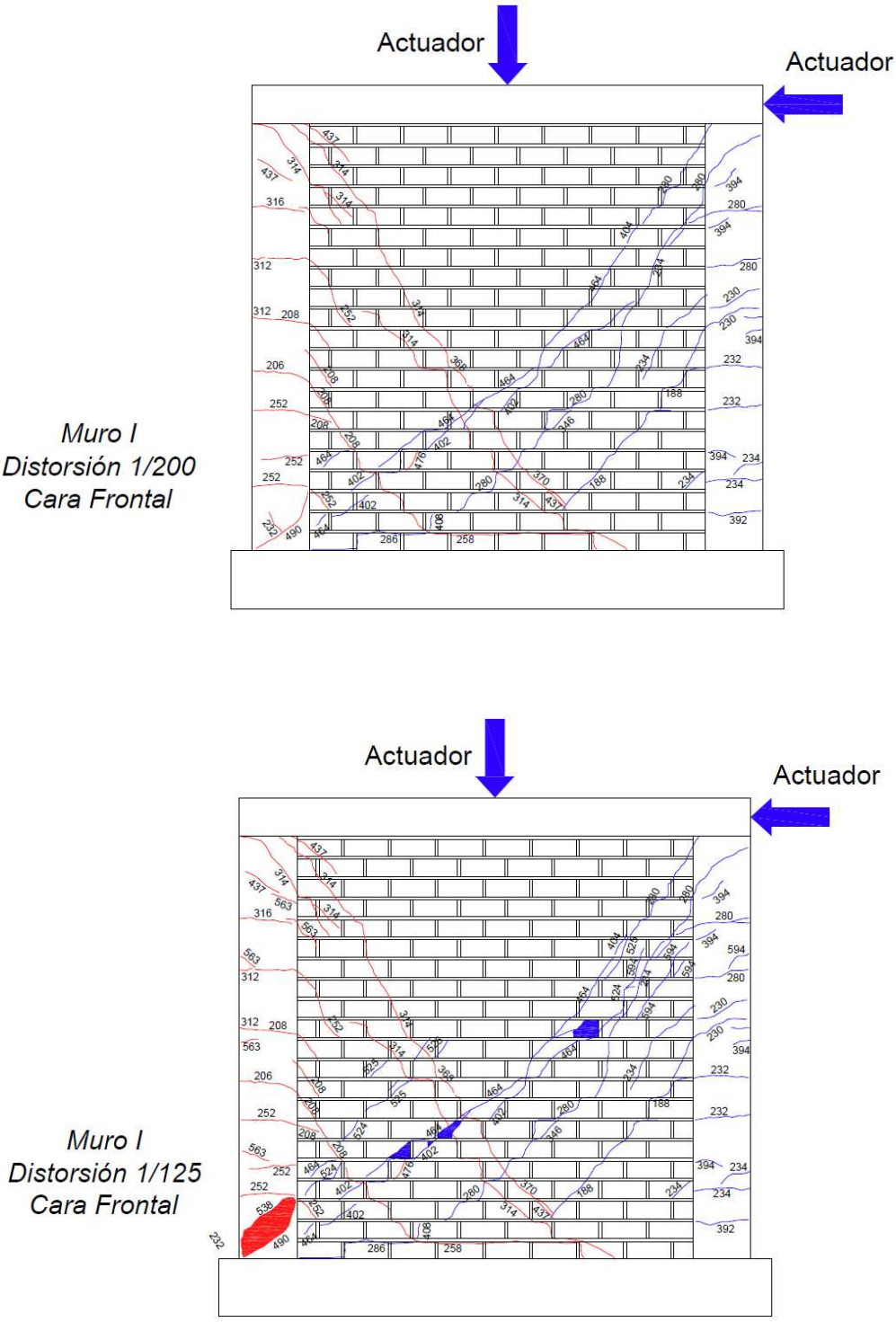


Figura 4.6: Agrietamiento evolutivo del muro sometido a desplazamiento lateral

Para poder usar los datos de los agrietamientos de los muros de la figura 4.6, éstos deberán presentarse con la representación de modelos de ceros y unos de acuerdo a lo especificado en la sección 4.2.

En la figura 4.7 se muestra la espaguetización realizada a la matriz de 16x16 usada como ejemplo a un vector de 256x1. La espaguetización consiste en colocar una columna sobre otra a partir de la matriz bidimensional y así obtener un solo vector columna.

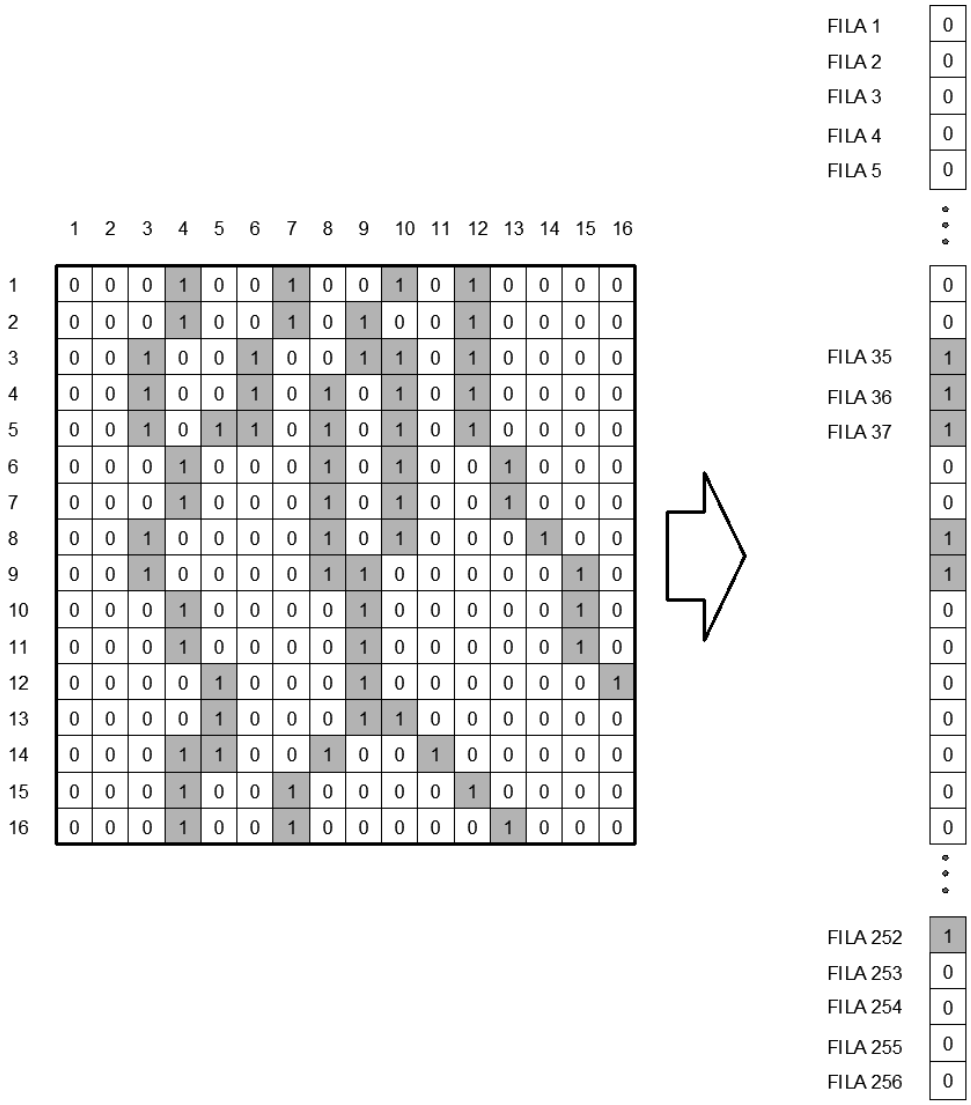


Figura 4.7: Cambio de dimensión de una matriz

A continuación se muestra la tabla 4.2 con el resumen de los datos del conjunto de entrenamiento para la red para MURO-01 de la tabla 4.1.

Datos de Entrada						
Desplazamiento horizontal (Monitor)	1/2700	1/1350	1/675	1/350	1/200	1/125
Carga Vertical	8.66	8.66	8.66	8.66	8.66	8.66
Cuantía de la sección de los elementos de confinamiento.	0.0056	0.0056	0.0056	0.0056	0.0056	0.0056
Relación ancho/alto del muro	1.11	1.11	1.11	1.11	1.11	1.11
Espesor t	0.13	0.13	0.13	0.13	0.13	0.13
f'm	78.9	78.9	78.9	78.9	78.9	78.9
Datos de Salida						
Agrietamientos	(*)	(*)	(*)	(*)	(*)	(*)
Fuerza (respuesta del muro)	10.82	19.80	18.62	24.29	24.24	12.35

Tabla 4.2: Datos de entrada y salida de la red

(*) Modelo numérico del muro en secuencias de ceros y unos considerado en el modelo discreto de 2cm x 2cm.

Finalmente se crea la arquitectura de red neuronal para el problema propuesto (figura 4.8). Esta red aprenderá a través de sus pesos, los respectivos agrietamientos, así como la curva de comportamiento del muro. Los detalles del modelo de red neuronal elegido se detallan en la siguiente sección.

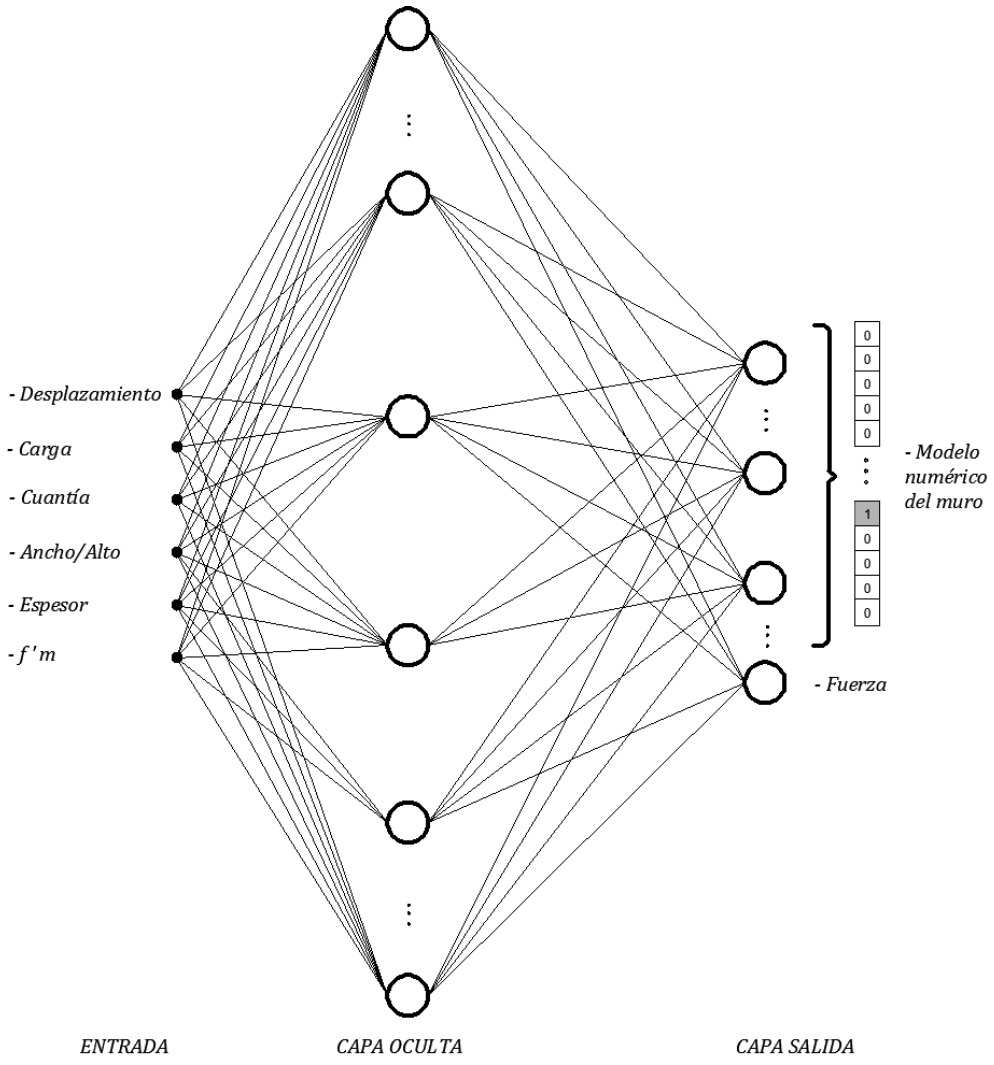


Figura 4.8: Arquitectura de la red neuronal a usar

4.4. Creación de una Red Neuronal Feedforward

Para crear una instancia de una red neuronal del Toolbox de MATLAB se usa la función feedforwardnet:

Sintaxis:

```
RED = FEEDFORWARDNET([S1 S2 … SN], BTF)
```

Donde:

RED: Objeto que representa la arquitectura de la red neuronal a construir.

[S1 S2 … SN]: vector con los números de neuronas en cada capa oculta.
BTF: Función de entrenamiento (ver anexo 02).

Para el presente trabajo se propuso una red de una capa oculta con 33,140 neuronas y se usa la función de entrenamiento traingdx (Gradiente descendente con momento & aprendizaje adaptativo), la cual fue ideal para el tipo de problema a resolver debido a la relativamente rápida convergencia en el aprendizaje.

```
red=feedforwardnet(33140,'traingdx');
```

A continuación se definen los parámetros que definen las funciones de transferencia para las capas oculta y de salida:

```
red.layers{1}.transferFcn='logsig';
red.layers{2}.transferFcn='logsig';
```

4.5. Configuración de la Red Neuronal Feedforward

Configura las entradas y salidas de la red para un mejor rendimiento en el entrenamiento.
Sintaxis:

```
RED = CONFIGURE(RED, P, T)
```

P: Matriz de entrada, cuyas columnas representan los vectores de entrada a la red.
T: Matriz de salida, cuyas columnas representan los vectores correspondientes a los patrones de agrietamientos a aprender.
A continuación se establecen las propiedades de la red para el posterior entrenamiento:

La cantidad máxima de épocas permitidas

```
red.trainparam.epochs=5000;
```

El error o rendimiento tope para la convergencia

```
red.trainparam.goal=0.01;
```

La cantidad máxima de validaciones realizadas

```
red.Trainparam.max_fail=1000;
```

La tasa de aprendizaje cuyo valor debe ser positivo. Un valor demasiado grande puede ocasionar que se oscile alrededor del mínimo sin alcanzarlo. Por otro lado, un valor pequeño puede alcanzar un mínimo de forma estable, pero

```
red.trainparam.lr=0.00001;
```

El incremento para la tasa de aprendizaje.

```
red.trainparam.lr_inc=1.1;
```

La constante de momento, la cual es la encargada de brindar inercia a la búsqueda del mínimo, a fin de evitar que se quede en un mínimo local poco profundo:

```
red.trainParam.mc=0.9;
```

El decremento para la tasa de aprendizaje

```
red.trainparam.lr_dec=0.05;
```

La gradiente mínima para la convergencia

```
red.trainparam.min_grad=0.0001;
```

Configuración de la red

```
red=configure(red,P,T);
```

4.6. Entrenamiento de la Red

Para el entrenamiento de la red se usa la función train.

Sintaxis:

```
TRAIN(RED,P,T)
```

Donde:

RED: es la instancia de la red creada en el punto 4.4.

P y T: matrices de entrada y salida respectivamente. Las columnas de la matriz P se encuentran asociadas uno a uno con las columnas de la matriz de salida T.

Para el presente problema se encontró que la red óptima está conformada por una capa oculta, cuya cantidad de neuronas en dicha capa debe ser el doble de las neuronas de salida. Esto se pudo comprobar en la práctica, donde para números de neuronas cercanos al número de salidas o muy superiores al doble de neuronas de la capa de salida, la calidad del aprendizaje decae significativamente.

La siguiente línea inicia el entrenamiento en la línea de órdenes del MATLAB:

```
[red,tr]=train(red,P,T);
```

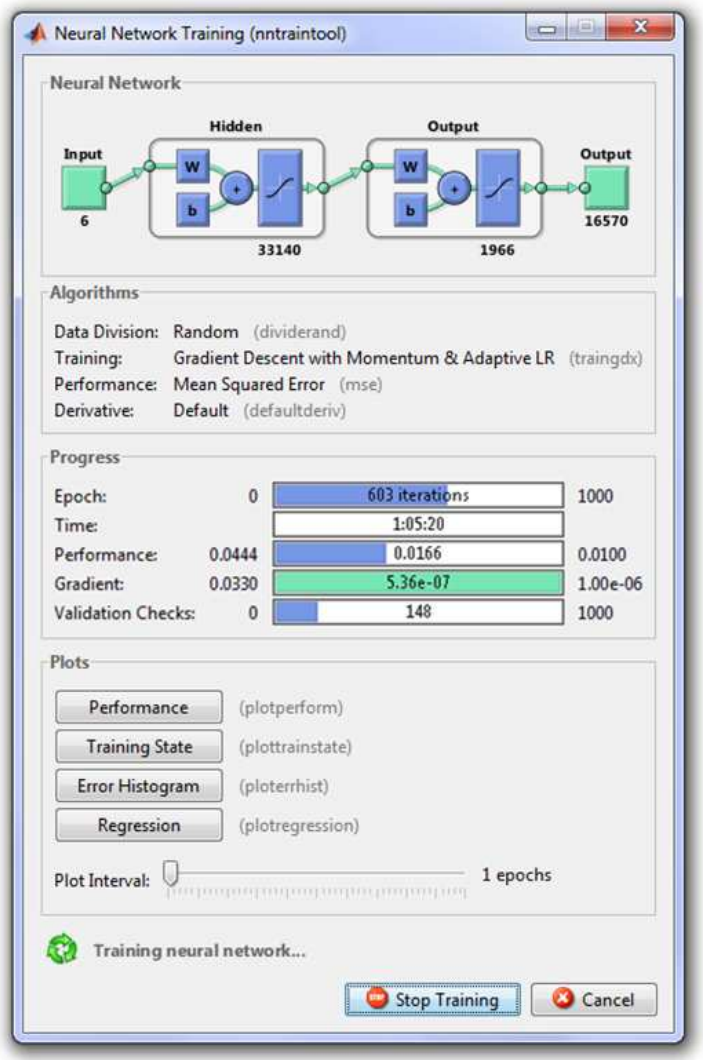


Figura 4.9: Monitoreo del entrenamiento de la red

La ventana de monitoreo mostrada en la figura 4.9 permite revisar la cantidad de épocas, tiempo transcurrido, rendimiento, gradiente de descenso y las validaciones realizadas al aprendizaje de la red. El proceso se detendrá cuando se alcance el máximo o el mínimo de cualquiera de los parámetros de monitoreo.

La siguiente figura muestra cómo los errores evolucionan durante las iteraciones realizadas (épocas). Se debe notar que el entrenamiento mejora notoriamente entre las épocas 50 y 150.

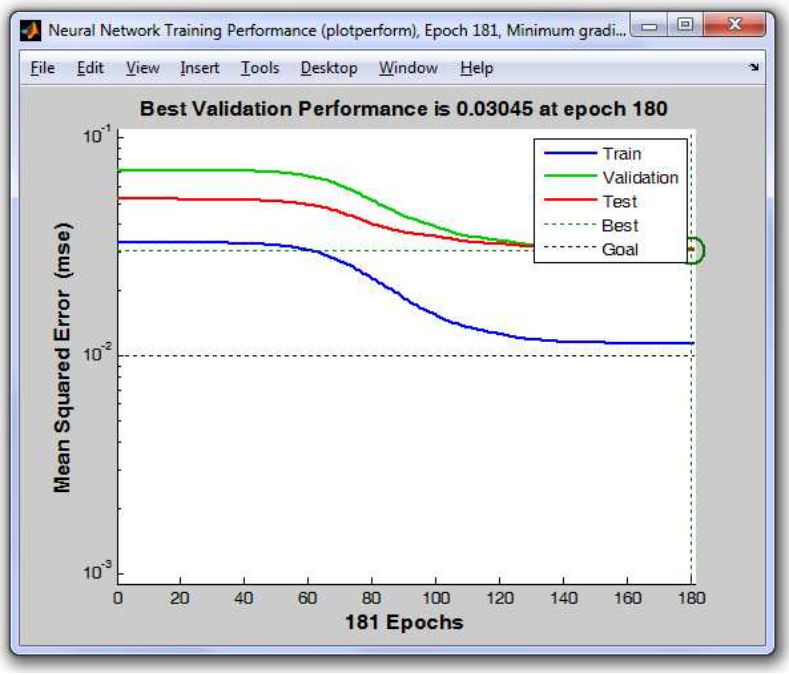


Figura 4.10: Progreso del aprendizaje durante las épocas

La siguiente figura muestra cómo la gradiente alcanza su valor más bajo en la época 151, además durante ese lapso se incremente aceleradamente la tasa de aprendizaje a partir de la época 100.

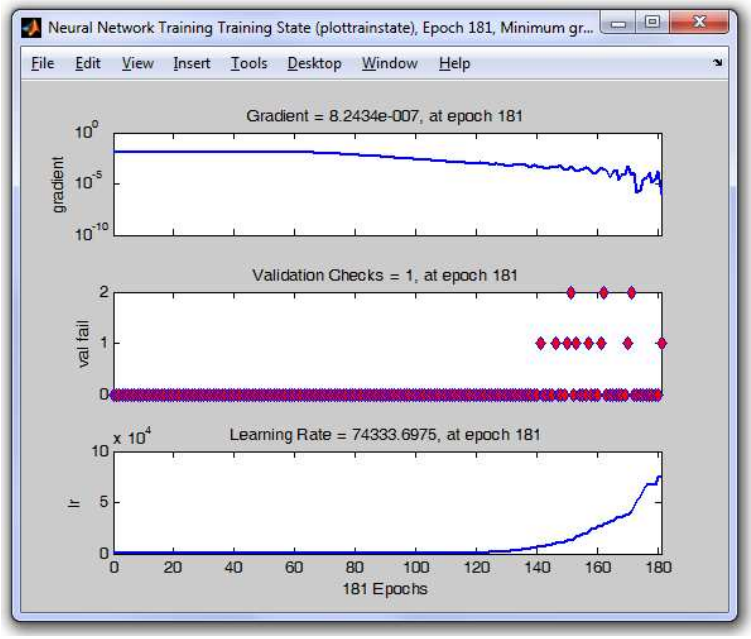


Figura 4.11: Monitoreo de la convergencia y progreso de la tasa de aprendizaje

4.7. Simulación

Una vez que la red ha sido entrenada y los pesos han sido calculados, se procede a evaluar el correcto aprendizaje. En este caso se evaluará la capacidad de la red neuronal de reconocer los datos de entrada usados para el entrenamiento usando la función de MATLAB indicada a continuación:

Sintaxis:

```
SIM(red,P)
```

A continuación se evalúan los diferentes datos asignados a la red:

Deformación lateral 1/2700

Se evalúa la primera columna de la matriz de datos de la red, la cual corresponde al muro con deformación 1/2700.

```
>>a=sim(red,P(:,1));
```

El valor devuelto por SIM es el valor predicho por la red para el agrietamiento en una secuencia de ceros y unos como vector columna. El siguiente paso es redimensionar estos datos para reconstruir el muro de su forma vectorial a su forma matricial:

```
>>b=reshape(a,122,135);
```

La función RESHAPE convierte la secuencia de ceros y unos dispuestos como vector a una matriz con las dimensiones proporcionales a las dimensiones del muro.

Finalmente se aplica la visualización en formato de imagen:

```
>> imshow(b)
```

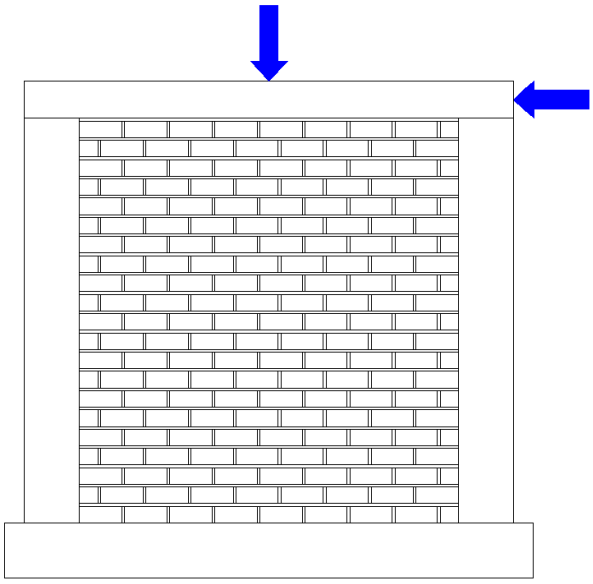
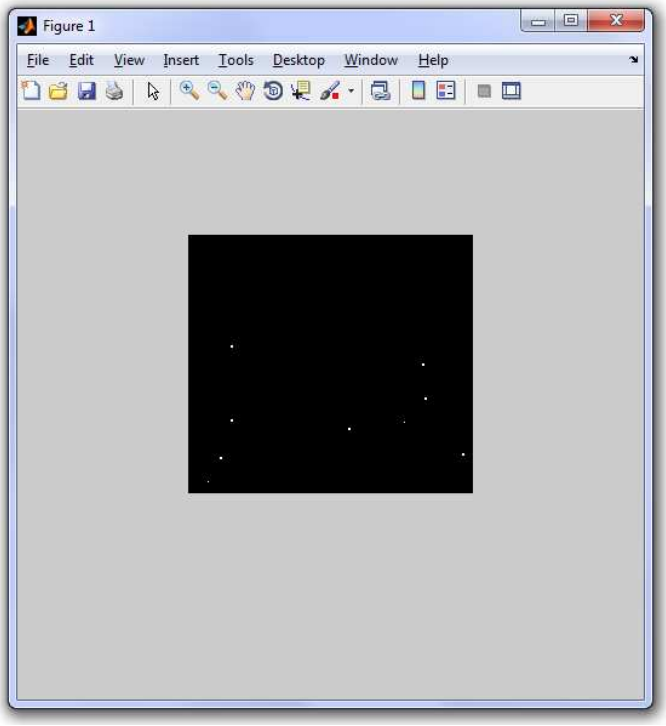
MURO	Datos
	Original: Drift=0.0003024 Fh=10.815
	>>a=sim(red,P(:,1)); >>b=reshape(a,122,135); >> imshow(b) Predicción: Fh=12.3874

Figura 4.12: Simulación para la distorsión de 1/2700

Deformación lateral 1/1350

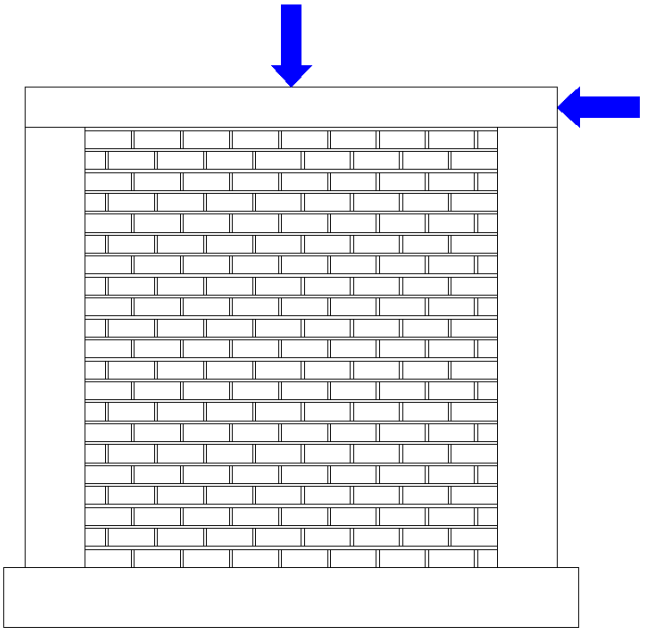
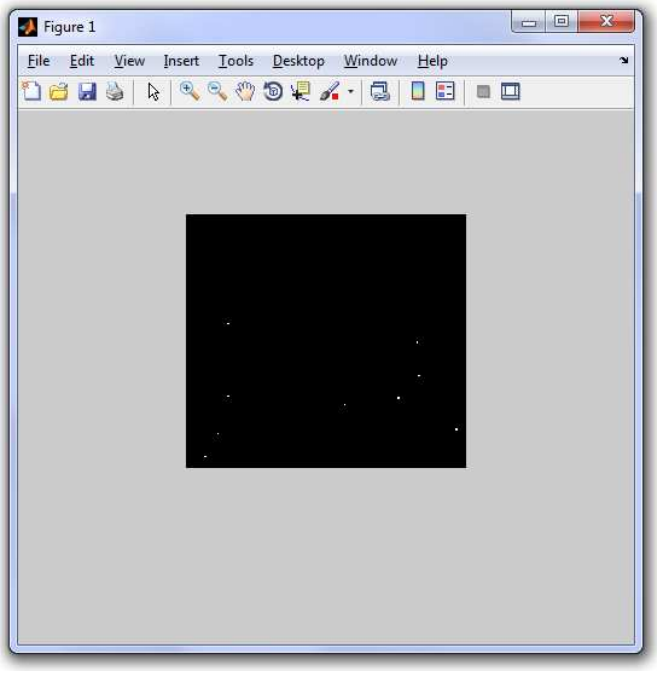
MURO I	Datos
	Original: Drift= 0.000774 Fh= 19.796
	>>a=sim(red,P(:,2)); >>b=reshape(a,122,135); >> imshow(b) Predicción: Fh=18.7025

Figura 4.13: Simulación para la distorsión de 1/1350

Deformación lateral 1/675

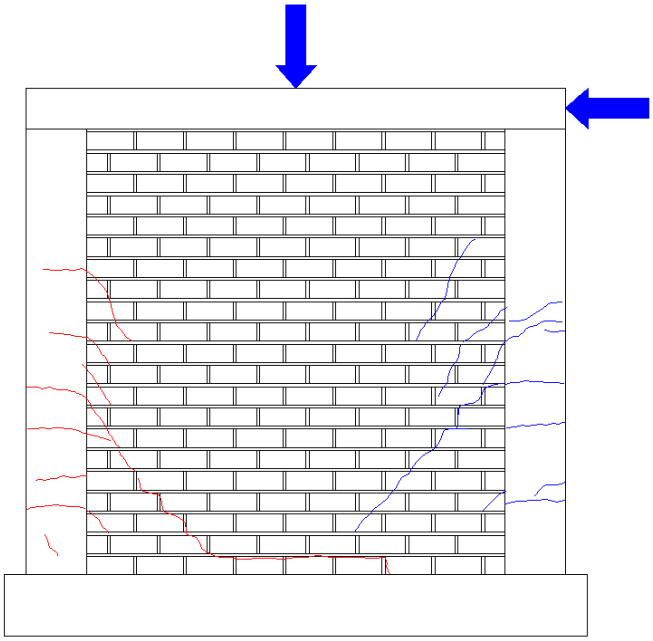
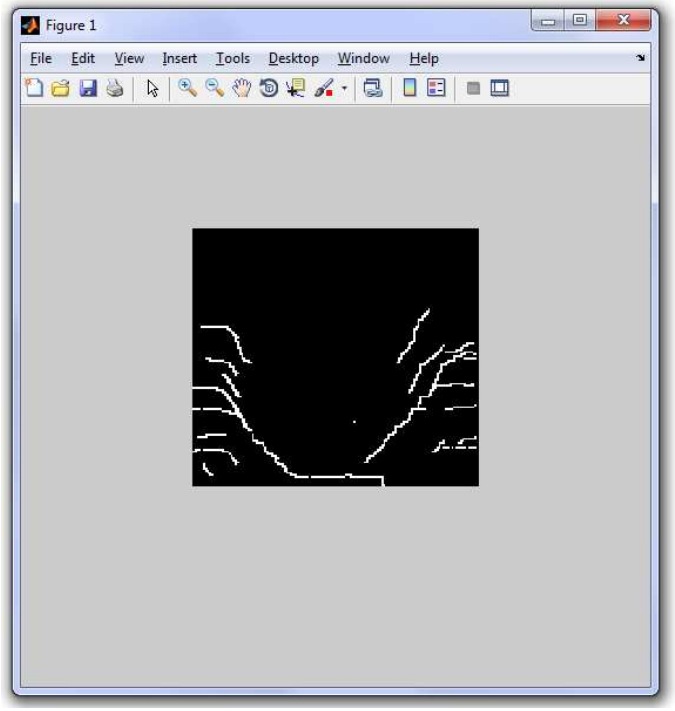
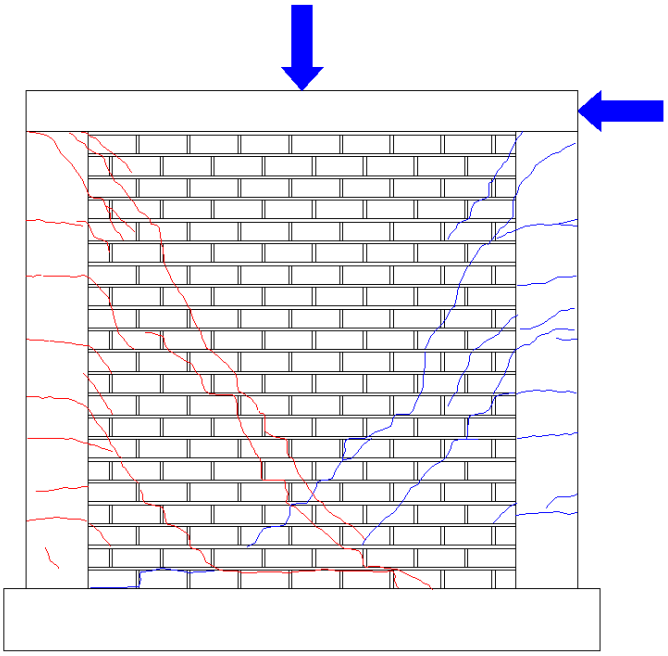
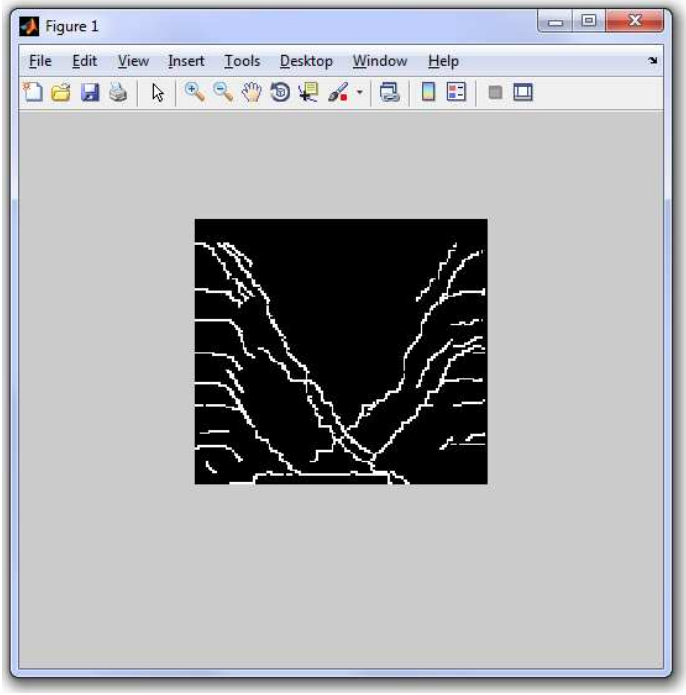
MURO I	Datos
	Original: Drift = 0.00143 Fh=18.617
	>>a=sim(red,P(:,3)); >>b=reshape(a,122,135); >> imshow(b) Predicción: Fh=18.7025

Figura 4.14: Simulación para la distorsión de 1/675

Deformación lateral 1/350

MURO I	Datos
	Original: Drift = 0.00256 Fh=24.289

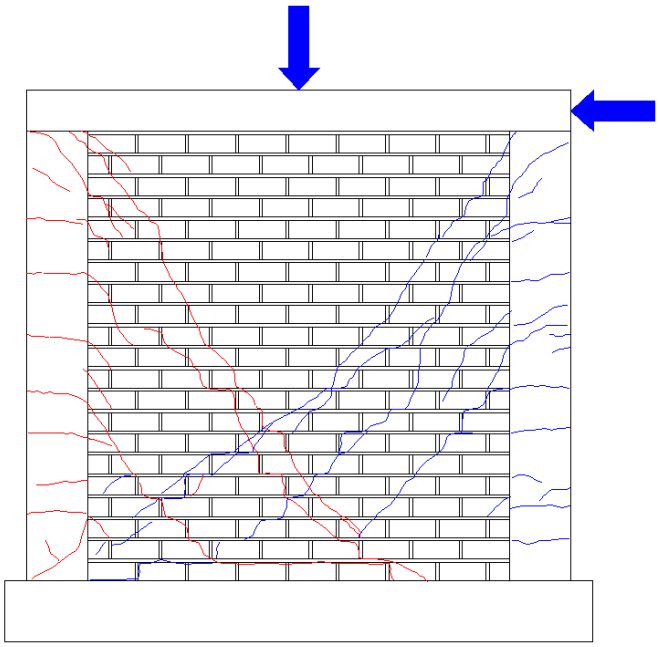


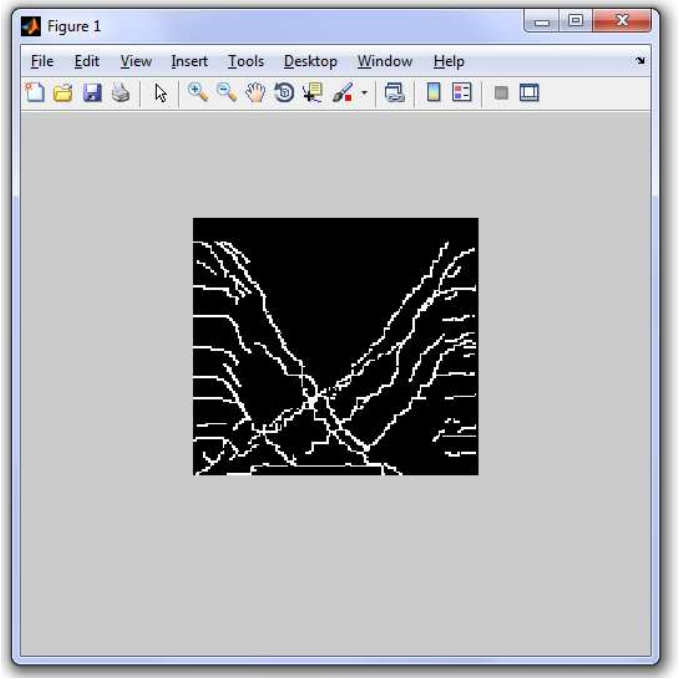

```
>>a=sim(red,P(:,4));
>>b=reshape(a,122,135);
>> imshow(b)
```

Predicción:
Fh=24.289

Figura 4.15: Simulación para la distorsión de 1/350

Deformación lateral 1/200

MURO I	Datos
	Original: Drift = 0.00488 Fh=24.24



>>a=sim(red,P(:,5));

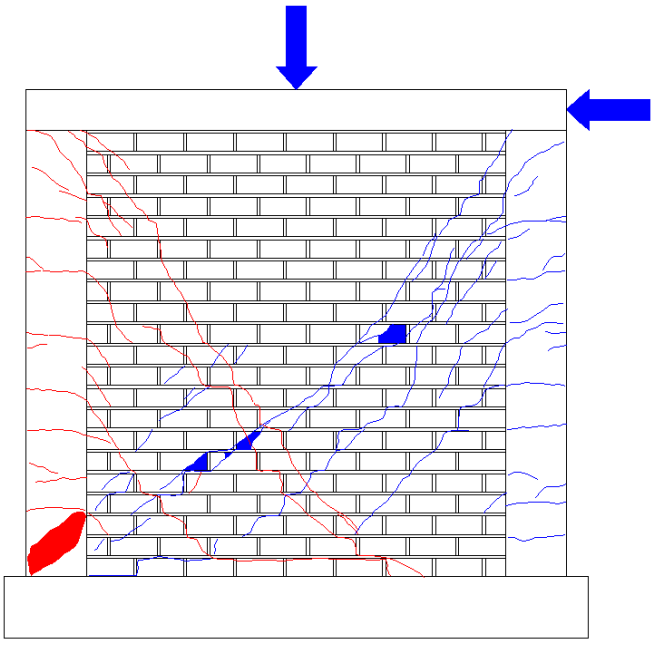
>>b=reshape(a,122,135);

>> imshow(b)

Predicción:
Fh=24.289

Figura 4.16: Simulación para la distorsión de 1/200

Deformación lateral 1/125

MURO I	Datos
	Original: Drift = 0.00743 Fh=12.35

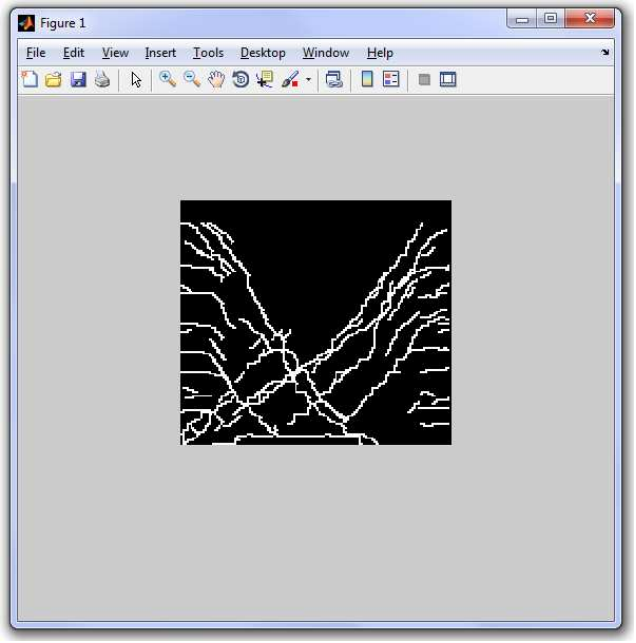
	>>a=sim(red,P(:,6)); >>b=reshape(a,122,135); >> imshow(b) Predicción: Fh=12.3874

Figura 4.17: Simulación para la distorsión de 1/125

Finalmente se muestra un gráfico de comportamiento del muro, se puede observar la similitud entre la información del experimento y el resultado de la simulación:

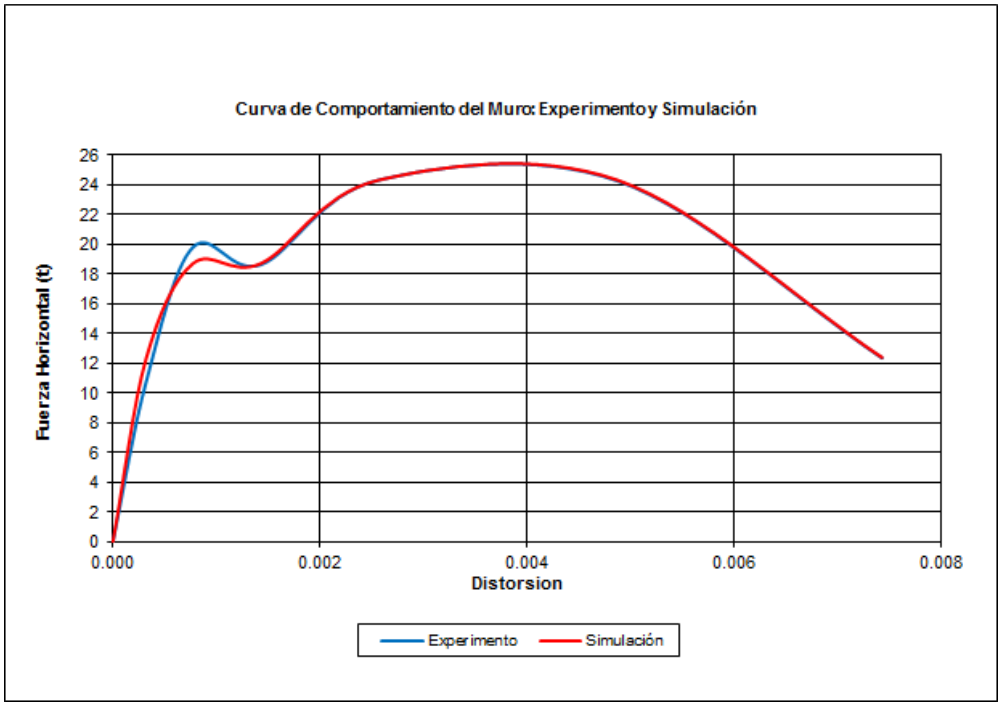


Figura 4.18: Curva de comportamiento del Muro

5. Conclusiones y Recomendaciones

5.1. Conclusiones

La gran capacidad de la red neuronal para resolver problemas de naturaleza no lineal ha permitido resolver con éxito la predicción de grietas y la respuesta de muros sometidos a desplazamientos laterales controlados. Este problema no hubiera podido resolverse analíticamente con métodos convencionales.

Se ha probado y ensayado Redes Neuronales considerando varias capas ocultas, para determinar de esta manera el número adecuado que permita la convergencia y al mismo tiempo utilice la menor cantidad de recursos de computadora (memoria RAM). En la presente tesis se determinó que con una sola capa oculta era posible obtener buenos resultados.

Se ha comprobado que la cantidad de neuronas en la capa oculta deberá ser aproximadamente el doble de la cantidad de salidas deseadas para obtener resultados aceptables. Además se observó que alejarse de esta cantidad hace que el aprendizaje sea ineficiente.

Entre todos los algoritmos de entrenamiento se escogió los métodos de segundo orden. Los cuales se basan en el método de gradiente conjugado. Estas técnicas al usar la segunda derivada pueden inducir la dirección de los mínimos usando la curvatura en lugar de la pendiente, siendo de esta manera más eficiente que los métodos de gradiente descendente.

Una buena opción para ser utilizada como algoritmo de segundo orden para la presente tesis fue el Levenberg-Marquart (LM). Este es el método por defecto del Toolbox de redes neuronales del Matlab. Converge de manera óptima a la solución en la superficie de error, pero requiere de bastantes cálculos entre iteraciones, lo que generalmente produce desbordamientos de memoria, ya que sus requerimientos aumentan exponencialmente con la cantidad de neuronas en las capas ocultas. Esta opción tuvo que ser rechazada debido a que sobrepasaba los límites de hardware disponible para la presente tesis.

La siguiente función de entrenamiento, que mejor se adaptó a las dimensiones del presente problema fue la función “Gradient Descent with Momentum & Adaptive LR”. Sin embargo, debido a que es una de las funciones más rápidas y eficientes, necesitó un enorme consumo de memoria y esfuerzo computacional. Se observó una fuerte caída de

velocidad de procesamiento de una estación de trabajo con sistema operativo de 64 bits y 16GB de RAM (el consumo de recursos fue tan demandante que se requirió disminuir la cantidad de elementos del modelo numérico del muro, sacrificándose algo de resolución en la representación del agrietamiento).

La función de transferencia elegida en todas las capas fue la "sigmoidal". Esta función es ideal cuando se requieren obtener valores entre cero y uno. El uso de otras funciones sólo dificultó el aprendizaje debido que en algunos casos se obtuvieron valores negativos que no tienen sentido para el problema estudiado.

El ingreso de datos de entrada redundantes, demora el entrenamiento y origina un incremento en el número de "Epochs". Por esa razón se realiza una depuración de las entradas para evitar el sobre entrenamiento.

El gráfico de la figura 4.18 muestra una gran coincidencia entre la curva de comportamiento obtenida por la RNA con la curva de comportamiento obtenida de manera experimental. Esto valida los resultados de la red frente al problema planteado.

5.2. Recomendaciones

El número de capas ocultas se utiliza para aumentar la complejidad del mapeo entre las entradas y salidas de la red (entrenamiento supervisado). Este número debe ser obtenido experimentalmente, controlando adecuadamente la cantidad de neuronas en cada una de ellas para evitar el sobre entrenamiento y que la red pierda su propiedad de generalización. Por otro lado, pocas neuronas en la capa oculta pueden resultar insuficientes para el aprendizaje de la red.

Se requiere utilizar computadoras con procesamiento en paralelo o supercomputadoras para trabajar con mayor cantidad de capas ocultas y salidas, debido a las millones de conexiones que se generan. Se ha tenido problemas para procesar modelos más refinados y también cuando se han considerado más capas ocultas.

El elegir un factor de aprendizaje adecuado es un tema bastante importante. Es decir, si se asigna un factor de aprendizaje muy alto, esto puede acelerar el aprendizaje, pero puede evitar que la red aprenda y empiece al oscilar alrededor del mínimo sin alcanzarlo.

6. Anexos

6.1. Fotos del Ensayo

Se adjuntan algunas fotos que ayudan a comprender el tipo de ensayo realizado sobre los muros.

Muro I

Foto 1: Montaje e instrumentación del muro a ensayar

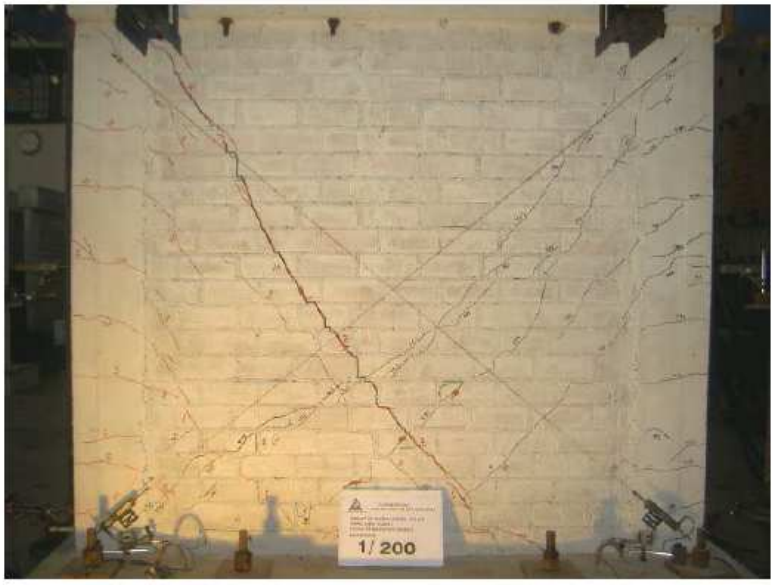


Foto 2: Agrietamientos producidos para la distorsión angular de 1/200

Foto 3: Detalle de agrietamientos y de la posición del sensor

6.2. Tablas de Referencia

Función	Algoritmo
trainlm	Levenberg-Marquardt
trainbr	Regularización Bayesiana
trainbfg	Quasi Newton
trainrp	Backpropagation adaptativo
trainscg	Gradiente conjugado escalado
traincgb	Gradiente conjugado con reinicios de Powell/Bale
traincgf	Gradiente conjugado de Fletcher-Powell
traincgp	Gradiente conjugado de Polak-Ribiére
trainoss	Secante de un paso
traingdx	Gradiente descendente con tasa de aprendizaje variable
traingdm	Gradiente descendente con momento
traingd	Gradiente descendente

Tabla 6.1: Algoritmos de aprendizaje incluidos en el Toolbox de Matlab

Función	Descripción
fopen	Permite abrir un archivo de texto para lectura o escritura.
feof	Devuelve el estado de fin de archivo. 0 si no encuentra el fin de archivo o 1 en caso de no encontrarlo.
fclose	Cierra un archivo abierto con la función fopen.
strcat	Concatena dos o más cadenas de texto.
fgets	Lee una línea de un archivo, conservando el carácter de fin de línea.
DibujarLinea	Dibuja una línea de 1s en una matriz rectangular de 0s. Se debe proveer las coordenadas inicial y final de la línea, además de la matriz rectangular de ceros sobre la cual se realizará el trazado.
transforma	Transforma una fila de un archivo leída como cadena a coordenadas X, Y de un punto de grieta.
reshape	Permite redimensionar una matriz siempre que se conserve la misma cantidad de sus elementos. En el caso del presente estudio, la transformación se realiza de una matriz rectangular que representa al muro a un vector que podrá ser usado como patrón de aprendizaje. Luego se realizará el proceso inverso en la simulación, cuando se obtengan los patrones de 0s y 1s a partir de una entrada evaluada, para obtener el agrietamiento simulado.
flipud	Refleja horizontalmente los elementos de una matriz de tal manera que los elementos de la parte inferior se encuentren en la parte superior y viceversa. Esta función resulta muy útil al momento de graficar la matriz de 0s y 1s, dado que la función DibujarLinea usa coordenadas cartesianas con el eje X positivo hacia la derecha y el eje Y positivo hacia abajo.
clear	Limpia variables de memoria especificadas como un argumento. Esto es importante al momento de ahorrar memoria antes de ejecutar los

	procedimientos de entrenamiento.
round	Redondea valores numéricos al entero más próximo.
load	Carga los datos de un archivo de texto y los vuelca en una matriz.

Tabla 6.2: Funciones usadas en Matlab

Bibliografía

[1] M. E. Williams, «Using Neural Networks to Position Live Loads on Bridge Piers,» University of Florida, Florida, 2000.

[2] A. Attal, Development of Neural Network Models for Prediction of Highway Construction Cost and Project Duration, Ohio: Russ College of Engineering and Technology, 2010.

[3] ACI Committee 211, Standard Practice for Selecting Proportions for Normal, Heavyweight, and Mass Concrete, Detroit: American Concrete Institute, 1991.

[4] E. Yildiz, «Lateral Pressures in Rigid Retaining Walls: A Neural Network Approach,» The Middle East Technical University, Turkey, 2003.

[5] L. R. O. de Lima, P. C. G. da S. Vellasco, S. A. L. de Andrade, J. G. S. da Silva y M. M. B. R. Vellasco, «Neural Networks Assesments of Beam-to-Column Joints,» vol. XXVII, nº 3, pp. 314-324, 2005.

[6] H. C. Cas y D. R. Parhi, «Application of Neural Network for Fault Diagnosis of Cracked Cantiliever Beam,» *World Congress on Nature and Biologically Inspired Computing,* vol. 1, nº 5393733, pp. 1303-1308, 2009.

[7] A. Stanic, V. Sigmud y I. Guljas, «Seismic Capacity of Structural Elements Using Neural Networks,» *13th World Conference on Earthquake Engineering,* vol. I, nº 403, pp. 1-10, 2004.

[8] M. H. Beale, M. T. Hagan y H. B. Demuth, Neural Network Toolbox™ 7 - User's Guide, Massachussets: The MathWorks, 2011.

[9] F. Rosenblatt, Principles of Neurodynamics, Washington: Spartan Press, 1961.

[10] F. Lara, Derivación Matricial, Bogotá: Fundación Universitaria KOnrad Lorenz, 2007.

[11] P. D. Wasserman, Advanced Methods in Neural Computing, New York: John Wiley & Sons, 1993.

[12] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning Representations by Back-propagation Errors,» *Nature,* vol. 323, pp. 533-536, 1986.

[13] R. A. Jacobs, «Increased Rates of Convergence Through Learning Rate Adaptation,» *Neural Networks,* vol. 1, nº 4, pp. 295-307, 1988.

[14] C. M. Bishop, Neural Networks for Pattern Recognition, USA: Oxford University Press, 1995.

[15] C. Zavala, L. Chang y E. Arellano, Factibilidad de uso de Mallas Electrosoldadas en Elementos de Confinamiento de Muros de Albañilería, Lima: CISMID, 2003.

[16] A. S. Bartolomé, Construcciones de Albañilería-Comportamiento Sísmico y Diseño Estructural, Lima: Fondo Editorial de la PUCP, 1998.