Trabajo Final de Licenciatura
en Ciencias de la Computación

Procesamiento de Imágenes Binoculares Mediante Redes Neuronales

Autor: *Flavio D. Garcia*

Director: *Francisco A. Tamarit*

Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba
Córdoba, Septiembre de 2002

Dedico este trabajo
a mis padres.

Agradecimientos

Agradezco a mi director Pancho, por la ayuda y la paciencia brindadas y por enseñarme a redactar un poco mejor.

A Sergio, Nico y Damian.

A todos mis compañeros y amigos.

A las chicas de despacho de alumnos.

Finalmente quiero agradecer muy especialmente a los docentes de la Fa.M.A.F. que a lo largo de toda la carrera, además de formarme, me hicieron sentir entre amigos.

Resumen

En este trabajo estudiaremos el uso de redes neuronales tipo feed–forward como una solución al problema de la estéreo visión. Realizaremos, mediante simulación, una evaluación de distintas configuraciones de redes neuronales y distintos tamaños para el conjunto de entrenamiento. Previamente haremos una pequeña introducción a la teoría de redes neuronales artificiales y a la percepción de profundidad a partir de imágenes binoculares. También describiremos un método, denominado "correlation based", que soluciona de manera algorítmica el problema abordado.

Abstract

This work covers de use of feed–forward neural networks to solve the stereo vision problem. We evaluate, by simulation, the influence of the training set size and network topology on the generalization error of the network. Before this analisis, we make an introduction to the theory of neural networks and depth perception, in order to give the reader a basic theoretical knowledge. We also talk about the "correlation based" method to solve de stereo vision problem algorithmicaly.

Índice general

Índice de figuras

Capítulo 1

Introducción

La naturaleza, en múltiples ocasiones, ha encontrado solución a grandes problemas de optimización. Los seres vivos realizan eficientemente tareas que computacionalmente son extremadamente costosas. Una de éstas es la percepción de profundidad, o distancia a la que se encuentra un objeto de su observador. La mayoría de los animales realizan esta tarea en tiempo real y con gran precisión, lo que motiva el uso de redes neuronales para emularlos.

La estéreo visión (stereo vision) es el proceso en el que se transforman dos imágenes planas (binoculares) de una escena, en una reconstrucción tridimensional de la misma, recuperándose sus profundidades, las cuales se perdieron cuando se proyectaron sobre los planos de cada imagen. Esto se puede realizar gracias a la disparidad de ambas imágenes y mediante triangulación. Este proceso tiene múltiples aplicaciones, como modelado del terreno, simulación de vuelo, navegación de robots, reconocimiento de rostro, entre muchos otros. El cálculo algorítmico de este problema es costoso y posee otros inconvenientes, como la validación de los datos y la interpolación de los puntos perdidos.

El objetivo de este trabajo es lograr una primera aproximación entrenando una red neuronal que realice dicho proceso, aprovechando la capacidad de interpolación que éstas poseen. La ventaja de realizar este proceso con redes neuronales es que

éstas poseen una estructura sumamente paralelizable. Además son potencialmente implementables en hardware (neural chips).

Es importante destacar que se asume como hipótesis el hecho de que las imágenes están acotadas *epipolármente*, para reducir así el espacio de búsqueda. Esto significa que un punto en una de las imágenes puede estar desplazado horizontal pero no verticalmente en la otra imagen. Para lograr ésto existen numerosos algoritmos que permiten el calibrado de las cámaras.

Además asumimos que los desplazamientos horizontales o *disparidades* son pequeños y locales. Esto significa que si (x, y) son las coordenadas de un punto arbitrario en la imagen izquierda, el mismo punto se encuentra en el entorno $(x - \delta_l \ldots x + \delta_h, y)$ alrededor del punto (x, y) de la imagen derecha. Ésto es una aproximación, ya que las disparidades dependen de las distancias a los objetos y de la configuración de las cámaras.

Comenzaremos realizando una breve introducción a las redes neuronales artificiales, siempre concentrándonos en la parte de la teoría de redes neuronales utilizada en este trabajo y tratando de mantener su la analogía con la biología. Luego explicaremos el problema de la estéreo visión y uno de los métodos algorítmicos más tradicionales para resolverlo, denominado "correlation based method". Intentaremos obtener una versión acabada del mismo, validando los datos producidos y realizando algún tipo de postprocesamiento que mejore los resultados. Además explicaremos como generar el ambiente 3D, recuperando las coordenadas de cada punto en la escena original. Luego describiremos el enfoque dado con redes neuronales a este problema específico, donde trataremos de encontrar aquella arquitectura que logre el mejor desempeño para resolver el problema abordado. Para ello realizaremos un gran número de simulaciones y analizaremos los errores producidos. Finalmente compararemos los resultados producidos, tanto por el método algorítmico como por la red neuronal.

Capítulo 2

Redes Neuronales

2.1. Introducción

Como lo mencionamos en el Capítulo 1, los seres vivos poseen gran habilidad para realizar algunas tareas que algorítmicamente serían muy costosas o difíciles de realizar. Un ejemplo de ésto es la destreza de un insecto que vuela por una habitación sin chocar con las paredes... (omitamos por favor del ejemplo a las polillas, que no son muy hábiles en esta tarea)... Intuitivamente parece poco probable que el insecto calcule la disparidad de lo que ven sus ojos, con esta información genere un mapa con las coordenadas de todos los objetos que lo rodean, calcule su ecuación de movimiento, la derive y calcule la fuerza a aplicar en sus alas para modificar su trayectoria convenientemente. Esto nos hace pensar que hay otra forma de realizar estas tareas.

Las redes neuronales intentan dar un enfoque distinto al modo de resolver los problemas del que le dan los métodos tradicionales. Estas no se programan, se entrenan. El entrenamiento consiste en presentarle a la red un conjunto de patrones de entrada-salida que de alguna forma se conoce a priori. Este conjunto de patrones entrada-salida se denomina *conjunto de entrenamiento*. Luego, se desea que la red neuronal infiera la regla que rige estos patrones y sea capaz de *generalizar*. O sea, de

obtener la salida correcta ante patrones de entrada que no pertenecen al conjunto de entrenamiento.

2.2. Neurofisiología del Cerebro

Las redes neuronales tratan de imitar la forma en la que el cerebro procesa la información. Éste está compuesto por células especializadas llamadas *neuronas*. Un ser humano adulto posee aproximadamente 10^{12} neuronas, cada una de las cuales se conecta en media con otras 10^5 neuronas, lo que le provee de una gran capacidad de paralelización. En la Figura 2.1 se puede apreciar un esquema simplificado de neurona[Brown & Benchmark]. En ella se puede observar como la morfología de

Figura 2.1: Diagrama de una Neurona

una neurona comprende tres elementos principales: el soma, o cuerpo principal; las dendritas, o terminaciones de la neurona que actúan como contactos funcionales de entrada con otras neuronas; y el axón, una rama más larga que será la encargada de conducir el impulso nervioso y que finaliza también en diversas ramificaciones. La comunicación entre neuronas se realiza a través de las sinapsis, que son los puntos

de conexión entre las fibras terminales del axón de una neurona y una dendrita de otra. El impulso nervioso producido por una neurona se propaga por el axón y, al llegar a un extremo, las fibras terminales pre-sinápticas liberan compuestos químicos llamados neurotransmisores. Los neurotransmisores se liberan en la membrana celular pre-sináptica y alteran el estado eléctrico de la membrana post-sináptica (ver Figura 2.2). En función del neurotransmisor liberado, el mecanismo puede resultar excitador o inhibidor para la neurona *receptora*. En el soma de una neurona se integran todos los estímulos recibidos a través de todas sus dendritas. Si como resultado se supera un potencial de activación, la neurona se "dispara", generando un impulso que se transmitirá a través del axón. Sino, se mantiene en reposo.

Figura 2.2: Esquema de liberación de neurotransmisores en la sinapsis

La repercusión de un estímulo nervioso en el estado excitatorio de la neurona receptora no es constante y se modifica con el tiempo en el proceso de aprendizaje.

A ésto se lo denomina *plasticidad sináptica*. Es de esperar, por ejemplo, que una neurona ubicada en la parte del cerebro dedicada a la visión, sea capaz de procesar la información o estímulos provenientes del nervio óptico e ignorar aquellos estímulos que provienen de otros centros receptivos. Para construir un modelo formal y simplificado se rescataron los siguientes aspectos:

- La neurona posee sólo dos estados: *excitatorio* o *de reposo*
- Existen dos tipos de sinapsis: *excitatorias* e *inhibidoras*
- La neurona es un dispositivo integrador, ya que suma los impulsos que le llegan de las dendritas .
- Cada sinapsis trasmite con mayor o menor intensidad los estímulos eléctricos, de acuerdo a su *Capacidad Sináptica*
- El aprendizaje consiste en modificaciones en las sinapsis.

2.3. El Modelo de McCulloch y Pitts

En 1943 McCulloch y Pitts publicaron un trabajo en el que se hablaba, por primera vez, de las neuronas artificiales y de cómo éstas podrían realizar cálculos lógicos en redes neuronales. El modelo de neurona formal propuesto representa a la neurona como una función booleana de dos estados: en reposo o activada. El potencial post-sináptico de membrana en una neurona arbitraria i se representa por la función h_i, definida como la suma ponderada de los estímulos recibidos a través de las dendritas:

$$h_i = \sum_{j=1}^{N} \xi_j w_{ij} \tag{2.1}$$

Aquí, las cantidades $w_{ij} \in \mathbb{R}$ modelan la eficacia sináptica entre la neurona i y la neurona j y la variable ξ_j que modela el estado de la neurona conectada presinápticamente a la j-ésima dendrita. Es importante destacar que las cantidades

w_{ij} pueden tomar tanto valores positivos como negativos, modelando el carácter excitatorio e inhibitorio de las conexiones sinápticas biológicas.

El estado de la neurona i se actualiza de acuerdo a la siguiente regla: si en el momento de realizar el cómputo se cumple que $h_i > \theta_i$ entonces diremos que la neurona se activa, y por lo tanto su variable booleana de estado O_i debe tomar el valor $+1$. Si por el contrario $h_i < \theta_i$, entonces la neurona permanecerá en estado de reposo, y O_i debe tomar el valor -1. Matemáticamente ésto se puede expresar de la siguiente forma:

$$O_i = \operatorname{sgn}\left(h_i - \theta_i\right) \ , \tag{2.2}$$

donde $\operatorname{sgn}(x)$ es la función signo. La cantidad θ_i modela el umbral de activación de membrana y la cantidad h_i se denomina usualmente campo local.

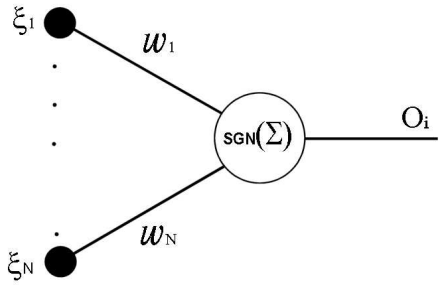


Figura 2.3: Neurona artificial

2.4. Capacidades y Limitaciones

Veamos ahora las capacidades y limitaciones de este modelo simple. Para ilustrar ésto pensemos que queremos construir una pequeña red de neuronas que compute la función booleana AND, cuya tabla de verdad es:

ξ_1	ξ_2	AND
0	0	-1
0	1	-1
1	0	-1
1	1	1

En la figura 2.4 representamos esquemáticamente esta red con dos neuronas de entrada ξ_1 y ξ_2 que no realizan cómputo alguno y una neurona de salida O que debe dar la respuesta correcta. El problema consiste en hallar los valores adecuados de w_1,

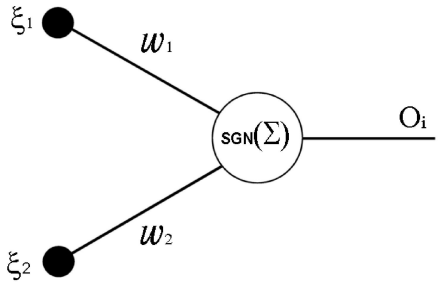


Figura 2.4: Red neuronal artificial para computar la función AND

w_2 y θ. Si pensamos en la función signo y miramos la tabla 2.4, podemos construir el siguiente sistema de ecuaciones:

$$\begin{aligned} -\theta &< 0 \\ w_2 - \theta &< 0 \\ w_1 - \theta &< 0 \\ w_1 + w_2 - \theta &\geq 0 \end{aligned}$$

Es fácil ver que por ejemplo:

$$\theta = 1{,}5$$

$$w_2 = 1$$

$$w_1 = 1$$

son solución del sistema propuesto. Es más, este sistema posee infinitas soluciones. Por lo tanto, la cantidad de configuraciones para calcular la función AND es también infinita.

Pero ésto no es siempre así. Si consideramos la función XOR cuya tabla de verdad es:

ξ_1	ξ_2	XOR
0	0	-1
0	1	1
1	0	1
1	1	-1

debemos resolver el sistema:

$$0 < \theta \tag{2.3}$$

$$w_2 \geq \theta \tag{2.4}$$

$$w_1 \geq \theta \tag{2.5}$$

$$w_1 + w_2 < \theta \ , \tag{2.6}$$

pero al intentarlo, nos encontraremos con que no tiene solución. Para demostrar ésto, multipliquemos la ecuación (2.4) por -1 y luego sumémosle la ecuación (2.6). Obtendremos que $w_1 < 0$. Esto se contradice con la condición $0 < w_1$, que se obtiene por transitividad de (2.3) y (2.5). Esto demuestra que no existe un red simple capaz de resolver el problema de la función XOR. Esto se debe a que éste no es un problema *linealmente separable*.

Se dice que un problema es *linealmente separable* cuando existe un plano en el espacio $\boldsymbol{\xi}$que separa aquellas salidas que deben valer 1 de las que deben valer -1 (plano clasificador).

Para solucionar problemas que no son linealmente separables, lo que haremos es combinar varias redes simples siguiendo ciertas prescripciones. De esta manera realizaremos la clasificación con más de un plano. En la Sección 2.10 hablaremos de como conectar varias redes simples para construir una red neuronal con la habilidad de resolver problemas *no linealmente separables*, como por ejemplo la función XOR.

2.5. Neuronas Continuas

Hasta ahora siempre hablamos de neuronas binarias, pero podemos definir otras familias de modelos neuronales cambiando simplemente la función de activación (que hasta ahora era la función signo). En forma genérica diremos que

$$O_i = g\left(\sum_k \omega_{ik}\xi_k - \theta_i\right) \tag{2.7}$$

donde en principio g puede ser cualquier función real. Existen dos tipos de modelos neuronales que utilizaremos en este trabajo, ambos caracterizados por ser la función g continua.

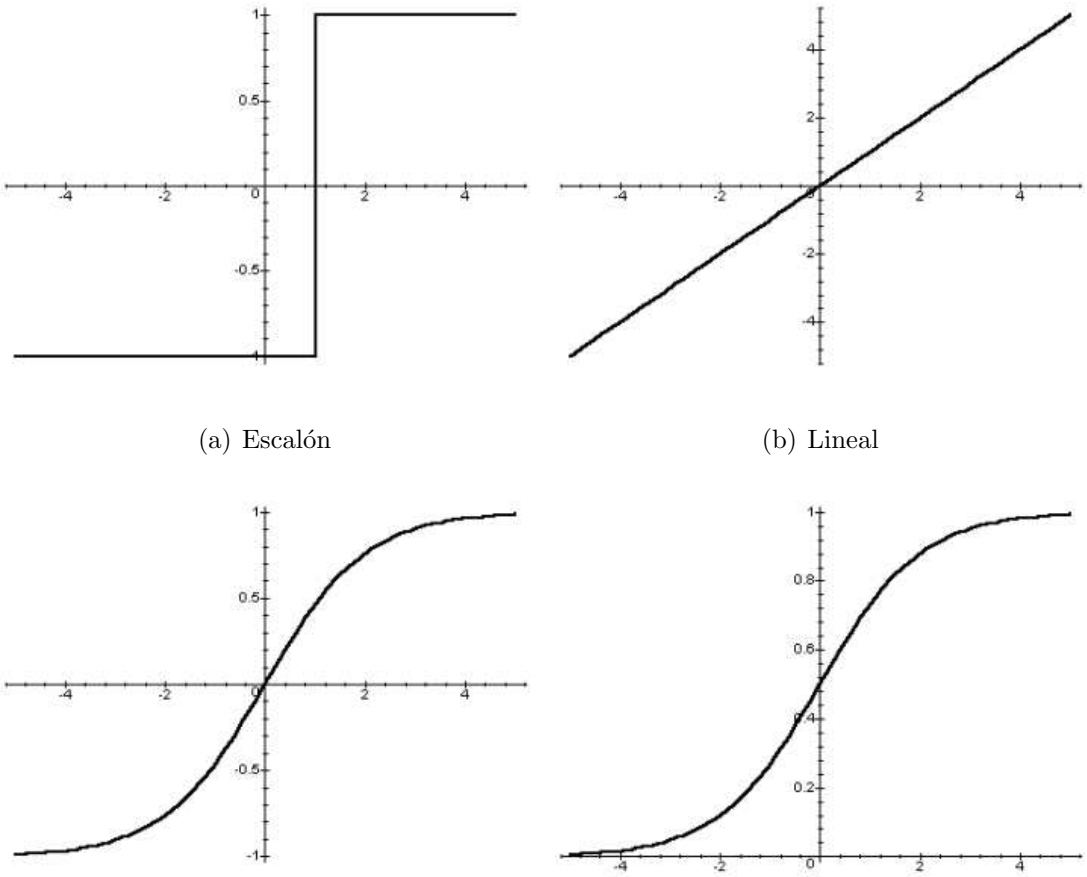
El primero está caracterizado por funciones lineales de la forma:

$$O_i = Ah_i + B \tag{2.8}$$

Veremos más adelante que este tipo de neuronas se usan frecuentemente como neuronas de salida. La segunda familia de neuronas está caracterizado por funciones g sigmoides, que están caracterizadas porque saturan para valores muy altos o muy bajos de su argumento. Este tipo de neuronas son de fundamental importancia porque introducen la no linealidad en el problema, sin la cual la capacidad de la red se ve drásticamente reducida. En este trabajo usaremos siempre neuronas de este tipo salvo para las neuronas de entrada y de salida. En particular, la función que usaremos tiene la forma:

$$g(x) = \tanh(\beta x) \tag{2.9}$$

que veremos más adelante, presenta ciertas ventajas adicionales a la hora de implementar numéricamente la red. El parámetro β se llama ganancia y determina cuán empinada será la curva en la zona de transición. En la figura 2.5 mostramos esquemáticamente diferentes modelos neuronales.



(a) Escalón

(b) Lineal

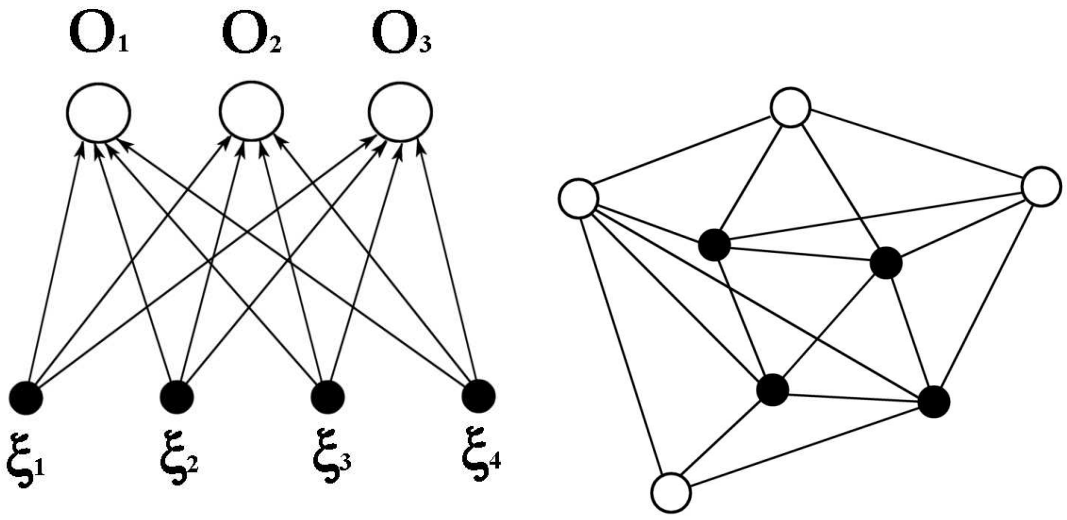
(c) Tangente hiperbólica, $\beta = \frac{1}{2}$

(d) Función de Fermi, $\beta = \frac{1}{2}$

Figura 2.5: Distintas funciones de activación

2.6. Arquitecturas neuronales

Existen básicamente dos tipos de arquitecturas neuronales: las redes *feed–forward* y las redes *recurrentes*. Las primeras, que describiremos a continuación, son también



(a) Perceptron simple (b) Red recurrente

Figura 2.6: Arquitecturas neuronales

llamadas *perceptrones* (el nombre fue introducido por Rosenblatt en 1962), por su similaridad con las neuronas dedicadas a la visión. Éstas se caracterizan por poseer la siguiente estructura (ver Figura 2.6(a)): poseen un conjunto de terminales o neuronas "tontas", que sólo cumplen la función de proveer a la red neuronal de los patrones de entrada. Éstas no realizan ninguna computación y están representadas por puntos negros en la Figura 2.6(a). Luego pueden tener una o más capas de neuronas intermedias, seguidas por una capa final o de salida, donde se leen las respuestas (outputs) de la red, en la figura representadas por círculos. Las neuronas de las camadas intermedias son llamadas a menudo *neuronas ocultas* pues no tienen conexión directa con el mundo exterior, sea a través del input o del output. En este tipo de arquitectura (feed–forward), una neurona sólo se conecta con las neuronas de *la* capa anterior. Por lo tanto no hay interconexiones entre neuronas que se encuentran en la misma capa y tampoco con aquellas en capas subsiguientes. Además, la información se dirige en un solo sentido, o sea, desde la entrada hacia la salida. Por lo tanto, una neurona sólo recibe estímulos provenientes de neuronas en la capa

anterior y sólo los transmite a aquellas en la capa superior. Ya que las conexiones son unidireccionales, las redes feed–forward poseen, por definición, matrices de sinapsis w_{ij} asimétricas, siendo w_{ij} o w_{ji} igual a cero. Si este tipo de redes sólo posee una camada de neuronas, o sea la de salida (los inputs normalmente no son contados como capas de neuronas), son comúnmente llamadas perceptrones simples y constituyen la red neuronal más básica que podemos construir. Como lo mencionaremos en la Sección 2.8, las estructuras neuronales en los seres vivos son, en su mayoría (pero no totalmente), feed–forward.

Por otra parte, las redes recurrentes son más generales que las feed–forward. En éstas pueden existir conexiones entre dos neuronas en ambos sentidos. Este tipo de redes no evoluciona, necesariamente, a un estado final estable para un patrón de entrada constante. Si las conexiones fueran simétricas ($w_{ij} = w_{ji}$), la estabilidad del estado final de la red neuronal estaría garantizada, como sucede en el caso de las redes Hopfield. Éstas son comúnmente empleadas en problemas de memoria asociativa. La Figura 2.6(b), ilustra un esquema de red recurrente donde las neuronas ocultas están representadas como puntos de color negro y las neuronas visibles con círculos. Ya que este tipo de redes no poseen una estructura jerárquica, la elección de que neuronas serán visibles y cuales no, es arbitraria. Además, del conjunto de neuronas visibles puede estar dividido en neuronas de entrada y de salida. Las *redes Hopfield* son un caso particular de red recurrente en el que no existen neuronas ocultas y la conectividad entre las neuronas visibles es total, o sea, toda neurona está conectada con todas las restantes (excepto en los casos de dilución). El input de la red es el estado inicial y el output es el estado estacionario. Un caso más general que las redes Hopfield lo constituyen las llamadas *Maquinas de Boltzmann* o *Boltzmann Machines* donde las conexiones son también simétricas ($w_{ij} = w_{ji}$), pero en este caso si podemos tener neuronas ocultas.

En este trabajo nos concentraremos en las redes feed–forward, ya que éstas pueden ser aplicadas de manera particularmente simple al problema de la estéreo

visión.

2.7. Aprendizaje

Como vimos en la Sección 2.4, con el caso de la función AND, hay situaciones en las que es posible, de manera analítica, encontrar una configuración adecuada de la red neuronal para un problema determinado. Pero estos casos son muy pocos. En general, cuando se tienen muchas neuronas interconectadas entre si, o cuando la tarea a realizar es muy compleja, encontrar una solución analítica para el problema se torna infactible. Por este motivo, surge la necesidad de encontrar una solución mediante algún método numérico.

La capacidad de aprender es una de la tantas condiciones que exigimos al definir a alguien o algo como *inteligente*, Si bien dar una definición precisa de *aprendizaje* es una tarea difícil (especialmente en sistemas biológicos), el proceso de aprendizaje en el contexto de redes neuronales artificiales puede entenderse como el problema de modificar la arquitectura de la red y el peso de sus conexiones, de forma tal que la red pueda desarrollar una tarea específica en forma eficiente. Ésta usualmente debe *aprender* el conjunto de conexiones sinápticas a partir de un dado conjunto de entrenamiento. Su capacidad aumenta en el tiempo mediante ajustes iterativos de sus conexiones. Esta abilidad de las redes neuronales artificiales de aprender a partir de ejemplos las hace particularmente interesantes desde el punto de vista biológico, y útiles desde el punto de vista computacional. En lugar de seguir un conjunto de especificaciones dictadas por un experto humano, estas redes son capaces de descubrir reglas y propiedades subyacentes, a partir de un dado conjunto representativo de ejemplos.

Para entender o para diseñar procesos de aprendizaje, uno debe primero tener un modelo de medio ambiente en el cual la red tendrá que operar. En otras palabras, debemos conocer que tipo de información puede usar nuestra red. Este mode-

lo usualmente se conoce como *paradigma de aprendizaje*. En segundo lugar, debemos entender como la red va a actualizar su arquitectura y sus conexiones sinápticas. Un algoritmo de aprendizaje es precisamente un procedimiento por el cual podemos ajustar repetidamente los pesos sinápticos.

Existen básicamente tres paradigmas de aprendizaje: supervisado, no supervisado e híbrido. En el *aprendizaje supervisado*, también conocido como *aprendizaje con un maestro*, la red cuenta con la respuesta correcta para un dado conjunto de entradas. Ella determina entonces los pesos sinápticos, a partir de sus propios errores durante un proceso de entrenamiento. El problema de *aprendizaje reforzado* es una variante de aprendizaje supervisado, en el cual la red no cuenta con la respuesta correcta, pero si con una crítica sobre su resultado. En el otro extremo, el *aprendizaje no supervisado* o *aprendizaje sin maestro* no requiere que la red conozca ninguna respuesta correcta. Ella explora la estructura subyacente de los datos o la correlación entre ellos, organizándolos en clases o categorías. En el *aprendizaje híbrido* se combinan técnicas de los dos métodos previos. Parte de los pesos se determinan usando aprendizaje supervisado y parte a partir de la estructura de los datos usados.

La investigación teórica en el proceso de aprendizaje trata de tres cuestiones fundamentales a la hora de calificar el desempeño de una dada red: capacidad, complejidad de muestras y complejidad computacional. Por *capacidad* nos referimos al número de patrones que podemos almacenar y que limitaciones funcionales y operativas tiene la red. La *complejidad de muestras* determina cuál es el número mínimo de ejemplos con los cuales debemos entrenar a la red (conjunto de entrenamiento), para que realice eficientemente la tarea deseada cuando le presentemos un caso no aprendido (en otras palabras, para que *generalice*). En general, si se entrena con pocos patrones se produce *overfitting* (la red funciona bien con el conjunto de entrenamiento pero no es capaz de generalizar la regla al tratar ejemplos nuevos no aprendidos previamente). La complejidad computacional se refiere al tiempo requerido para que la red encuentre o aprenda una solución adecuada para el conjunto de

entrenamiento. Muchos algoritmos existentes, si bien son teóricamente eficientes, en la práctica se tornan engorrosos pues cuando el número de neuronas se hace medianamente grande, el tiempo que requieren para aprender se torna una seria limitación para las computadoras actuales.

Existen básicamente cuatro reglas de aprendizaje: por corrección de error, reglas Hebbianas, reglas de Boltzmann y aprendizaje competitivo. En este trabajo sólo nos dedicaremos a la primera familia de problemas.

Vamos a describir brevemente ahora el problema conocido como *aprendizaje supervisado*. Recordemos que en el aprendizaje supervisado los resultados de la red pueden se cotejados con las respuestas correctas, recibiendo un feedback que depende de la magnitud del error cometido y que altera la matriz de sinapsis.

Asumiremos que la red dispone de un conjunto finito de pares input–output correctos, que llamaremos *conjunto de entrenamiento*, los cuales se usarán para entrenarla. Este conjunto es un subconjunto del conjunto de todos los posibles pares input–output, y desearíamos por supuesto que fuese lo menor posible. Cuando le presentamos a la red uno de los inputs del conjunto de entrenamiento, podemos comparar la respuesta que ésta produce con el output correcto. A continuación podemos modificar las sinapsis w_{ij} de forma tal que minimicemos esta diferencia. Ésto se hace normalmente en forma incremental, haciendo pequeños ajustes en respuesta al resultado de la red, ante cada par input–output del conjunto de entrenamiento, de modo tal que los w_{ij} converjan – si el método es adecuado – a una solución para la cual el conjunto de entrenamiento es resuelto con fidelidad (con poco error). Una vez hecho ésto, es interesante ver como funciona la red con un input que no forme parte del conjunto de entrenamiento, a fin de verificar si consiguió *generalizar* adecuadamente lo que le hemos enseñado. Como ya dijimos, en este trabajo nos concentraremos en redes con arquitecturas multicapas. En este caso es particularmente simple entender el proceso de aprendizaje.

2.8. El Perceptron Simple

En esta sección vamos a hablar sobre perceptrones de una única camada, los cuales se conocen usualmente como perceptrones simples. Existe entonces un conjunto de N neuronas de entrada y una camada de salida. Vamos a denotar el estado de las neuronas de entrada como ξ_k y el estado de las neuronas de salida como O_i. La computación que realiza la red será entonces:

$$O_i = g(h_i) = g\left(\sum_{k=1}^{N} w_{ik}\xi_k\right) \ , \tag{2.10}$$

donde $g(h)$ es la función de activación que computa cada neurona. $g(h)$ usualmente es una función no–lineal, pero también aquí podemos usar a priori cualquier tipo de función de activación. Más adelante veremos esto en más detalle.

Notemos que el output es una función explícita del input. Existe un verdadero feed–forward: el input se propaga hacia adelante a lo largo de la red hasta producir el output.

Hemos omitido por ahora todos los umbrales de activación en nuestro tratamiento, pues éstos pueden ser siempre reemplazados por una conexión extra a una neurona adicional en la capa de entrada que está fija en el estado -1. Más específicamente podemos fijar $\xi_0 = -1$ y elegir las conexiones $w_{i0} = \theta_i$, de forma tal que:

$$O_i = g(h_i) = g\left(\sum_{k=0}^{N} w_{ik}\xi_k\right) = g\left(\sum_{k=1}^{N} w_{ik}\xi_k - \theta_i\right) \ , \tag{2.11}$$

con θ_i haciendo el papel del umbral de activación de la neurona i-ésima de la camada de salida.

La tarea de asociación que deseamos que realice la red puede imaginarse como dar la respuesta correcta ζ_i^μ ante a una dada entrada ξ_k^μ. En otras palabras, queremos que cuando la entrada sea ξ_k^μ, la salida O_i sea igual a ζ_i^μ:

$$O_i^\mu = \zeta_i^\mu \qquad \text{(esto es lo deseado)} \tag{2.12}$$

para todo i y todo μ. Para el perceptron simple, como vimos, la salida está dada por

$$O_i^\mu = g(h_i) = g\left(\sum_k^N w_{ik}\xi_k^\mu\right) \ . \tag{2.13}$$

El conjunto de pares input–output $\{\xi_k^\mu, \zeta_i^\mu\}$ es el conjunto de entrenamiento, y vamos a suponer con consta de p pares de patrones ($\mu = 1, 2, \ldots, p$).

Tanto los inputs ξ_k, como los outputs O_i y los valores de salida del conjunto de entrenamiento ζ_i pueden ser binarios o continuos. En el caso de los outpus esto dependerá del tipo de función de activación que usemos. Algunas veces podemos tener outputs continuos en la red y discretos en el conjunto de entrenamiento.

Entre las posibles funciones de la red se puede incluir el problema de memoria asociativa. En este caso la camada de salida será del mismo tamaño que la camada de entrada. Este tipo de problema se conoce usualmente como *auto–asociación*, en contraste con problemas de *hetero–asociación* en los cuales estos dos conjuntos no coinciden. En general el tamaño de la camada de salida es menor o mayor que el tamaño de la camada de entrada, dependiendo del problema que estemos analizando. Algo importante de destacar es que para perceptrones simples, si existe un conjunto de acoplamientos sinápticos w_{ij} que son capaces de realizar una computación particular, entonces estos acoplamientos pueden encontrarse mediante un proceso de aprendizaje. La regla de aprendizaje comienza con una configuración inicial arbitraria para los valores de las conexiones y luego realiza sucesivas mejoras. Este método lleva a un conjunto adecuado de conexiones en un número finito de pasos.

Existen sin embargo problemas extremamente simples y conceptualmente muy importantes desde el punto de vista computacional que no pueden ser resueltos con un perceptron simple. Esto no contradice el párrafo anterior, pues el mismo nos decía que el método de aprendizaje converge siempre y cuando exista una solución (un conjunto de conexiones sinápticas) del problema y nada afirma sobre la existencia o no de la misma.

2.9. Descenso por el Gradiente en el Perceptron Simple

En esta sección describiremos un algoritmo de aprendizaje para el perceptron simple, que nos permitirá encontrar, en caso de que converja, un conjunto de conexiones sinápticas adecuado para solucionar un problema determinado. El método que describiremos a continuación, llamado descenso por el gradiente, comenzará de un conjunto de sinapsis arbitrario y realizará sucesivas modificaciones en éste, a fin de aproximarse cada vez más a una solución del problema. Para ello, definiremos una función que nos permitirá medir la magnitud del error como:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 \tag{2.14}$$

$$= \frac{1}{2}\sum_{i\mu}\left[\zeta_i^\mu - g\left(\sum_k^N w_{ik}\xi_k^\mu\right)\right]^2. \tag{2.15}$$

Ésta, que depende sólo de los w's y del conjunto de entrenamiento, siempre adopta valores positivos. Éstos son más pequeños cuanto más cerca se encuentra el conjunto de sinapsis de la solución del problema, siendo cero cuando hallamos la solución exacta. Por lo tanto, podemos pensar en el proceso de aprendizaje como un descenso por la superficie de la función de error, en el espacio $\mathbf{w}$. El método del descenso por el gradiente propone corregir cada w_{ik} por una cantidad Δw_{ik} proporcional al gradiente de E en la posición actual:

$$\Delta w_{ik} = -\eta\frac{\partial E}{\partial w_{ik}} \tag{2.16}$$

$$= \eta\sum_\mu\left[\zeta_i^\mu - g(h_i^\mu)\right]g'(h_i^\mu)\xi_k^\mu. \tag{2.17}$$

Ahora, y solo por claridad conceptual, definiremos los δ_i's como:

$$\delta_i = g'(h_i^\mu)[\zeta_i^\mu - O_i^\mu], \tag{2.18}$$

y reescribiremos la ecuación (2.17) de la siguiente manera:

$$\Delta w_{ij} = \eta \sum_{\mu} \delta_i \xi_k^{\mu}. \tag{2.19}$$

Hay que destacar que cuando la función de activación es una sigmoide (como la tanh), el factor $g'(h_i^{\mu})$ en las ecuaciones (2.17) y (2.18) se hace grande cuando el valor absoluto del campo local $|h_i^{\mu}|$ es pequeño. Ésto hace que los cambios en las sinapsis sean bruscos en las situaciones en donde el output de la neurona se torna *dudoso* ($|h_i^{\mu}| \approx 0$). Ésto permite salir rápidamente se estas situaciones de indeterminación.

Una factor importante que hace que la *tanh* sea una buena elección como función de activación, es que:

$$g'(h) = \beta(1 - g^2)$$

por lo tanto, es muy económico calcular $g'(h)$ en (2.18), una vez que se tiene $O_i^{\mu} = g(h_i^{\mu})$. Algo similar sucede con la función de Fermi:

$$g(h) = \frac{1}{1 + exp(-2\beta h)},$$

si usamos neuronas con valores entre 0 y 1, ya que su derivada:

$$g'(h) = 2\beta g(1 - g),$$

también se escribe en función de si misma.

La condición necesaria para que el método del descenso por el gradiente encuentre solución a un problema dado, es la independencia lineal de los patrones. Pero lamentablemente, ésta no es condición suficiente, ya que como el algoritmo siempre desciende sobre la superficie de la función de error, éste puede caer en un mínimo local (mayor a 0). La Figura 2.7, ilustra una función de error que posee un mínimo global, representado con un punto solido (•), y dos mínimos locales representados con círculos (○), donde el algoritmo podría quedar atrapado. En la Sección 2.11.2 explicaremos como solucionar, al menos en la mayoría de los casos, el problema de los mínimos locales para el algoritmo de backpropagation, que está basado en el descenso por el gradiente. Las mismas consideraciones pueden ser aplicadas aquí.

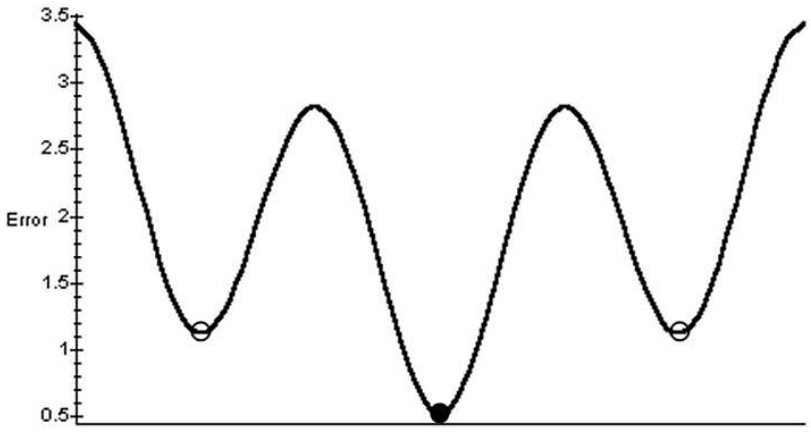


Figura 2.7: Función de error que posee un mínimo global y dos mínimos locales

2.10. Perceptrones Multicapa

Como ya lo mencionamos, el cerebro humano posee 10^{11} neuronas. Cada una de ellas se conecta en media con otras 10^4 neuronas. Por lo tanto, la conectividad es mucho más pequeña que el numero total de neuronas. Además, ésta no es aleatoria. Si así lo fuese, el cerebro sería mucho más grande de lo que es. Si observamos, la primera organización global es una estructura de capas de neuronas. Estas están ordenadas jerárquicamente y en orden de creciente de complejidad. Un ejemplo de este tipo de estructura es el sistema visual [Hubel and Wiesel, 1959, 1962, 1965]. Inspiradas por ésto y en su contrapartida formal, existen arquitecturas de lo mas variadas. Una de las más usadas y que mejor se adapta al problema abordado es la de las redes feed forward multicapa, llamados *Perceptrones*, donde las neuronas se distribuyen en capas (layers) y la información se dirige en un solo sentido. La Figura 2.8 ilustra una arquitectura típica de una red neuronal con 4 neuronas de entrada, 2 neuronas de salida y 1 capa de 3 neuronas ocultas.

La versatilidad de este tipo de redes, como aproximador de funciones continuas, está asegurada por el teorema de Kolmogorov. Éste dice que, dada una función

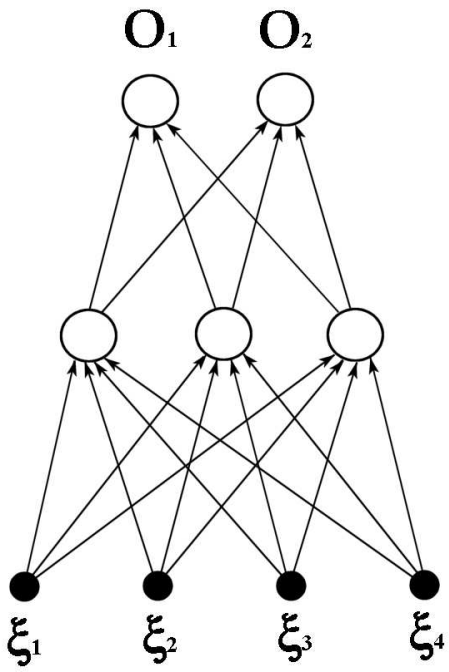


Figura 2.8: Perceptrón Multicapa

arbitraria $f : [0,1]^N \to \Re$ y $\epsilon > 0$, existe un entero M y conjuntos de reales α_i, θ_j, w_{ij}, $i = 1 \ldots M$, $j = 1 \ldots N$, tales que si definimos:

$$F(\xi_1, \ldots, \xi_N) = \sum_i \alpha_i g\left(\sum_j w_{ij}\xi_j - \theta_j\right) \tag{2.20}$$

para $\overrightarrow{\xi} \in [0,1]^N$, entonces:

$$|F(\overrightarrow{\xi}) - f(\overrightarrow{\xi})| < \epsilon \qquad \forall \overrightarrow{\xi} \in [0,1]^N \tag{2.21}$$

para $g(x)$ continua, acotada y monótona creciente. La demostración se puede ver en [Hecht-Nielsen, 1987] y la omitiremos aquí.

Si bien el teorema demuestra la existencia de una red neuronal, con *una sola* capa oculta, para aproximar cualquier función continua $[0,1]^N \to \Re$ con precisión arbitraria, encontrarla no es trivial y plantea problemas asociados al algoritmo de aprendizaje, como el ya discutido problema del mínimo local.

2.11. El Algoritmo de Backpropagation

2.11.1. Breve Descripción

El algoritmo de Backpropagation Error [Rumelhart. 1986] está basado en el descenso por el gradiente y intenta encontrar un mínimo en una cierta función de error.

A continuación y por simplicidad omitiremos los umbrales, igual que en el caso del perceptron simple, ya que estos se pueden considerar como una sinapsis adicional, conectada a una neurona que permanece en estado -1.

Supongamos ahora que tenemos una red neuronal con $M-1$ capas ocultas. $v_i^{\mu,M}$ es la salida y ζ_i^μ la salida deseada para la neurona de salida i para el patrón $\mu \in$ {Conjunto de entrenamiento}. Definimos ahora, y de manera similar al descenso por el gradiente, el error cuadrático como:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{\mu i}\left(\zeta_i^\mu - v_i^{\mu,M}\right)^2 \tag{2.22}$$

y el campo local h para la neurona de salida i

$$h_i^{\mu,M} = \sum_j w_{ij}^M v_j^{\mu,M-1}. \tag{2.23}$$

Por lo tanto:

$$v_i^{\mu,M} = g(h_i^{\mu,M}) \tag{2.24}$$

Podemos definir ahora el error en la i-ésima neurona de salida como:

$$\delta_i^{\mu,M} = g'(h_i^{\mu,M})[\zeta_i^\mu - v_i^{\mu,M}] \tag{2.25}$$

y si aplicamos la regla del descenso por el gradiente a (2.22), donde desplegamos $v_i^{\mu,M}$, las modificaciones en las sinapsis de las neuronas de la capa de salida nos quedan:

$$\Delta w_{ij}^M = -\eta\frac{\partial E}{\partial w_{ij}^M} = \eta\sum_\mu \delta_i^{\mu,M} v_j^{\mu,M-1} \tag{2.26}$$

donde η es la tasa de aprendizaje o *learning rate*.

Hasta ahora siempre hicimos referencia a las neuronas de la capa de salida. Pero si retropropagamos los errores a las capas ocultas, tomando como error para la capa m-ésima:

$$\delta_i^{\mu,m} = g'(h_i^{\mu,m}) \sum_j \delta_j^{\mu,m+1} w_{ji}^{m+1} \tag{2.27}$$

la ecuación (2.26) nos queda:

$$\Delta w_{ij}^m = \eta \sum_\mu \delta_i^{\mu,m} v_j^{\mu,m-1} \tag{2.28}$$

donde $v_j^{\mu,0}$ es un alias de ξ_j^μ.

Para modificar las sinapsis en tiempo $t+1$ debemos hacer:

$$w_{ij}^m(t+1) = w_{ij}^m(t) + \Delta w_{ij}^m(t+1) \tag{2.29}$$

Notemos que en el caso en que las neuronas son lineales, en la ecuación (2.25) el término $g'(h_i^\mu)$ es 1, ya que $g(x) = x$.

Para una descripción mas detallada sobre el algoritmo de backpropagation ver [[Hertz]. pag. 115] .

2.11.2. Algunas Consideraciones de Implementación

- Existen dos modalidades para la actualización de las sinapsis. Una de ellas, descrita en la ecuación (2.28), consiste en calcular las modificaciones Δw que producen todos los patrones del conjunto de entrenamiento y luego modificar las sinapsis. Esta modalidad es denominada *bach*. De esta forma la unidad de tiempo de la ecuación (2.29) es una *época*, es decir, el tiempo necesario para pasar todos los patrones del conjunto de entrenamiento. Esta modalidad requiere memoria adicional para acumular las modificaciones durante la época.

Otra posibilidad es calcular las modificaciones Δw que produce cada patrón y actualizar las sinapsis inmediatamente después. A esta modalidad se la llama *on-line* y en este caso en particular dió mejores resultados, aunque en general, ésto depende del problema. En este caso, la unidad de tiempo de la ecuación (2.29) es un patrón y la ecuación (2.28) quedaría:

$$\Delta w_{ij}^{m} = \eta \delta_i^{\mu,m} v_j^{\mu,m-1} \tag{2.30}$$

En esta modalidad es muy importante en cada época permutar aleatoriamente el orden en que se pasan los patrones, ya que ésto permite una exploración más amplia de la superficie de la función de error y evita cancelaciones entre las modificaciones que producen los patrones.

- Uno de los problemas de este algoritmo es la localidad, ya que siempre desciende (cuando el η es suficientemente chico) por la superficie de la función de error. Es posible que éste caiga en un *mínimo local*. Para evitar esto, se debe lanzar el algoritmo muchas veces, partiendo de sinapsis aleatorias distintas cada vez. Luego se debe analizar los resultados producidos para descartar aquellos en los que no convergió. Otra opción es la incorporación de ruido, o sea, realizar cambios aleatorios en las sinapsis a fin de poder escapar de los mínimos locales. Ésto permite que el algoritmo esporádicamente ascienda una magnitud aleatoria sobre la superficie de la función de error.

- Para asegurar que las sinapsis no estén saturadas desde el inicio del entrenamiento, éstas se deben inicializar con valores aleatorios que oscilen entre $\pm\frac{1}{\sqrt{Fi}}$, donde Fi es el fan-in o número de neuronas de entrada de la neurona en cuestión.

- Una posible modificación es la incorporación de *momento* o inercia, que contribuye a mejorar la eficiencia del algoritmo, en zonas donde la función de

error es bastante plana. El algoritmo como lo describimos antes es de dinámica disipativa. Si modificamos la ecuación (2.29) incorporándole un término de momento, la regla de actualización tiene la forma:

$$w_{ij}^m(t+1) = w_{ij}^m(t) + \Delta w_{ij}^m(t+1) + \alpha \Delta w_{ij}^m(t) \tag{2.31}$$

donde $\alpha \in [0,1]$. Si $\alpha \simeq 0$ tenemos dinámica disipativa y si $\alpha \simeq 1$ tenemos dinámica conservativa.

- No es trivial encontrar la tasa de aprendizaje η adecuada, ya que si ésta es demasiado grande el algoritmo comienza a oscilar pudiendo incrementar el error. Por otra parte, si η es demasiado pequeño el algoritmo tarda mucho tiempo en converger. Y lo que es más... una valor adecuado para el comienzo del entrenamiento suele no serlo para las etapas finales. Una solución para este problema es la incorporación de *parámetros adaptivos*. Un posible criterio de adaptación es: partir de un valor relativamente grande para η y si luego de una iteración, el error disminuye, no hacer nada. Pero si el error aumenta, hacer $\eta = \rho\eta$ para algún $\rho \in [0,1]$ y hacer $\alpha = 0$ en la próxima iteración.

2.11.3. Backpropagation Paso a Paso

A continuación daremos una descripción paso a paso del algoritmo en su modalidad *on-line*, que intenta aclarar la estructura general del mismo. Incluiremos momento y parámetros adaptivos.

1. Inicializar los w's con valores aleatorios entre $\pm\frac{1}{\sqrt{Fi}}$
2. $E_{old}(\mathbf{w}) = \infty$ y $\alpha = 0$ la primera iteración.
3. Permutar el orden de los patrones
4. Para cada patrón μ hacer:

a) Tomar $v_i^0 = \xi_i$

b) Si $E_{new}(\mathbf{w}) \leq E_{old}(\mathbf{w})$ hacer

$\alpha = \alpha_0$

si no hacer $\eta = \rho\eta$ y $\alpha = 0$

c) Propagar la señal desde $m = 1 \ldots M$:

$$v_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m v_j^{m-1}\right) \tag{2.32}$$

d) Calcular el error en la salida:

$$\delta_i^M = g'(h_i^M)[\zeta_i - v_i^M] \tag{2.33}$$

e) Retropropagar los errores desde $m = M - 1 \ldots 1$:

$$\delta_i^m = g'(h_i^m) \sum_j \delta_j^{m+1} w_{ji}^{m+1} \tag{2.34}$$

f) Calcular las modificaciones para $m = 1 \ldots M$:

$$\Delta w_{ij}^m = \eta \delta_i^m v_j^{m-1} + \alpha \Delta w_{ij}^m \tag{2.35}$$

g) Actualizar las sinapsis:

$$w_{ij}^m = w_{ij}^m + \Delta w_{ij}^m \tag{2.36}$$

5. Si no *aprendió* volver a 3.

2.12. Aprendizaje y Generalización

Sea Ts el conjunto de entrenamiento y $P = |Ts|$. Definamos ahora el error en el aprendizaje promedio como:

$$E_{ap} = \frac{1}{P} \sum_{\mu i} (\zeta_i^\mu - O_i^\mu)^2 \qquad \mu \in Ts \tag{2.37}$$

y vamos a definir el error de generalización, que es una magnitud que nos indica que tan bien responde la red ante patrones que no pertenecen al conjunto de entrenamiento. Pero como muchas veces el dominio de la función a aproximar es infinito, como este caso en particular, se toma un subconjunto del dominio $Gs \subseteq DOM(f)$, donde f es la función a aproximar tal que $Ts \cap Gs = \emptyset$ y sea $G = |Gs|$. El error de generalización será:

$$E_{gen} = \frac{1}{G} \sum_{\mu i} (\zeta_i^\mu - O_i^\mu)^2 \qquad \mu \in Gs \tag{2.38}$$

No es trivial encontrar la arquitectura óptima para un problema determinado. Más aún, no es fácil determinar un buen criterio que permita decidir cuando una topología es mejor que otra, aunque el error de generalización es un buen indicio. Si la cantidad de sinapsis es excesiva, se produce lo que se denomina *overfitting*, que es como interpolar pocos puntos con una función con muchos grados de libertad... el ajuste es muy bueno (E_{ap} chico), pero la interpolación es muy mala (E_{gen} grande). Por otra parte, si la cantidad de sinapsis es demasiado chica, el aprendizaje es muy malo. En nuestra analogía sería como intentar interpolar una parábola con una recta.

Un número demasiado grande de sinapsis no es la única causa de *overfitting*. Un ajuste excesivo sobre los puntos del conjunto de entrenamiento, debido a un número grande de iteraciones, produce un resultado similar.

Nuestro objetivo es que la red generalice lo mejor posible. Pero cuando decimos que no es fácil determinar un buen criterio que permita decidir cuando una topología es mejor que otra, nos referimos a que podemos tener: una aproximación mediocre a todos los puntos, o bien una que ajusta muy bien a la mayoría y muy mal a unos pocos y el error se mantenga constante. Claramente, según el problema, una puede resultar mejor que otra.

2.13. Redes Modulares

Además de la estructura de capas, si observamos el cerebro a un nivel más microscópico, se puede ver una pequeña estructura muy regular. Pequeños módulos, de no más de cientos de neuronas, conocidas como *minicolumnas*(Mountcastle, 1975). Además podemos encontrar múltiples flujos paralelos de información. Un ejemplo de ésto, una vez más, es el sistema visual, donde los diferentes aspectos de los estímulos visuales, como color, movimiento y ubicación, son procesados en paralelo por sistemas neuronales separados(Livingstone and Hubel, 1988; Hilz and Rentschler, 1989). La información visual, procesada por separado, luego es integrada por un sistema de mayor nivel, para formar una única percepción .(Poggio, Gamble and Little, 1988; DeYoe and Van Essen, 1988).[[Happel and Murre]]. La división anatómica del cerebro humano en dos hemisferios, por ejemplo, es una clara muestra de paralelización y especialización. Todos los grupos de funciones mentales están ubicados en diferentes mitades del cerebro. Pacientes con el cerebro dividido, donde la conectividad entre los hemisferios fue totalmente dañada, pueden tener vidas normales. Esto nos da una idea de que los dos hemisferios son muy independientes.

Una de las ventajas de la modularización es la minimización de la interferencia entre los procesos paralelos.

Una vez más, en la contrapartida formal, ésto constituye otro posible principio de construcción, pudiendo asignar a una pequeña red neuronal o módulo, una pequeña parte del problema y después reunir el resultado producido por cada módulo.

Capítulo 3

El Método Algorítmico

3.1. El Principio de Percepción de Profundidad

A continuación, daremos una descripción intuitiva que intenta explicar el principio de funcionamiento de la estéreo visión. Una descripción más detallada y precisa, de como recuperar la profundidad de cada pixel y generar el hipercubo que contiene la escena, se describe en la Sección 3.6. Supongamos que estamos observando con dos cámaras una superficie curva (ver Figura 3.1) en la que identificamos dos puntos particulares (en la figura, representados por $\bullet$ y $\star$). Se puede apreciar que las proyecciones sobre la imagen izquierda y derecha de $\star$, tienen coordenadas similares. No ocurre lo mismo con $\bullet$, ya que su proyección sobre la imagen derecha, se encuentra... valga la redundancia... más a la derecha. Y es más, ésta, estará más a la derecha cuanto más lejos esté $\bullet$ de las cámaras. Para ilustrar ésto, $\circ$ en la imagen izquierda corresponde a la proyección de $\bullet$ en la imagen derecha. Por el contrario, si $\bullet$ estuviera muy cerca de las cámaras, para la configuración de éstas descripta en la Figura 3.1, $\circ$ estaría a la izquierda de $\bullet$. Por lo tanto, midiendo la distancia entre las proyecciones de un mismo punto, podemos determinar a que distancia se encuentra éste de su observador.

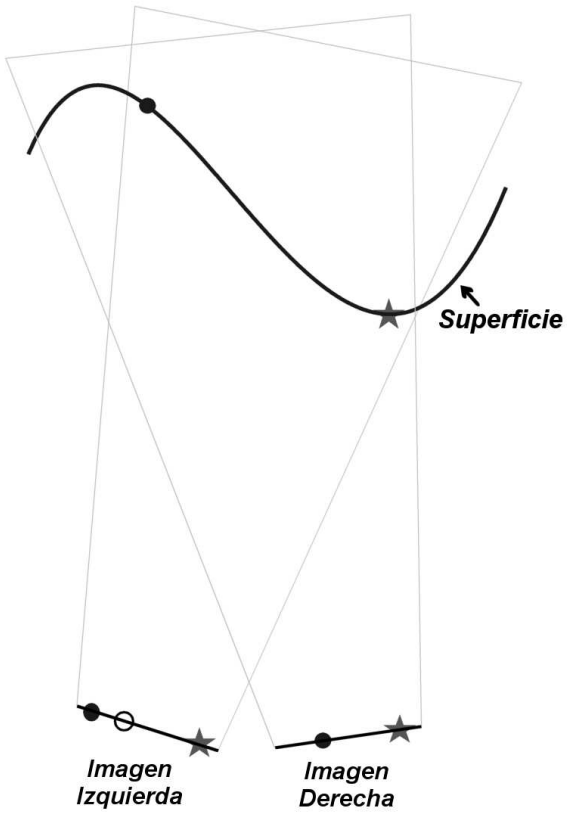


Figura 3.1: El concepto básico de estereovisión

3.2. Descripción del Algoritmo

Para entrenar la red, surgió la necesidad de implementar un algoritmo capaz realizar el cálculo de las profundidades que hiciera las veces de "supervisor" o "maestro". A continuación, describiremos uno de los métodos algorítmicos más tradicionales para realizar dicha tarea. Para el correcto funcionamiento de éste, es importante destacar, como lo hicimos en la sección 1, que asumimos que las imágenes están acotadas epipolarmente. Otra asunción importante es el hecho de que las disparidades o desplazamientos horizontales son pequeños y locales. Por ésto, dado un punto en la superficie original cuya proyección sobre la imagen izquierda tiene coor-

denadas (x, y), supondremos que su proyección sobre la imagen derecha se encuentra en el entorno $(x - \delta_l \dots x + \delta_h, y)$ del punto (x, y).

El método implementado es de los denominados "correlation based". Consiste en elegir una pequeña ventana (i.e.7x7 pixeles) o subimagen fija, digamos en la imagen izquierda, y desplazar otra horizontalmente en la imagen derecha. Más precisamente (ver Figura 3.2), si fijamos en la imagen izquierda aquella ventana centrada en el pixel de coordenadas (x,y), consideraremos en la imagen derecha aquellas ventanas centradas en los pixeles $(x - \delta_l \dots x + \delta_h, y)$. A medida que vamos realizando este

Figura 3.2: Desplazamiento de la ventana de correlación

desplazamiento, vamos computando algún *criterio de correlación* entre ambas ventanas (que analizaremos luego). Esto nos permitirá decidir que ventana, en la imagen derecha, se *ajusta* mejor a la fijada en la imagen izquierda. El objetivo de ésto es encontrar las proyecciones, sobre los planos de ambas imágenes, de un mismo punto en la superficie original. A continuación, cuando nos refiramos a la ventana (x,y), nos estaremos refiriendo a la ventana centrada en el pixel que posee éstas coordenadas.

El tamaño de la ventana impacta directamente en la precisión del ajuste, ya que si ésta es muy pequeña, es más probable que ajuste bien con otra ventana que no es la que le corresponde. Por el contrario, si ésta es muy grande, se pierden detalles en la superficie o pequeñas rugosidades, además de ser mucho más costoso en tiempo

de cálculo.

Representaremos al cubo que contiene la superficie como una matriz de dos dimensiones HM. Aquí, el valor HM_{xy} corresponde a la distancia del pixel de coordenadas x e y al observador. Llamaremos a esta matriz el *mapa de profundidades*.

3.3. El Criterio de Correlación

Existen varios criterios de correlación. Como no es el objetivo de este trabajo la investigación de los distintos métodos algorítmicos, sólo se evaluaron dos criterios que se describen a continuación:

Sean $I(x, y)$ y $D(x, y)$, las intensidades de los pixeles de coordenadas (x, y) en la imagen izquierda y derecha respectivamente, $d = x - \delta_l \ldots x + \delta_h$ el desplazamiento en el segmento de búsqueda, $2w$ el ancho y $2h$ el alto de la ventana de correlación. Ambos criterios son:

$$C_1(x, y, d, I, D) = \frac{\sum_{i,j}(I(x+i, y+j) - D(d+i, y+j))^2}{\sqrt{\sum_{i,j} I(x+i, y+j)^2}\sqrt{\sum_{i,j} D(d+i, y+j)^2}} \tag{3.1}$$

$$C_2(x, y, d, I, D) = \frac{\sum_{i,j} I(x+i, y+j)D(d+i, y+j)}{\sqrt{\sum_{i,j} I(x+i, y+j)^2}\sqrt{\sum_{i,j} D(d+i, y+j)^2}} \tag{3.2}$$

para $i = -w \ldots w$, $j = -h \ldots h$.

Ahora diremos que la ventana que está más correlacionada en segmento de búsqueda de la imagen derecha, con la fijada en la imagen izquierda (x, y), es aquella centrada en (d, y), para d que minimiza C_1 ó maximiza C_2. Ésto es así, porque C_1 usa las diferencias entre los niveles de grises de los pixeles, y por lo tanto, dos ventanas serán más parecidas cuanto menores sean estas diferencias. Por otra parte, C_2 multiplica los niveles de grises de ambas ventanas, por ello, cuanto más éstas se parezcan, mayor será el valor de C_2 obtenido.

Más adelante, en la Sección 3.4, compararemos el desempeño de ambos criterios, ya que allí veremos una buena forma de evaluarlos. Para una descripción más profunda acerca de distintos criterios de correlación y su desempeño, ver [Faugeras & Hotz].

Detalle de implementación:

Notemos que sólo estamos interesados en encontrar aquel $d = x - \delta_l \dots x + \delta_h$ que minimice o maximice $C(x, y, d, I, D)$, según el criterio adoptado. Para tal fin, una vez fijos x e y, el término $\sqrt{\sum_{i,j} I(x+i, y+j)^2}$ en el denominador permanece constante. Por ello, éste puede ser omitido ahorrando importante tiempo de cálculo.

3.4. Validando los Ajustes (Matches)

Es muy importante la validación de las correspondencias entre los puntos. Claramente, si las imágenes no son muy texturadas o si poseen patrones repetitivos, el algoritmo falla. Además, existen zonas de *oclusión*, que son puntos visibles desde una cámara, que no lo son desde la otra. Estos degradan mucho los resultados, produciendo picos en las superficies.

(a) Imagen de la cámara izquierda

(b) Mapa de alturas generado

Figura 3.3: Fotografía aérea del pentágono

Para ilustrar ésto, veamos la Figura 3.3(a), que corresponde a la imagen izquierda

de un par de fotografías aéreas del pentágono. En la Figura 3.3(b), se puede apreciar el resultado de aplicar el método antes descripto a este par de imágenes, sin ningún tipo de post–procesamiento. Esta imagen, corresponde al mapa de alturas HM, representado en escala de grises. En ella, los puntos blancos corresponden a aquellos muy cercanos a las cámaras, y los negros a aquellos muy lejanos. Podemos ver puntos blancos, aislados, correspondientes a grandes alturas, en zonas totalmente planas. Estos son producto de los malos ajustes o fallas a las que nos referimos. Es muy importante detectar estas fallas y descartar estos puntos antes de generar el ambiente 3D. Los puntos descartados posteriormente serán interpolados. Para la detección de estos puntos se probaron muchos métodos, como por ejemplo: agregar umbrales en la correlación del mejor ajuste, umbrales en la diferencia de correlación entre el mejor ajuste y el anterior, etc.. Pero la elección de estos umbrales no es trivial y generalmente dependen de las imágenes en cuestión. Además, en el caso que existan patrones repetitivos, las correlaciones con éstos son muy altas.

Un buen criterio de validación, es ver que si b, en la imagen derecha, es el punto que mejor ajusta a a, en la imagen izquierda, a sea el punto que mejor ajusta a b.

Más formalmente, si pensamos en el criterio C_2 y sea:

$$x_D = \max_{d=x-\delta_l \ldots x+\delta_h} C_2(x, y, d, I, D) \tag{3.3}$$

la validación consiste en verificar que se cumpla:

$$x = \max_{d=x_D-\delta_h \ldots x_D+\delta_l} C_2(x_D, y, d, D, I) \tag{3.4}$$

La probabilidad de que ésto ocurra sin que se trate del ajuste correcto, es bastante baja (suele pasar sólo un par da veces de cada 10,000 ajustes). La densidad del mapa de disparidad es un buen indicador de la calidad del criterio de correlación. Como era de esperar, el criterio C_2 produjo mejores resultados que C_1. En una muestra de 12 pares de imágenes, C_2 arrojó un 6 % más puntos validados que C_1.

En la Figura 3.4 se puede ver el mapa de alturas del pentágono, donde representamos en color blanco aquellos puntos que no pasaron la validación descrita. Notar,

Figura 3.4: Puntos que no pasaron la validación

que en las zonas de oclusión, junto a las aristas, la densidad de éstos es muy grande.

En la siguiente sección veremos como interpolar estos puntos y como depurar aún más las fallas del algoritmo.

3.5. El Postprocesamiento

Una vez computado el mapa de alturas y validados los ajustes, procederemos al post–procesamiento del mapa de alturas.

La validación descrita en la sección anterior descarta muy bien aquellos puntos que se encuentran en una zona de oclusión. Pero como ya lo mencionamos, otro factor que introduce error es la ocurrencia de patrones repetitivos. La Figura 3.5 muestra el mapa de alturas, donde interpolamos los puntos descartados (más adelante veremos de que forma). Allí podemos ver que aún quedan algunos malos ajustes por eliminar. Éstos se manifiestan en el mapa de alturas como un gran pico, aislado en la superficie. Por ello, podemos pensar en el mapa de alturas como una imagen con ruido. Éste ruido es similar al denominado “Sal y Pimienta” o “Salt & Pepper”, donde un pixel varía abruptamente su intensidad con respecto a la de sus vecinos. Para eliminarlo se aplico un “filtro de media” o “median filter”. Éste, consiste en calcular la altura media de los vecinos de cada pixel, considerando en ésta, sólo aquellos puntos que

Figura 3.5: Mapa de alturas luego de interpolar los puntos descartados

(a) Imagen con ruido "sal y pimienta"

(b) Resultado de la aplicación del "filtro de media"

Figura 3.6: Filtro de Media

pasaron la validación. Luego, si la distancia de éste con dicha media es mayor que un cierto valor dado Δ_{max}, a este pixel se le asigna el valor de la mencionada media. En la Figura 3.6(a), se puede apreciar una imagen a la que se le incorporó ruido "Sal y Pimienta". La Figura 3.6(b) muestra el resultado de la posterior aplicación del filtro descripto.

Ahora, sólo nos queda asignar algún valor a los puntos descartados. Lo que haremos es tomar como profundidad de éstos, el promedio de las profundidades de los puntos vecinos, repitiendo este procedimiento hasta la eliminación total de los mismos.

También se probó interpolación lineal, pero dado que generalmente los puntos

Figura 3.7: Mapa de alturas luego de la aplicación del filtro de media e interpolación.

de oclusión se generan junto a una arista o un cambio brusco en la profundidad, la interpolación lineal deforma mucho los resultados, transformando escalones en rampas.

Figura 3.8: Snapshoot's del ambiente 3D.

En la Figura 3.7, se puede apreciar el mapa de alturas del pentágono, luego de la aplicación del filtro de media y de la interpolación los puntos descartados, como la describimos antes. La Figura 3.8, muestra un snapshot del ambiente 3D, generado a partir de este mapa, mediante OpenGL. En él, se aplicó como textura, la imagen de la cámara izquierda.

3.6. Generando la Escena

Luego del postprocesamiento, procedemos a la reconstrucción del hipercubo que será nuestra representación de la escena original.

Supongamos que tomamos un punto en la imagen izquierda, digamos $p_l(x, y)$, y sea $C_j(x, y, d, I, D)$ el mejor ajuste según el criterio adoptado j, para $d = x-\delta_l \ldots x+\delta_h$. Podemos pensar que $p_l(x, y)$ y $p_r(d, y)$, son las proyecciones de un mismo punto $p(x_h, y_h, z_h)$ en el hipercubo original, sobre los planos de las imágenes izquierda y derecha respectivamente.

Figura 3.9: Configuración Paralela de las Cámaras

Ahora, con esta información, y si conociéramos las configuraciones internas y externas de las cámaras, estaríamos en condiciones de reconstruir las coordenadas de p.

Éstas, serán obtenidas por triangulación de la siguiente manera [WANG AND HSIAO]:

$$x_h = \frac{\beta(x+d)}{2(x-d)}$$

$$y_h = \frac{\beta y}{x-d}$$

$$z_h = \frac{\beta f}{x-d}$$

donde β es la distancia entre las cámaras y f la distancia focal (Ver Figura 3.9). Estas ecuaciones valen para una configuración paralela en las cámaras. Para una configuración arbitraria en el ángulo de éstas, siempre es posible rectificar las imágenes [Papadimitriou and Dennis]. La rectificación, consiste en proyectar ambas imágenes sobre el plano paralelo a la línea base o *baseline*. Ésta, como se puede apreciar en la Figura 3.9, es aquella que pasa por ambas cámaras.

Capítulo 4

Estéreo visión con Redes Neuronales

4.1. La Estructura de la Red

Como hemos visto,en el cálculo algorítmico el mayor esfuerzo está centrado en determinar, dada una ventana fija en una de las imágenes, cual es su ventana más correlacionada en la otra.

A continuación, centraremos nuestros esfuerzos en encontrar una pequeña red neuronal, o módulo, que realice esta tarea. Para ello, nuestra red tendrá como entrada la ventana fija de una de las imágenes, y el segmento de búsqueda de la otra. La salida comprende una única neurona entera, que luego del entrenamiento, esperaremos que corresponda al índice en el segmento de búsqueda, de la ventana que mejor ajuste, según el criterio de correlación adoptado. Ésto nos provee de la información necesaria para el cálculo de un punto en el mapa de profundidades. Por claridad, pensaremos que tenemos muchos módulos que comparten sus sinapsis. Llamaremos M_{xy}, a aquel que esperamos que calcule el mejor ajuste para la ventana centrada en el pixel de coordenadas (x,y). Dado que el criterio de correlación C_2 produjo mejores resultados, de ahora en más sólo pensaremos en éste. Por lo tanto, la función a aproximar por

M_{xy} será:

$$F(I,D) = \underset{d=-\delta_l \ldots \delta_h}{\text{máx}} \frac{\sum_{i,j} I(x+i,y+j)D(x+d+i,y+j)}{\sqrt{\sum_{i,j} I(x+i,y+j)^2}\sqrt{\sum_{i,j} D(x+d+i,y+j)^2}} \tag{4.1}$$

con $i = -w \ldots w$, $j = -h \ldots h$.

Para ello, la entrada de la red es de $4wh + 2h(\delta_l + \delta_h)$ neuronas, y comprende las intensidades $I(x+i, y+j)$ y $D(x-\delta_l \ldots x+\delta_h, y+j)$ (ver Figura 4.1). Éstos son los pixeles que el método algorítmico utiliza para calcular la profundidad de este punto. A la neurona de salida de la red la llamaremos O_{xy}. Luego, sólo hay que tener tantos

Figura 4.1: La Entrada del Módulo M_{xy} sombreada en color gris

módulos como puntos en el mapa de profundidades, o bien, desplazar un conjunto más pequeño de estos (posiblemente uno) para obtener secuencialmente el mismo efecto. Como se puede ver en la Figura 4.2, estos módulos poseen dos capas ocultas, hl_1 y hl_2 de $\mathbf{n_1}$ y $\mathbf{n_2}$ neuronas respectivamente. Estas neuronas son todas *no lineales* y adoptaremos como función de activación la tanh. O_{xy} es la única neurona *lineal* de la red. Esto es así para que ésta pueda alcanzar cualquier valor en su salida con igual distribución.

A continuación, fijaremos algunos parámetros sobre el tipo de imágenes a utilizar, para poder comenzar a hablar de una estructura más concreta para nuestra red.

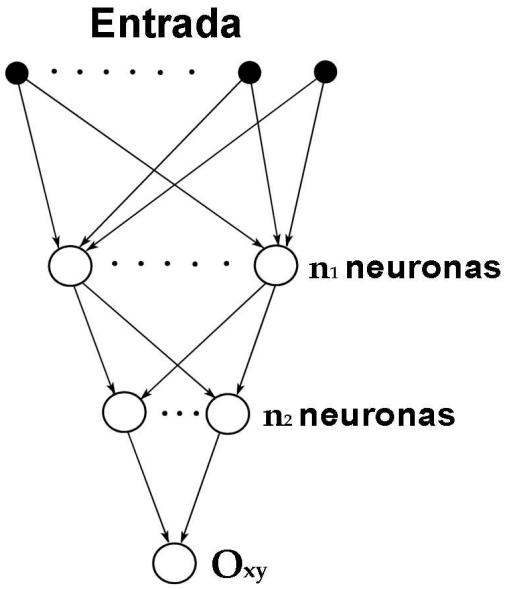


Figura 4.2: La Estructura de la Red

4.2. Las Imágenes Utilizadas

Trabajaremos con imágenes de una resolución de 128×128 pixeles en escala de grises. Este tipo de imágenes pueden ser vistas como una matriz de 128×128 enteros, cuyos valores oscilan entre 0 y 255. Estos valores representan las intensidades de cada pixel, siendo 0 negro y 255 blanco.

Ya que adoptamos como función de activación la tanh, es muy importante destacar la necesidad de preprocesar las imágenes antes de presentarlas a la red neuronal. De lo contrario esta función saturaría rápidamente. El preprocesamiento consiste en mapear los 256 colores o enteros posibles de cada pixel, a valores pseudo–reales en el intervalo $[-1, 1)$. Ésto se logra, suponiendo que $I_I(x, y)$ y $I_D(x, y)$ son los valores enteros para la intensidad del pixel (x,y) en la imagen izquierda y derecha respectivamente, haciendo:

$$I(x, y) = \frac{2I_I(x, y)}{256} - 1 \tag{4.2}$$

$$D(x,y) = \frac{2I_D(x,y)}{256} - 1. \tag{4.3}$$

Por el contrario, no realizaremos ningún tipo de preprocesamiento a los patrones de salida, ya que la red posee una neurona de salida lineal a fin de poder adoptar cualquier valor entero en ésta.

Ahora procederemos a fijar algunos parámetros como el alto y ancho de la ventana de correlación (h, w) y la máxima disparidad: $\delta_l + \delta_h$. Con el fin de minimizar el tamaño de la red, trataremos de que estos valores sean los más pequeños posibles, siempre y cuando ésto no degrade demasiado los resultados del procesamiento algorítmico.

Como lo mencionamos en el Capítulo 1, la disparidad máxima es una aproximación, ya que ésta depende de la configuración de las cámaras y de otros factores como la distancia a los objetos. Es de esperar que las cámaras estén calibradas según la distancia media a la superficie que deseamos reconstruir. En este trabajo utilizamos imágenes que poseen distintas distancias focales y diferentes configuraciones en el ángulo de las cámaras. Esto puede ser un factor que degrade notablemente los resultados pero se ajusta mejor a situaciones reales.

Para fijar el tamaño de la ventana de correlación, se evaluaron los resultados del método algorítmico con sólo la verificación visual. Para ello fue de gran utilidad el software desarrollado para este trabajo, que genera mediante OpenGL la reconstrucción en tres dimensiones de la superficie generada. Todas las escenas 3D mostradas en este trabajo son generadas mediante éste programa. Únicamente se probaron ventanas cuadradas, partiendo de un tamaño relativamente grande, y decrementándolo mientras los resultados fueran visualmente aceptables. El mínimo tamaño "aceptable", fue con una ventana de correlación de 6×6 pixeles.

El valor para la disparidad máxima, $\delta_l + \delta_h$, fue más fácil de encontrar. Se tomó un conjunto de imágenes y se corrió el método algorítmico con un valor grande para $\delta_l + \delta_h$. Finalmente, el valor adoptado fue la máxima disparidad registrada. En este caso, el máximo desplazamiento fue de 14 pixeles.

Es probable que si las imágenes fueran tomadas con la misma configuración en las cámaras y fuesen de objetos a distancias similares, este intervalo fuese menor. Al ser más chico éste, también lo sería la entrada de la red. Esto disminuiría notablemente el ruido producido por pixeles que no son relevantes al punto que estamos interesados en calcular. Hay que pensar que en el conjunto de imágenes, tenemos desde fotos aéreas hasta algunas tomadas a unos pocos centímetros. Además, y ya que la salida de la red neuronal corresponde a un índice en este intervalo, al ser éste más pequeño, estamos disminuyendo también el rango de valores en la salida de la red. Ésto produciría una notable mejora en el desempeño de la misma.

4.3. Buscando la Mejor Topología

En general, encontrar la mejor topología de una red neuronal para un problema determinado es un proceso empírico. Sólo consideraremos redes con dos capas ocultas, ya que se pudo comprobar rápidamente que aquellas con una sola capa oculta no aprendían. Por otra parte, aquellas que poseen más de dos eran demasiado lentas y no mejoraban significativamente la generalización. En nuestro caso, testearemos de forma incremental un conjunto de configuraciones para los tamaños de las capas ocultas n_1 y n_2.

Para realizar estas pruebas se seleccionaron 13 pares de imágenes, de los cuales uno fue reservado para el testeo de la generalización y no intervino para nada en el entrenamiento. A cada uno de estos pares de imágenes se le aplicó el método algorítmico. Posteriormente se validaron los datos como lo describimos en la Sección 3.4, para descartar aquellos puntos que no son confiables. De esta forma, los 12 pares de imágenes produjeron 114.405 patrones de entrada-salida. A este conjunto de patrones lo llamaremos Bs. Por otra parte, 10.447 patrones fueron producidos por el par de imágenes de testeo. A este conjunto de patrones de entrada-salida lo llamaremos Gs.

Entrenar la red neuronal con todos los patrones de una imagen no produce ventajas significativas. Ésto se debe a que ellos están muy correlacionados entre si, ya que un patrón de entrada con su adyacente, sólo está desplazado un pixel. Por ello, sólo elegiremos aleatoriamente un subconjunto de Bs como conjunto de entrenamiento. Llamaremos a este subconjunto Ts. El resto de los puntos de Bs, que llamaremos Rs, serán también utilizados para evaluar la generalización:

$$Rs = Bs - Ts \ , \tag{4.4}$$

y por lo tanto $Rs \cap Ts = \emptyset$.

Ahora describiremos los errores que valoraremos en el estudio del desempeño de la red neuronal. Definimos el error porcentual de aprendizaje absoluto promedio como:

$$E_{Ts} = \frac{100}{(\delta_l + \delta_h)|Ts|} \sum_{\mu \in Ts} |\zeta^\mu - O^\mu| \ . \tag{4.5}$$

También definiremos el error de generalización tanto en Rs como Gs de forma similar:

$$E_{Rs} = \frac{100}{(\delta_l + \delta_h)|Rs|} \sum_{\mu \in Rs} |\zeta^\mu - O^\mu| \ , \tag{4.6}$$

$$E_{Gs} = \frac{100}{(\delta_l + \delta_h)|Gs|} \sum_{\mu \in Gs} |\zeta^\mu - O^\mu| \ . \tag{4.7}$$

El término $(\delta_l + \delta_h)$ en el denominador, se utiliza a fin de normalizar los errores y obtener una magnitud más descriptiva. Este intervalo define el rango de valores que puede adoptar la salida de la red neuronal.

Los valores que propusimos en la sección 4.2 para el tamaño de la ventana de correlación y el intervalo de búsqueda, nos dejan una entrada para nuestra red neuronal de 156 neuronas ($156 = 4wh + 2h(\delta_l + \delta_h)$).

Ahora, y a fin de encontrar la cantidad óptima de neuronas para las capas ocultas, n_1 y n_2, simularemos un gran número de configuraciones y evaluaremos los

errores producidos. Para evaluar cada configuración, se entrenó la red varias veces (mediante backpropagation), con el objeto de obtener la media de los errores antes mencionados. Para cada simulación, se eligió aleatoriamente un nuevo conjunto de entrenamiento y de sinapsis iniciales. *A continuación, cuando hablemos de error*, ya sea en el aprendizaje como en la generalización, *nos estaremos refiriendo a* ***estas medias***.

En la implementación de algoritmo de backpropagation se incluyeron *todas* la consideraciones descriptas en la Sección 2.11.2.

El criterio de detención o parada del algoritmo de entrenamiento fue que la tasa de aprendizaje η, o el error de aprendizaje E_{Ts}, fuesen suficientemente pequeños. Recordemos que con parámetros adaptivos, η disminuye cada vez que se produce un comportamiento oscilatorio en el error de aprendizaje. Este criterio (η chico), funciona de manera similar a tener un numero fijo de iteraciones, aunque no exactamente igual, ya que si se alcanza un mínimo rápidamente, luego se producirán situaciones oscilatorias, decrementando rápidamente η y produciendo una finalización temprana del entrenamiento.

Para encontrar las cantidades óptimas de neuronas en las capas ocultas hubiera sido conveniente variar *simultáneamente* las cantidades de neuronas en las capas ocultas y el tamaño del conjunto de entrenamiento. Pero, al tener tantos parámetros libres, ésto se vuelve computacionalmente inviable (al menos para los tiempos disponibles para este trabajo). Recordemos que cada simulación se repite varias veces, a fin de obtener la media de los errores. Cada una de éstas repeticiones puede llegar a requerir 10 horas de cómputos o más.

Lo que se hizo fue fijar el tamaño del conjunto de entrenamiento en 30000 patrones. Luego, se varió las cantidades de neuronas $n_1 = 12 \ldots 26$ y $n_2 = 6 \ldots 24$ con pasos de 2 neuronas. Más tarde, y una vez determinada la mejor arquitectura, fijaremos ésta y evaluaremos su desempeño a medida que decrementamos o incrementamos el tamaño del conjunto de entrenamiento.

En la Figura 4.3, se puede apreciar el comportamiento del error de aprendizaje E_{Ts}, en función de las cantidades de neuronas en las capas ocultas, donde cada curva representa un valor distinto para n_1, y sobre la abscisa se encuentran los distintos valores de n_2. Allí se puede ver que el ajuste mejora a medida que aumentamos el

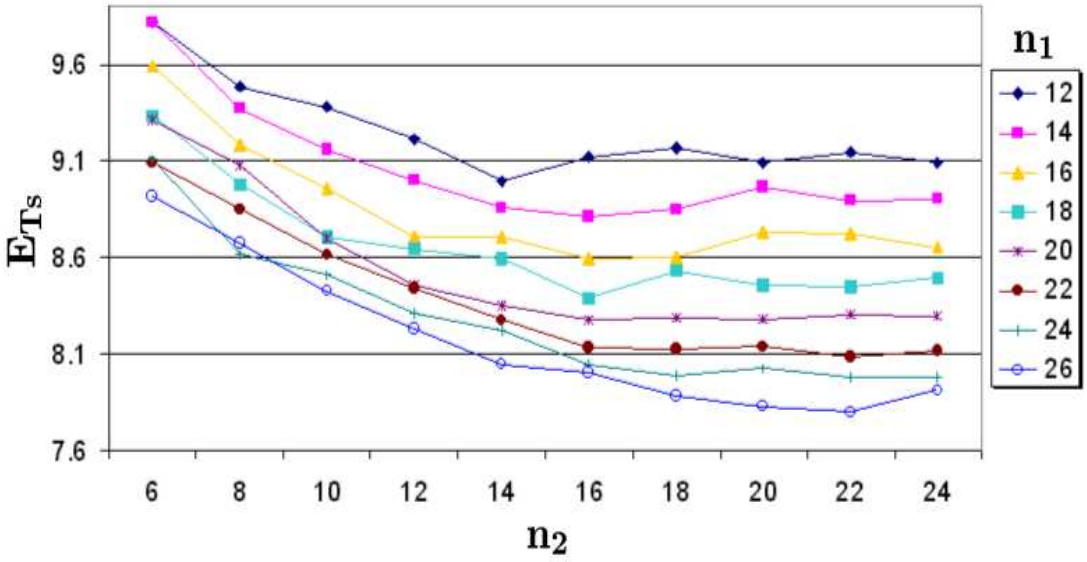


Figura 4.3: Error de aprendizaje E_{Ts} en función de n_1 y n_2

número de neuronas. Por otra parte, en la gráfica se puede apreciar que no produce mejoras el incrementar n_2 muy por encima de n_1. Es más, después veremos que también empeora la generalización. Es por ello que para valores grandes de n_1, *no se testearon* valores de n_2 aún más grandes.

Recordemos que estamos interesados en encontrar aquella arquitectura que mejor generalice. Por lo tanto, el error de aprendizaje (E_{Ts}), sólo nos sirve para determinar que tan bien la red neuronal está aprendiendo, y para obtener una cota inferior para el error en la generalización.

Veamos entonces la Figura 4.4, en donde de manera similar a la gráfica anterior, se puede apreciar el comportamiento del error, en función de las variaciones de n_1 y n_2. Esta vez, el error corresponde a la generalización en los puntos de Bs no utilizados

en el entrenamiento (E_{Rs}). Según esta gráfica, parecería que la red generaliza mejor

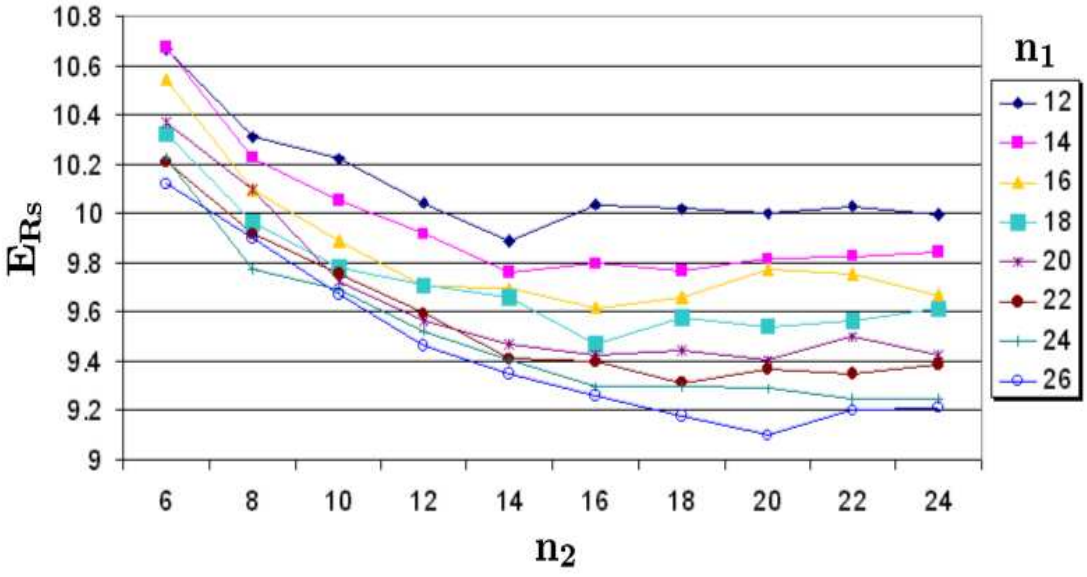


Figura 4.4: Error de generalización E_{Rs} en función de n_1 y n_2

cuanto mayor sea el número de neuronas en sus capas ocultas. Ésto se contradice con la teoría, ya que debería existir un valor óptimo para el número de sinapsis, a partir del cual, aumentar el número de éstas debería empeorar la generalización. Además, tenemos indicios, que veremos más adelante, de que éste valor óptimo está dentro del rango de las configuraciones evaluadas. Por ello, atribuiremos éste comportamiento al hecho de que los patrones en Rs están muy correlacionados con los del conjunto de entrenamiento Ts. Por lo tanto, la mejora en la supuesta generalización es sólo una consecuencia o efecto secundario de tener un mejor ajuste sobre los puntos del conjunto de entrenamiento.

Un indicador mucho más confiable del error en la generalización es aquel tomado en base a una imagen que no intervino para nada en el entrenamiento. Ésto se debe a que los patrones de entrada-salida que ésta provee, están mucho menos correlacionados con los de Ts. La Figura 4.5, ilustra el comportamiento del error de generalización en el conjunto de testeo Gs, donde igual que antes, para un conjunto

de entrenamiento de 30.000 patrones, variamos n_1 y n_2 en los rangos ya descriptos. Allí podemos ver como lo habíamos mencionado antes, que incrementar n_2 por enci-

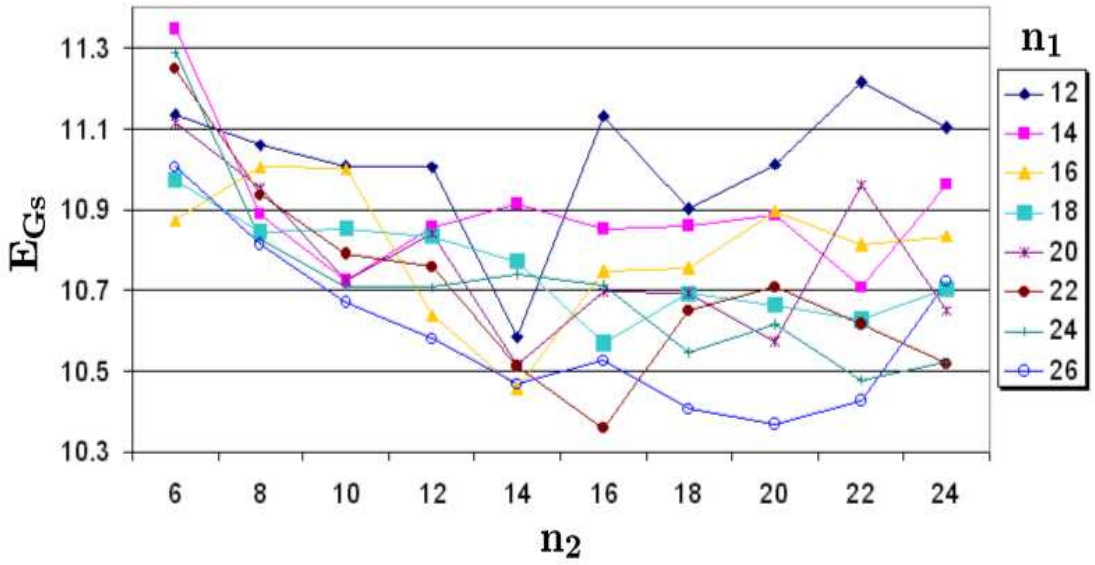


Figura 4.5: Error de generalización E_{Gs} en función de n_1 y n_2

ma de n_1 *degrada* la generalización. Además, se puede apreciar que E_{Gs} disminuye a medida que incrementamos n_1, mientras $n_1 < 22$. Cuando $n_1 = 22$ y $n_2 = 16$ tenemos el E_{Gs} mínimo de todas las configuraciones evaluadas. Esta configuración es la que adoptaremos como óptima, ya que la gráfica indica que aumentar el número de neuronas por encima del de ésta no mejoraría la generalización, además de tener el inconveniente de que la red neuronal se vuelve más lenta y costosa si pensamos en su implementación en hardware y en software. Por lo tanto *diremos que la red neuronal que produjo el mejor desempeño, fue la* 156-22-16-1, o sea: 156 neuronas de entrada, 22 neuronas en la primer capa oculta, 16 en la segunda y una única neurona de salida.

Ahora veremos como se comporta ésta a medida que variamos el tamaño del conjunto de entrenamiento. En la Figura 4.6 podemos ver la curva del error en el aprendizaje E_{Ts} para la red 156-22-16-1 a medida que incrementamos exponen-

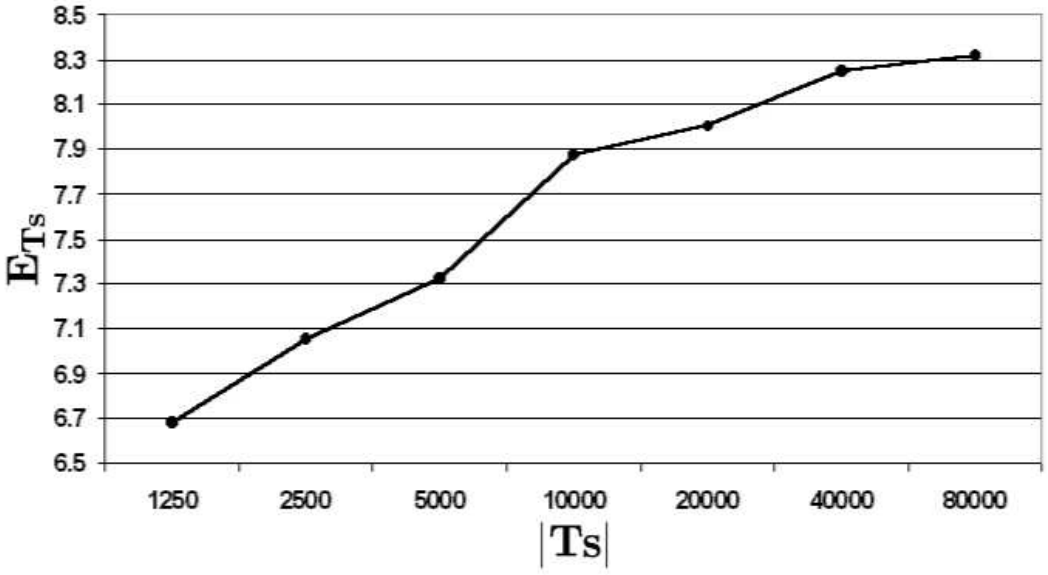


Figura 4.6: Error de aprendizaje E_{Ts} para la red 156-22-16-1, en función de la cantidad de patrones de entrenamiento

cialmente el número de patrones del conjunto de entrenamiento. Los valores en la ordenada corresponden a E_{Ts} y los valores en la abscisa corresponden a la cantidad de patrones de entrada-salida del conjunto de entrenamiento Ts. Allí, podemos ver como el ajuste empeora a medida que intentamos aprender más patrones. Por el contrario, se puede ver en las Figuras 4.7 y 4.8, que la generalización mejora notablemente. En éstas, se puede apreciar el impacto que tiene la cantidad de patrones que intervienen en el entrenamiento, sobre los errores de generalización E_{Rs} y E_{Gs} respectivamente. Una vez más, E_{Gs} es un parámetro más confiable de la capacidad de generalización que E_{Rs}. Según los resultados de las simulaciones, éste último siempre es menor que el primero, a pesar de tratarse de patrones elegidos al azar. Tomaremos ésto como un indicio de la alta correlación de los patrones de Rs con el Ts.

Si observamos la curva de error de la figura 4.8, parecería que si entrenáramos la red neuronal con más patrones, la generalización mejoraría significativamente.

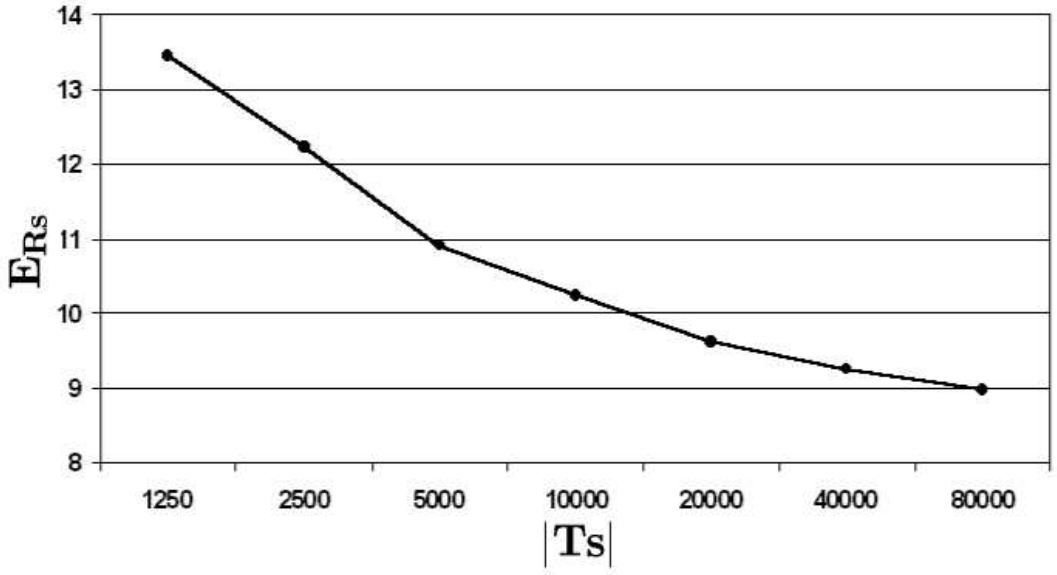


Figura 4.7: Error de generalización E_{Rs} para la red 156-22-16-1, en función de la cantidad de patrones de entrenamiento

Lamentablemente, al no poseer los elementos para la adquisición de imágenes binoculares y estar trabajando con aquellas disponibles en la web, no podemos probar conjuntos de entrenamiento mayores. Si bien aún quedan patrones en Bs no utilizados en el entrenamiento, éstos están muy correlacionados y producirían grandes beneficios. Además, si tuviéramos un conjunto de imágenes más grande, los patrones de Ts estarían menos correlacionados entre si, pudiendo producir mejores resultados incluso para $|Ts|$ más pequeños.

4.4. Comparación de los Resultados

A continuación mostraremos los resultados producidos por la red neuronal y por el método algorítmico, a fin de poder compararlos. En las Figuras 4.10(a), 4.10(c) y 4.10(e) se pueden apreciar los resultados producidos por la red neuronal, a partir del par de imágenes de la Figura 4.9(que pertenece al conjunto de entrenamiento),

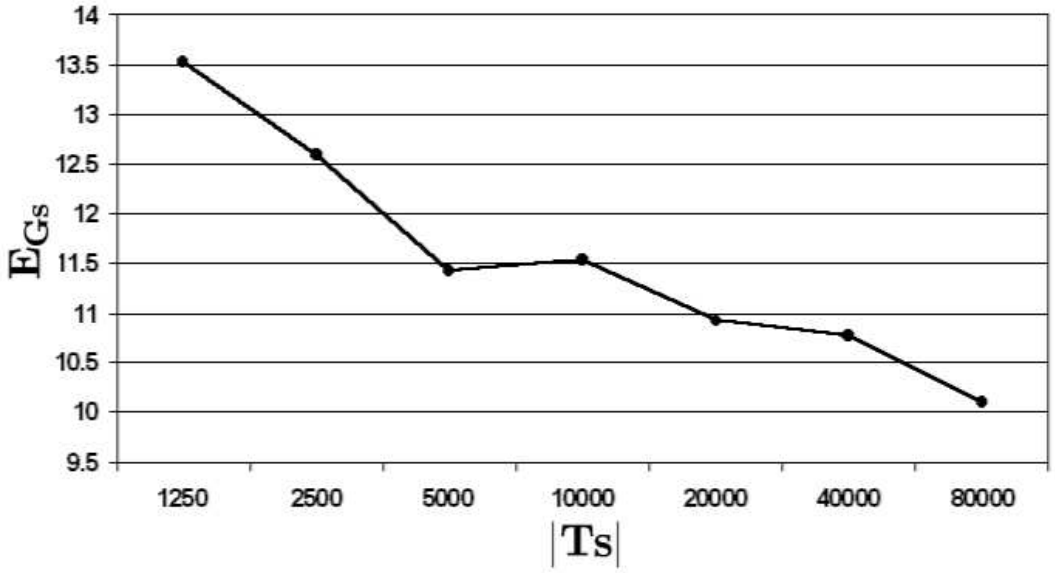
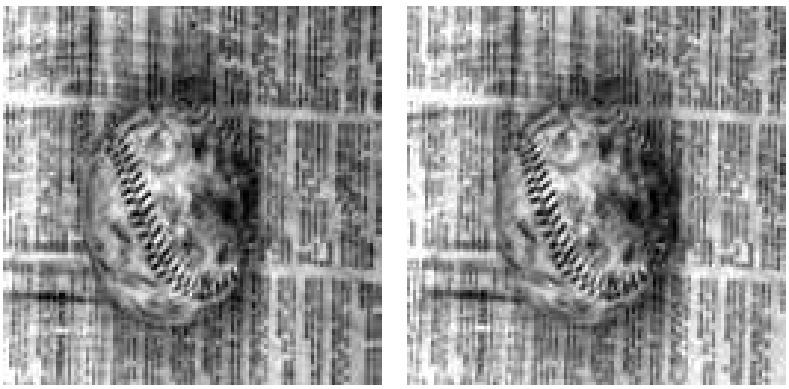


Figura 4.8: Error de generalización E_{Gs} para la red 156-22-16-1, en función de la cantidad de patrones de entrenamiento

sin ningún tipo de validación en los datos y luego de la aplicación del filtro de media descripto en la Sección 3.5 al mapa de alturas. Aproximadamente la mitad de los puntos de esta imagen intervinieron en el entrenamiento. Las Figuras 4.10(b), 4.10(d) y 4.10(f), muestran los resultados de la aplicación del método algorítmico, luego de la validación de los datos y el postprocesamiento descripto en la Sección 3.5, al mismo par de imágenes.

Notemos que en los resultados de la red neuronal se ha cortado un segmento a la derecha del mapa de alturas. Ésto se debe a que la entrada de ésta es de tamaño fijo y por lo tanto no se puede desplazar más allá del borde derecho de las imágenes. En el método algorítmico lo que se hace es disminuir la longitud del segmento de búsqueda a medida que nos aproximamos al borde. De esta forma el último segmento de búsqueda tiene longitud uno. Ésto produce condiciones de borde bastante indeseadas. En el caso neuronal, estos puntos simplemente fueron omitidos.

La Figura 4.12 muestra de manera similar los resultados generados a partir del

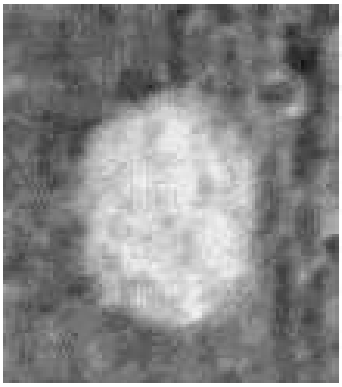


(a) Imagen de la cámara izquierda.

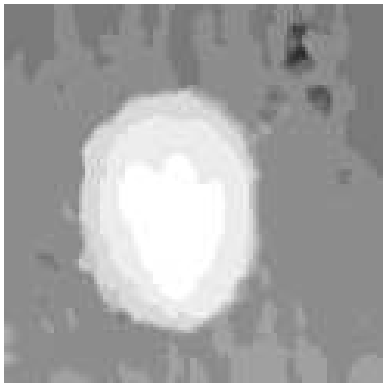
(b) Imagen de la cámara derecha.

Figura 4.9: Par de imágenes de bola de béisbol sobre un periódico.

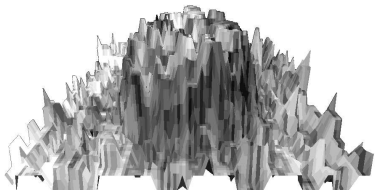
par de imágenes de la Figura 4.11 que no intervino para nada en el entrenamiento.



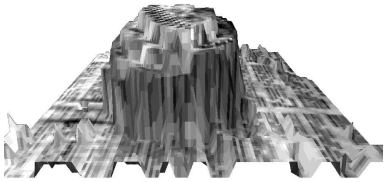
(a) Mapa de alturas generado por la red neuronal.



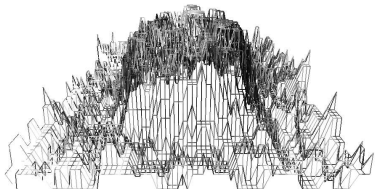
(b) Mapa de alturas generado por el método algorítmico.



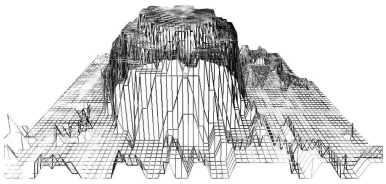
(c) Ambiente 3D generado por la red neuronal.



(d) Ambiente 3D generado por el método algorítmico.

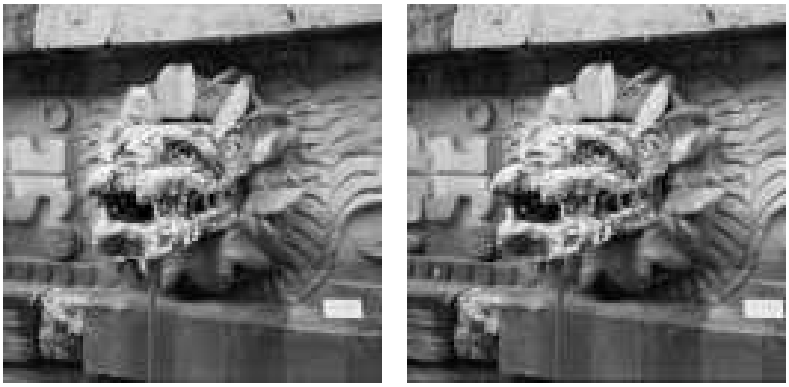


(e) Maya generada por la red neuronal.



(f) Maya generada por el método algorítmico.

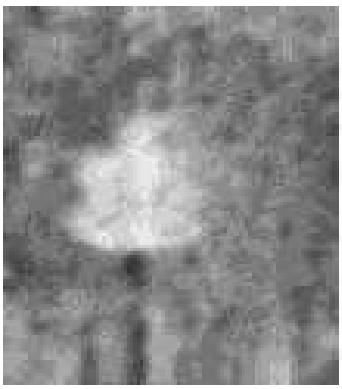
Figura 4.10: Comparación de los resultados en un par de imágenes perteneciente al conjunto de entrenamiento.



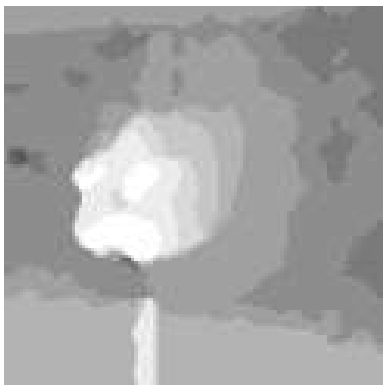
(a) Imagen de la cámara izquierda.

(b) Imagen de la cámara derecha.

Figura 4.11: Par de imágenes de una ruina mexicana.

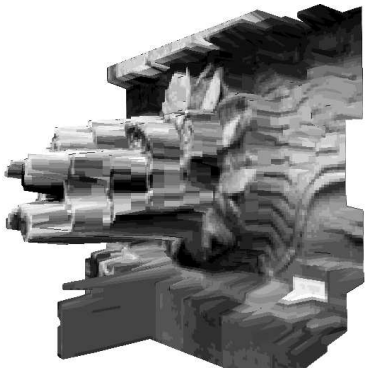


(a) Mapa de alturas generado por la red neuronal.

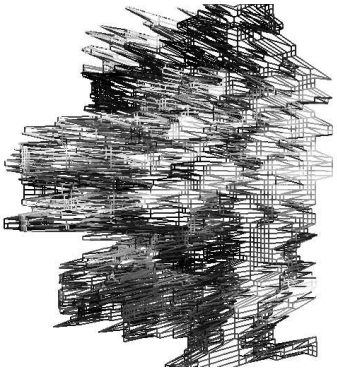


(b) Mapa de alturas generado por el método algorítmico.

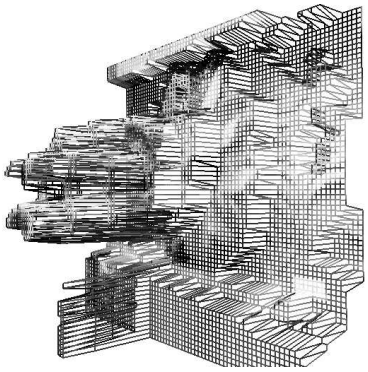
(c) Ambiente 3D generado por la red neuronal.



(d) Ambiente 3D generado por el método algorítmico.



(e) Maya generada por la red neuronal.



(f) Maya generada por el método algorítmico.

Figura 4.12: Comparación de los resultados en un par de imágenes que no intervino en el entrenamiento.

Capítulo 5

Conclusiones

En este trabajo hemos estudiado las redes neuronales artificiales, sus orígenes, su inspiración en los seres vivos, las capacidades y limitaciones que éstas poseen, siempre focalizándonos en la arquitectura feed–forward y el algoritmo de backpropagation.

También hemos estudiado el problema de la estéreo visión, describiendo como se obtienen las profundidades a partir de dos imágenes planas. Presentamos un método bastante acabado, que de manera algorítmica resuelve este problema. Además se describieron métodos para abordar, al menos en la mayoría de los casos, problemas clásicos como los puntos de oclusión y patrones repetitivos.

Finalmente, abordamos el problema de la estéreo visión mediante el uso de redes neuronales. Trabajamos en el desarrollo de una pequeña red, realizando estudios en cuanto a arquitectura, cantidad de neuronas ocultas y tamaño del conjunto de entrenamiento, que nos permitió emular al método algorítmico con cierta precisión. Los resultados de las simulaciones arrojaron errores de generalización que rondan el 10 %. Si bien estos errores son algo elevados, los resultados obtenidos mediante la red neuronal preservan las cualidades básicas de sus pares obtenidos mediante el método algorítmico...pero solo éstas. Con esto queremos decir que la red neuronal puede percibir en que zonas existe gran profundidad y en cuales no. El problema es que se

pierden detalles en las superficies, ya que éstas se ven bastante distorsionadas. Por otra parte, si no se requiere gran precisión en los detalles o sólo se necesita determinar la presencia de un objeto cercano, como por ejemplo en el caso de la navegación de robots, donde la velocidad es prioritaria, la idea de la implementación de la red en hardware se vuelve interesante. Más aún si pensamos que sólo necesitamos 39 neuronas para la implementación de un módulo.

Un factor que seguramente degrada significativamente los resultados, es el hecho de que las imágenes no son tomadas con el mismo ángulo en las cámaras ni con la misma distancia focal. Quizá hubiera sido conveniente adquirir nuestras propias imágenes, pero como lo mencionamos en la Sección 1, en este trabajo sólo intentamos hacer una primera aproximación y para tal fin fueron suficientes las imágenes disponibles en la WWW.

Bibliografía

[Faugeras & Hotz] INRIA: Real Time Correlation-Based Stereo: Algorithm, Implementations and Applications
Olivier Faugeras - Bernard Hotz - Hervé Mathieu - Thierry Viéville Zhengyou Zhang - Pascal Fua - Eric Théron - Laurent Moll Gérard Berry - Jean Vuillemin - Patrice Bertin - Catherine Proy. ISSN 0249-6399. 1993

[Papadimitriou and Dennis] Papadimitriou D.V., Dennis T.J., Epipolar line estimation and rectification for stereo image pairs. IEEE Trans. On image processing, vol. 5(4), 1996.

[Brown & Benchmark] Brown & Benchmark: Introductory Psycology. Times Mirror Higher Education Group. Inc. 1995.

[Hertz] Hertz John, Krogh Anders, Palmer Richard, Introduction to the Theory of Neural Computation. 1991. ISBN 0-201-50395-6

[Happel and Murre] Bart L.M. Happel and Jacob M.J. Murre. The Design and Evolution of Modular Neural Network Architectures.1994.

[WANG AND HSIAO] JUNG-HUA WANG AND CHIH-PING HSIAO. On Disparity Matching in Stereo Vision via a Neural Network Framework. 1999.

[Russ] Russ John C.: The Image Processing Handbook-Third Edition. ISBN 3-540-64747

[Heiko Hirschmüller] Heiko Hirschmüller: Improvements in Real–Time Correlation–Based Stereo Vision

[Cuenca] Cuenca Acuña, F. Matías: Estudio de pronóstico de contaminación ambiental basado en redes neuronales.

[Crane] Crane R.: A Simplified Approach to Image Processing-Clasical and Modern Thechniques in C. ISBN 0-13-226416-1