DISEÑO Y ENTRENAMIENTO EN PARALELO DE REDES NEURONALES, POR MEDIO DE ALGORITMOS GENETICOS DESORDENADOS Y ALTAMENTE RECURSIVOS.

DELIA JOHANNA MUÑOZ V.
Estudiante
Universidad Tecnológica de Pereira.
Delia.johanna@gmail.com

RESUMEN
El siguiente trabajo ataca de manera inusual, dos típicos problemas de optimización, como lo son, el diseño (capas-neuronas) y entrenamiento (acople de pesos) de una red neuronal artificial. Para tal propósito se hace uso de un algoritmo genético adaptado, que en últimas y al menos en teoría, poseerá la capacidad de predecir de manera óptima el diseño y el resultado del entrenamiento de una RNA (Red Neuronal Artificial), para alguna tarea en particular que el programador desee que aprenda. También se considera la gama de dificultades y limitaciones que se generan en el sistema, animando y dando recursos al lector para el desarrollo de una posible evolución de la técnica.

PALABRAS CLAVES: Redes Neuronales, Algoritmos genéticos, optimización.

ABSTRACT
The following work attacks in an unusual way, two typical problems of optimization, as they are it, the design (layer-neurons) and training (it couples of weights) of a net artificial neural. For such a purpose use of an adapted genetic algorithm is made that in last and at least in theory, it will possess the capacity to predict in a good way the design and the result of the training of a RNA, for some task in particular that the programmer wants that he learns. It is also considered the range of difficulties and limitations that are generated in the system, encouraging and giving resources to the reader for the development of a possible evolution of the technique.

KEYWORDS: *Neural Networks, genetics algorithms, optimization.*

1. INTRODUCCION

A parte de la experiencia y probablemente la intuición del diseñador de redes neuronales, no existe ninguna herramienta que le permita determinar con precisión la arquitectura de una red, que aprenda determinada tarea en particular. Talvez nunca se pueda contar con dicha herramienta (optimizada), puesto que este problema es un problema de infinitas soluciones debido a que, por ejemplo, una red que aprenda la tarea OR, pudiese contar por decir algo con 250 capas de 3600 neuronas cada una o con una sola capa de una neurona (realmente).

Lo único entonces que se puede hacer frente a varias estructuras de una red que ejecuta la misma tarea, es clasificarlas entre buenas y malas dependiendo del error que produzcan respecto al set de entrenamiento, en comparación con la tolerancia o margen de error que se esté dispuesto a soportar (entre menos error y menos tamaño de arquitectura, mejor la estructura).

Tanto en teoría como en la práctica son tan inciertos el número de capas, como el número de neuronas por capa, para cada tarea en particular y definitivamente después de muchos intentos al entrenar una red con cierta estructura para alguna tarea específica, al otro lado del mundo alguien pudo entrenar para la misma tarea otra red con diferente estructura y con un error tres veces más pequeño que el primero y talvez hasta menor cantidad de capas y/o neuronas (a no ser que se cuente con mucha experiencia). En resumidas cuentas la única "técnica" que existe actualmente para el anterior fin, es la del ensayo y error, que no necesita describirse para deducir su poca eficiencia.

Luego de la determinación (alguna posible) de la estructura, viene un paso con iguales características y mayores proporciones que el anterior, y es determinar los pesos óptimos para cada una de las conexiones (sinapsis) que resultaron en la red debido a la conformación de la estructura. Estos pesos son números reales y por lo tanto cada uno con posibilidades infinitas. Además si se analiza con calma la verdadera magnitud del problema, se puede ver que es un árbol de infinitos nodos, con infinitas ramas (cada nodo) de probabilidades, el que se abre, debido a que por cada posible estructura que se ensaye (de infinitas posibles), se desprenderán una cantidad igualmente infinita de combinación de pesos óptimos para dicha arquitectura.

Tal vez sea un poco más intuitiva la determinación de capas-neuronas, pero para la configuración de los pesos o

Fecha de Recepción: 31 Enero de 2006
Fecha de Aceptación: 08 Abril de 2006

entrenamiento (proceso más importante) se han desarrollado numerables métodos, que ayude a acoplar estos factores sinápticos de tal modo que la red aprenda la tarea que se necesite. Entre los métodos de entrenamiento, más conocidos, están [1] el de retropropagación (backpropagation) o el de LMS (windrow). Que son métodos heurísticos que describen y escalan hacia un mínimo (por lo regular local), una superficie de error, por medio de la técnica del gradiente descendente.

Por lo tanto y como será obvio en una persona que haya trabajado con algoritmos genéticos, cuando se habla de la búsqueda de "buenas" soluciones en espacios de posibilidades infinitos, esta técnica ha demostrado comportarse de manera aceptable (por no decir buena) a la solución de dichas situaciones problemáticas. Razón en la que se sustenta la idea de la inclusión de un algoritmo genético dentro del proceso de diseño y entrenamiento de sistemas neuronales [2], que determine tanto la estructura de una red como los pesos correspondientes a dicha arquitectura, de manera paralela, de tal forma que dicha red neuronal resuelva o aprenda determinado problema que se quiera atacar.

2. ALGORITMOS GENETICOS

Esta es una técnica heurística [2], relativamente nueva, que como se dijo anteriormente permite hallar soluciones de buena calidad, a problemas donde las opciones de solución son infinitas o por lo menos muy grandes, se inspira en la evolución natural (Darwin) donde generación tras generación los mejor adaptados cuentan con mayor probabilidad de reproducirse y subsistir al punto de entregar parte de su estructura genética a sus descendientes.

El algoritmo genético aplicado a un problema real se basa básicamente en la codificación (normalmente binaria) y los operadores de cruzamiento y mutación. Además la evolución de este se apega a una función objetivo o fittnes que se desea minimizar o maximizar.

La *codificación* es el proceso mediante el cual se representan o se modelan en general las posibles soluciones del problema; mediante la codificación en cadena numérica de longitud fija (genoma). Formando entonces por cada cadena un individuo que no representará nada más que alguna posible solución al problema [2].

El operador de *cruzamiento*, es el que se encarga de recombinar el material genético de los individuos para dar paso a nuevos individuos con características combinadas (genes) de sus padres. Por lo general este proceso de cruzamiento se aplica por parejas (probablemente las más aptas) de individuos con uno o varios puntos de combinación o mezcla.

La *mutación* es la que se encarga de evitar el predominio de individuos rápidamente dominantes (generalmente mínimos locales y no globales del problema), y da pie a un cambio intempestivo en una población, debido al abrupto cambio que se da en algún gen de algún individuo. Claro está, que este hecho ocurre con una probabilidad muy baja en el transcurso del algoritmo, como también sucede en la realidad.

La función *Fittnes*, es aquella con la que se evalúan los individuos, para considerarles más o menos aptos que los demás, para resolver el problema. Por lo regular esta función es la misma que representa en sí, todo el problema de optimización y por ende es la que se desea maximizar o minimizar. Por lo tanto los individuos del algoritmo no son más que representaciones de los elementos de R^n que toma la función.

3. DESCRIPCION FORMAL DEL PROBLEMA

Al mirar ambos problemas (diseño y entrenamiento) como problemas de optimización o búsqueda de buenas soluciones en cantidades imponderables o infinitas de ellas. Seria razonable pensar en solucionar cada uno de ellos con algoritmia genética, pero más pretencioso es tratar de darles solución con un solo algoritmo integrado, de manera paralela y más automatizada.

En este punto se debe recurrir a los Algoritmos genéticos recursivos y desordenados. *Recursivos* (idea original) por el echo de que un algoritmo genético se llamará durante su ejecución, en múltiples ocasiones a él mismo. Y *desordenados* [1], puesto que estos no poseerán longitudes de genomas especificas y mucho menos constantes, debido a que deberán contar con un alto grado de flexibilidad o variabilidad respondiendo a las exigencias de la invocación que se les haga en determinado momento, invocaciones que en ultimas serán múltiples e igualmente variables; además los primeros algoritmos (padres o nodos principales) que invocan nuevos algoritmos del mismo tipo, no poseerán en sí, función de fittnes, sino que dependerán directamente de la de sus hijos o invocados. El sistema se delimita en complejidad recursiva al determinar que ésta solo poseerá dos grados de invocación, como se verá más adelante.

En aras del entendimiento, es preciso en este momento exponer una especie de boceto preliminar del algoritmo general para más adelante especificar minuciosamente en detalles básicos del mismo.

3.1 Boceto y Especificaciones del Algoritmo General

El primer tema o problema a solucionar por parte del algoritmo, es el de diseño. Puesto que sin previo diseño no puede haber entrenamiento, razón por la cual surge el algoritmo genético principal o contenedor, cuya tarea es precisamente encontrar diseños de red e invocar otros AG

(Algoritmo genético) que determinen los pesos de estos. Se ve entonces que los individuos de este algoritmo podrían codificarse de la manera como aparecen en la figura 1.

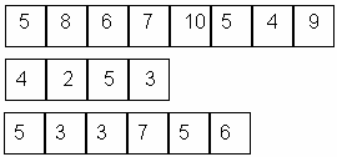

Figura 1. Tres individuos cualesquiera del AG principal

En la figura No 1, se observa el genotipo de los individuos para dicho algoritmo, en donde cada gen, puede tomar valores desde cero hasta el infinito (en teoría el cero y todos los enteros positivos) debido a que cada valor de éste, representa la cantidad de neuronas de una capa de la red, y desde luego la cantidad de genes del individuo, establece la cantidad de capas de la posible red. Así entonces, en la figura 1, el individuo dos, por ejemplo, representará una red de cuatro capas, cada una con, cuatro, dos, cinco y tres neuronas respectivamente, lo que un diseñador nombraría como una estructura de red: 4-2-5-3.

Para este punto el algoritmo genético global, no poseerá aún recursos suficientes para evaluar la calidad de algún individuo. Motivo que le obliga a recurrir por cada individuo, a otro algoritmo genético que encuentre los pesos "óptimos" de éste (entrene); y que retornará a su vez el error que produzca dicho individuo (diseño de estructura) con sus mejores pesos sinápticos (ya encontrados por otro AG), ante el set de entrenamiento. Set que se obtiene dependiendo, claro está, del problema o los patrones que pretendemos que la red aprenda.

Si observamos con cuidado, en este momento, el AG, como se dijo antes, debe llamar por cada individuo otro AG, que encuentre sus pesos y le valore. Pero además en la invocación deberá enviar el tamaño de los individuos (genotipo), que el algoritmo invocado, deberá generar. Y ya que estos individuos no serán ahora, estructuras, sino la representación de los pesos de dicha estructura, su tamaño será mucho más grande y complejo. Por ejemplo para el caso de la estructura (individuo del AG llamador) 4-2-5-3, se deberá invocar recursivamente otro AG que encuentre sus pesos "óptimos" y por ende genere individuos de tamaño: 4*2+2*5+5*3 = 33 c/u (sin contar bias). Motivo que obviamente genera gran robustez en el proceso, teniendo muy presente que bien, si el anterior es, el tamaño de un individuo (genéticamente hablando), cada gen de estos, será un número real, razón que acrecienta aun más la complejidad.

Finalmente se debe resaltar un hecho que no es nada favorable en la reducción y optimización del sistema. Y es: que por efectos de recombinación (en el AG) y para poder crear una gama infinita de números reales (los pesos o individuos del AG entrenador) estos se deben descomponer de dígitos a genes [1]. Ósea, por ejemplo, el 0.236 representará computacionalmente 4 genes: el 0, 2, 3 y 6 respectivamente, hecho que agrandará de manera exponencial el tamaño del individuo (genotipo) de el AG invocado o entrenador. Puesto que como se vio anteriormente, si para la estructura 4-2-5-3 se generaban individuos de 33 genes en el AG invocado, si se suponen reales de 5 dígitos (todos), este tamaño se agrandará en dicho factor 33*5=165 (tamaño final del individuo, aun sin contar bias). El resto es cuestión de interpretar la cadena y saber que cada 5 genes en verdad se tiene un real que representa un peso sináptico, y que cada cierto número de reales se obtienen los pesos de una capa etc.

En la figura 2 se esboza un esquema gráfico (diagrama de bloques) de la evolución del algoritmo hasta encontrar la red óptima que aprenda la función OR.

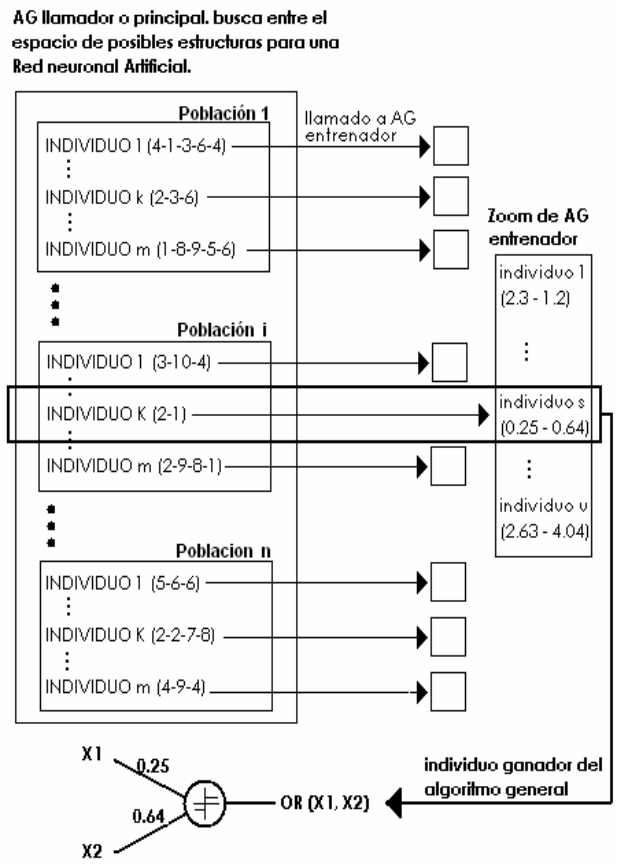

Figura 2. Diagrama de bloques del AG general, para encontrar el diseño y entrenamiento de una RNA que resuelva la función OR.

3.2. Consideraciones Prácticas del Algoritmo

Definitivamente vale la pena considerar, en el AG llamador o principal, el tamaño de sus individuos (genotipo), puesto que para la práctica, en verdad es de vital importancia, el echo que éste AG sea recursivo, mas

no desordenado, puesto que como se podrá intuir más adelante, si no se le da un tamaño fijo a estos individuos (delimitar la cantidad de capas) el problema tomará un curso descomunal en el crecimiento de su complejidad computacional, más aún del que ya posee. Incluso se debe considerar también el hecho de delimitar el rango de variación de cada gen de este individuo, pues por las mismas razones anteriores, no pueden en la práctica variar de cero a infinito (delimitar el número de neuronas por capas) y en verdad se debe generar un tope, para éstas. Obviamente no se excluirá el cero, puesto que al, un gen tomar este valor de cero, lo que indica es una capa nula o la eliminación de la misma (interpretación). Además se debe tener en cuenta que tanto el primer gen como el último de cada individuo, son constantes y predeterminados por el programador, ya que una red define sus neuronas de entrada y de salida, por si sola dependiendo del problema que se esté resolviendo con ella (espacios de los patrones de entrada y salida, del set). A si que el individuo en realidad solo deberá representar la estructura de capas intermedias.

Fíjese también, en el caso de los AG llamados o entrenadores; que es igualmente necesario predeterminar la cantidad de dígitos de cada real que representará un peso sináptico. Obviamente esta debe ser baja, por efectos de simplificación. Tampoco se pueden olvidar los factores de bias de la red (uno por neurona) que igualmente son reales, como los anteriores pesos y deben ser determinados en el entrenamiento, o sea por los AG llamados o invocados.

Lo anterior da pie para proponer el siguiente tamaño e interpretación de un individuo de AG entrenador cuyo llamado fue efectuado por un individuo de AG principal con genotipo: por ejemplo 3-2-2. Los individuos de este AG entrenador contarán con un tamaño final de:
[(3*2*2) + (3+2+2)] * 4 = 76, donde el primer paréntesis contiene la cantidad de pesos de las conexiones totales de la estructura, el segundo, la cantidad de bias de la red, y la multiplicación por cuatro, es suponiendo reales de cuatro dígitos. Se interpretaría bien, este individuo, asumiendo que cada 4 genes hay un peso (real), los primeros 6 pesos que se encuentren (24 genes), son los pesos de la primer capa, los siguientes 4 son los de la segunda capa, y de allí en adelante cada peso que se encuentre se le asignará a una neurona respectivamente como su bias (cada gen variará de 0 a 9).

3.3 Análisis de la complejidad

Obviamente se esta generando un sistema o programa, grande, complejo y humanamente incontrolable, a tal medida que puede desahuciar una computadora en instantes y bloquearla al punto de desbordar toda capacidad de velocidad y procesamiento que esta posea, por más actualizada que sea su tecnología. Por este motivo es de por si, ya un problema, el solo hecho de analizar con mucha precaución los parámetros, limites y tiempos que se deben otorgar al comienzo y tratar de visualizar a grosso modo el alcance del mismo.

El problema de generar miles de posibles estructuras de una red neuronal y para cada una de estas, generar también miles de posibles combinaciones de pesos sinápticos (tarea de los AG), hasta encontrar una buena estructura, con unos buenos pesos, para resolver alguna tarea. Se puede aproximar (la cantidad de posibles soluciones) de manera matemática con la siguiente formula:

$$\sum_{i=1}^{i=10^{(n*cdc)}} 10 \qquad (in*C_1^i + \sum_{u=1}^{u=n} C_u^i * C_{u-1}^i + C_n^i * out) * cdw \qquad (1)$$

Donde:

n = numero de capas intermedias de la red

cdc = cantidad de dígitos de cada capa

cdw = cantidad de dígitos de cada peso

in = tamaño del vector de entrada de la red

out = tamaño del vector de salida de la red

C^i_u = el numero de neuronas de la capa u de la estructura i-esima

El 10 se genera debido a que, es la cantidad de posibilidades de variación de un gen cualquiera, que varía de 0 a 9. La primera sumatoria (grande) se refiere a los individuos del AG llamador, o posibles estructuras (capas-neuronas) y la segunda sumatoria (pequeña) se refiere a los individuos del AG entrenador (pesos de una estructura).

Pero, para dimensionar de manera más contundente aún, la complejidad del sistema se intentará de modo menos general, encontrar el número exacto de posibles soluciones para un caso concreto donde n=3, cdc=3, cdw=4, in=5, out=3. Asombrosamente al realizar un programa en Matlab que calculará la cantidad de soluciones para este problema especifico, según la formula No1, resultó ser un fracaso, ya que la computadora se bloqueo de manera irreversible tratando de realizar el “simple” calculo (simple si lo comparamos con el verdadero problema, que sería generar y evaluar cada una de esas posibilidades que ni siquiera se pudieron enumerar).

Pero es justo, aclarar que lo anteriormente dicho se basa en casos extremos, y que además al trabajar con Algoritmos genéticos, no se tienen que examinar todas las posibles soluciones del problema, al contrario, se supone que el AG propende siempre por acortar el camino (mejorar los individuos de cada generación) hacia la solución optima.

3.4 Pseudo-código del Algoritmo

Para observar de manera resumida y global el funcionamiento del algoritmo, véase en el siguiente pseudo-código y recuérdese que el parámetro n, que recibe un AG al comienzo, es el tamaño en genes (genoma) de cada individuo que generará en su interior. Por lo tanto al principio de todo el proceso, cuando llamemos el AG principal este parámetro será definido por nosotros. Caso contrario en los AG, llamados o entrenadores donde este parámetro variará según el individuo (estructura) del AG llamador que lo invoque.

Por último el parámetro que recibe el AG, también al comienzo, y que se denomina tipo, le indicará al AG, si se debe ejecutar como principal o entrenador.

Algoritmo_ genético (n , tipo)

1. Generar una Población inicial de K individuos de tamaño n*cd con genes variantes de 0 a 9.
// Recordemos que k puede ser cualquier cantidad //predeterminada y que consideremos razonable para el //tamaño de una población, y cd es la cantidad de dígitos // que usemos, que puede ser diferente entre un llamador // y un entrenador.

2. *Si tipo == 0 //principal*
Para i=1, hasta i=k, i de uno en uno…
Evaluación(i)=Algoritmo_genético(tamaño_ind(i),1)
Sino //entrenador
Para i=1, hasta i=k, i de uno en uno…
Evaluación (i)=error de la red (i) respecto al set

3. Asignar las probabilidades de reproducción, de manera proporcional al vector de evaluación de los individuos de la población actual.

4. Generar aleatorios entre 0 y 1 (ruleta) y formar parejas de reproducción dependiendo de los aleatorios y de la probabilidad asignada a cada individuo. Formar tantas parejas como sea suficiente para generar una población de igual tamaño, sabiendo que cada pareja produce 2 hijos.

5. Generar aleatorios enteros entre 1 y n*cd, tantos como parejas hallan, para encontrar el punto de recombinación de cada pareja.

6. Realizar la recombinación mezclando los genes de las parejas hasta formar una nueva generación.

7. Producir un aleatorio entre 0 y 1 y si cae dentro de la probabilidad de mutación, cambiar aleatoriamente algún gen de la generación actual.

8. Si se cumple algún criterio de parada
Retornar el mejor individuo que se halla visto en el algoritmo. *Sino* volver al paso 2.
Obviamente este pseudo-código será más claro para quien tenga una noción sobre algoritmos genéticos. Sino es recomendable estudiar primero la teoría de AG, puesto que no se detalla para nada en pasos que se asumió como obvios para el lector, respecto al proceso general de un AG.

4. CONCLUSIONES

Es innegable que el sistema se puede tornar y se torna lento en ocasiones, incluso en otras, se debe optar por una solución no muy buena. Pero en general para redes de tamaño relativamente pequeño el sistema ha sido en verdad bondadoso y efectivo, además teniendo en cuenta que está susceptible a muchos cambios y mejoras que se le pueden realizar con el estudio más profundo de la teoría de AG.

Es también valido recordar que el tiempo que se demora el sistema para converger a soluciones optimas, se subsana con la comodidad que se le da al entrenador de no tener que hacer prácticamente nada, y dejar que todo sea realizado en paralelo por el sistema, incluso el mortificante paso de ensayo y error para encontrar una estructura, paso que tomaba a veces demasiado tiempo incluso días o semanas. Y además se equilibra el consumo de tiempo, con el también desmesurado consumo que requiere el entrenamiento de una red con un método convencional, que por lo general es de noches y días enteros y en ocasiones semanas.

Se debe tener presente el hecho de que en realidad no se pueden o no es justo, poner a competir o probar de manera formal los métodos tradicionales y el sistema anteriormente descrito, ya que convencionalmente el diseño y entrenamiento de RNA’s con métodos tradicionales, es casi de suerte y azar, probablemente cuando se tarde dos días en finiquitar una red para alguna tarea, alguien lo pudo haber hecho en dos horas y con mejores resultados (redes grandes y tareas complejas). Ahora si se hacen pruebas con redes pequeñas y tareas simples (or, xor, and…), es obvio que de manera tradicional será mas efectivo el entrenamiento en la mayoría de ocasiones, debido a que ya conocemos las arquitecturas básicas y comunes de una red para dichas tareas, mientras el sistema en realidad parte de ningún conocimiento previo.

Lo anterior hace pensar que en verdad nuestro sistema es mucho más útil para redes grandes y de diseño y entrenamiento extenuante, donde no se tiene indicio

alguno de su posible estructura y obviamente menos de los pesos del entrenamiento.

Así que al pararse ante esta balanza de pros y contras entre el sistema aquí planteado y los métodos convencionales, no resta más que dejarse guiar por los gustos, experiencia y requerimientos de cada uno.

5. AGRADECIMIENTOS

El autor brinda sus más sinceros agradecimientos al Ingeniero Luís Alfonso Muñoz de la Universidad Coruniversitaria (Tolima) y Elizabeth Villanueva, a ambos por la ayuda y amor incondicional brindados durante toda su carrera y vida. También al Ingeniero Juan Diego Gómez (UTP) quien dirigió y asesoro el proyecto, además colaboró en el desarrollo de la algoritmia y programación de software, sin su experiencia y orientación nada hubiera resultado.

6. BIBLIOGRAFIA

[1] B.M del Brio, “Redes Neuronales y sistemas difusos” 2da ed. Vol 1. Ed. Zaragoza España: Alfaomega Ra-Ma 2002.

[2] J.R Hilera “Redes Neuronales Artificiales” Ira ed. Vol 1. Ed Madrid España: Alfaomega Ra-Ma 2000.

[3] Nilson “Inteligencia artificial” Ira ed. Vol 1. Ed Madrid España: Alfaomega Ra-Ma 2000.

[4] Ph.d Revuelta “Modelo de representación y procesamiento de movimiento en tiempo real” Universidad de Alicante. España, 2001.

[5] Orallo “introducción a la minería de datos” Ira ed. Vol 1. Ed Madrid España 2004: Pearson (Prentice Hall).