TRABAJO ESPECIAL DE GRADO

DISEÑO DE UN SOFTWARE ENVOLVENTE PARA LA REPRESENTACIÓN DE SIMULACIONES

Presentado ante la Ilustre Universidad Central de Venezuela para optar por el Título de Ingeniero Químico por el Br. Sykora, Vladimir

Caracas, Marzo de 2001

TRABAJO ESPECIAL DE GRADO

DISEÑO DE UN SOFTWARE ENVOLVENTE PARA LA REPRESENTACIÓN DE SIMULACIONES

Tutor Académico: Dra. Rosalba Sciamanna
Tutor Industrial: Dr. Christian Kuhlmann

Presentado ante la Ilustre Universidad Central de Venezuela para optar por el Título de Ingeniero Químico por el Br. Sykora, Vladimir

Caracas, Marzo de 2001

Trabajo Especial de Grado aprobado en nombre de la Universidad por el siguiente jurado:

_____

Dra. Rosalba Sciamanna (Tutor)

_____

Dr. Cesar Pérez (Jurado)

_____

Dr. Luis García (Jurado)

Caracas, 23 de Marzo de 2001

The known is finite, the unknown infinite; intellectually we stand on an islet in the midst of an illimitable ocean of inexplicability. Our business in every generation is to reclaim a little more land.

Thomas H. Huxley

Agradecimientos

A mi madre por su apoyo incondicional. A Daniel Deicas por toda su ayuda.
A Allison Yeske por todo su apoyo, y al Dr. Christian Kuhlmann por toda su dedicación en la empresa Bayer AG.
A la profesora Rosalba Sciamanna por todo el apoyo en la Universidad, así como todo el tiempo dedicado.
A todos, muchas gracias.

Sykora G., Vladimir J.

DISEÑO DE UN SOFTWARE ENVOLVENTE PARA LA REPRESENTACIÓN DE SIMULACIONES

Tutor Académico: Prof. Rosalba Sciamanna. Tutor Industrial: Dr. Christian Kuhlmann. Tesis. Caracas, U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Química. Año 2001, n° pag. 160.

Palabras Claves: Redes Neuronales, Simulaciones, Selección de datos, Programación.

Resumen: En la representación de modelos mediante redes neuronales artificiales, existen casos en los que los datos usados para el entrenamiento presentan regiones críticas ó de alto gradiente; regiones en las cuales la data pronosticada por las redes neuronales se aleja de los resultados esperados. El presente trabajo se realiza con el fin de desarrollar un algoritmo que permita seleccionar datos en las regiones de alto gradiente de una simulación cualquiera, para luego entrenar una serie de redes neuronales con el fin de evaluar su desempeño.

S u m a r i o

En años recientes, la programación no tradicional ha tenido un gran avance, tanto en sus métodos, como en sus aplicaciones. Las redes neuronales artificiales, las cuales se engloban dentro de la programación no tradicional, han sido aplicadas exitosamente en el campo de la ingeniería. Éstos nuevos métodos de programación han logrado reproducir datos altamente ruidosos, y en general, aprender de datos introducidos como entradas. Por otra parte, el uso de simuladores, y la programación por interfases, han logrado reducir el tiempo en que los estudios de diversos procesos de la ingeniería química son realizados.

El presente trabajo de grado tiene como objetivo el desarrollo de un software amigable, bajo ambiente Windows, el cual permita mediante la utilización de las interfases del simulador comercial Aspen Custom Modeler, introducir entradas y obtener salidas que conformen una adecuada colección de datos para representar un modelo. Adicionalmente, se pretende evaluar el desempeño de diversos algoritmos de selección de datos, así como el desempeño de las redes neuronales como herramientas de modelaje.

La programación de dicho software se realizó en lenguaje Visual Basic. La conexión del programa al simulador se efectuó mediante el uso de las interfases del mismo, al igual que para el modelador de redes neuronales artificiales NN Tool 2000 y Excel.

Por otra parte, los algoritmos de selección de datos fueron usados para generar el conjunto de datos representativos de tres modelos de la ingeniería química: un modelo de un Fermentador en estado estacionario, una ecuación altamente no lineal la cual predice el tiempo de residencia de la madera en una refinería de hojuelas de madera (en este caso se usaron tres pares de entradas diferentes); y por último una ecuación no lineal la cual genera dos picos en su representación tridimensional.

Los algoritmos de selección de datos fueron evaluados comparando el desempeño de las redes neuronales artificiales entrenadas con los datos generados por los mismos.

El software resultante es fácil de usar y permite realizar estudios de modelos en un tiempo muy corto. A través del mismo se generó datos de los diferentes modelos usando dos tipos de algoritmos de generación de datos: el algoritmo de rejilla al azar, y el algoritmo de refinación.

De los resultados obtenidos se tiene que las redes neuronales entrenadas con los datos generados por el algoritmo de refinación dieron, en el 75% de los casos estudiados, menor error que sus contrapartes generadas con el algoritmo de rejilla al azar, cuando los pronósticos son realizados en zonas de alto gradiente.

Por otra parte, en el modelo del Fermentador se obtuvo un error porcentual ponderado de sus promedios de 1,68% y 17,17% para la primera y segunda salida respectivamente (X y S) estudiadas con las redes neuronales entrenadas utilizando la colección generada con el algoritmo de refinación; y 1,31% y 5,6% para la primera y segunda salida respectivamente estudiada con las redes neuronales entrenadas empleando la colección generada con el algoritmo de rejilla al azar.

Por otro lado, con la ecuación no lineal se obtuvo 0,14% y 0,4% de error promedio en las redes neuronales entrenadas utilizando la colección generada con el algoritmo de refinación y con el algoritmo de rejilla al azar respectivamente, para las primeras dos variables estudiadas (μ_t y μ_r); para las dos siguientes variables estudiadas (r_1 y r_2) se obtuvo 0,38% y 0,32% respectivamente; y 0,96% y 0,73% para las últimas dos variables estudiadas (ω y c).

Con la ecuación de dos picos, la mayor correlación que se obtuvo fue 0,97 para las redes neuronales entrenadas utilizando la colección generada con el algoritmo de refinación, y la mayor correlación fue de 0,91 para las redes neuronales estudiadas entrenadas empleando la colección generada con el algoritmo de rejilla al azar.

En resumen, del estudio realizado se concluye que las redes neuronales entrenadas utilizando la colección generada con el algoritmo de refinación hacen mejores predicciones en las zonas de alto gradiente. Adicionalmente se

concluye que las redes neuronales artificiales son un buen método para la representación de modelos sencillos de la ingeniería química, aun cuando exista poca cantidad de series de datos para el entrenamiento.

Índice

Índice de Figuras

Anexos

Índice de Tablas

I. Introducción

Con el advenimiento de la computación moderna se han abierto cantidad de posibilidades y herramientas, no sólo para la resolución de problemas en la ingeniería, sino que han trascendido para crear un método en la toma de decisiones, así como en las optimizaciones. El campo de la ingeniería ha reconocido el poder, así como el alcance de éstas nuevas herramientas computacionales, aplicadas con éxito en sus diversas ramas, llegando a obtener muy buenos resultados.

Las herramientas computacionales juegan y seguirán jugando un papel fundamental en la ingeniería, desde la toma de decisiones para ubicar los centros de distribución en una cadena de suministros, hasta grandes modificaciones en plantas como resultado de complejas optimizaciones.

En años recientes, se ha dado aplicación a ciertos sistemas, englobados dentro de la inteligencia artificial, los cuales prometen aumentar aún más el poder de las herramientas que el ingeniero moderno dispone para enfrentar los problemas que a diario se le presentan.

Las herramientas computacionales las cuales se engloban dentro del campo de la inteligencia artificial, y las cuales fueron usadas en el presente trabajo de grado, son las llamadas *Redes Neuronales Artificiales*. Cabe señalar que aunque el comportamiento y alcance de las redes neuronales se aleja de lo que todos estamos acostumbrados a concebir como “inteligencia artificial”, éstas salen del estudio de la misma, es decir, tuvieron como origen el estudio de la inteligencia artificial.

Por otro lado, las redes neuronales artificiales tienen la capacidad de predecir, clasificar, asociar data, conceptualizar data, y filtrar data valiosa de conjuntos ruidosos. Sólo enumerando éstas cinco categorías de aplicaciones de las redes neuronales artificiales, se abren posibilidades inmensas para la ingeniería.

El presente trabajo de grado tiene como objetivo construir un software que permita crear colecciones de datos, mediante el uso de un simulador comercial, pero con la posibilidad de poder generar series de datos adicionales en las

regiones donde el modelo presente gradientes mayores que un valor introducido por el usuario, y con ellos, crear automáticamente redes neuronales.

Las redes neuronales artificiales son creadas mediante la utilización de NN Tool 2000®, el cual es un programa que permite crear y modelar redes neuronales artificiales. Una vez creadas las redes neuronales, el programa permitirá hacer pronósticos con éstas, y una vez calculando el valor real de las salidas mediante el uso del simulador, éste permite calcular la suma del cuadrado de los errores absolutos generados en los pronósticos hechos por cada red neuronal.

Por lo tanto, el presente trabajo engloba algunos de los nuevos sistemas que han resultado de gran utilidad en el campo de la ingeniería; el ingeniero que desea dar la talla en el mundo competitivo de hoy en día tiene que adaptarse a los cambios y dominar éstas nuevas técnicas, las cuales le han dado a la profesión otro enfoque, uno más eficiente, más limpio, optimizado.

Es seguro que el ingeniero del futuro tendrá en sus manos herramientas mucho más poderosas que las actuales, para poder enfrentar los retos cada vez más desafiantes del mundo competitivo de hoy.

II. Objetivos

A continuación se presentan los objetivos generales, así como los específicos del presente trabajo especial de grado:

Objetivos Generales

- Desarrollar un programa amigable y bajo ambiente Windows, que se acople a un simulador comercial (Aspen Custom Modeler®), el cual permita enviar entradas y obtener salidas, creando así una adecuada colección de datos del modelo. Adicionalmente, se quiere evaluar el desempeño de diversos algoritmos de selección de data, así como el desempeño de las redes neurales como herramientas de modelaje.

Objetivos Específicos

- Crear las interfases envolvente-simulador necesarias.
- Desarrollar el software envolvente.
- Desarrollar los algoritmos de selección. Evaluar su desempeño.
- Evaluar el desempeño de redes neurales como herramientas de modelaje de procesos de la ingeniería química.
- Evaluar las colecciones de datos usadas para el entrenamiento de las redes neurales.

III. Revisión Bibliográfica

En años recientes, ha habido gran interés por parte de ingenieros en el desarrollo de métodos para lograr mejores soluciones de problemas típicos ingenieriles en el menor tiempo posible. En el desarrollo de éstos nuevos métodos tiene que haber un compromiso entre la sencillez del sistema a desarrollar y la eficacia del método.

Ingenieros, matemáticos, y físicos han compartido el interés en las ciencias biológicas, buscando en éstas la inspiración de nuevas ideas, métodos y diseños.

Las redes neuronales artificiales han sido sin duda inspiradas en los sistemas biológicos, pero la correspondencia entre éstas y sus contrapartes biológicas ha sido siempre algo débil [Zurada, 1992]. Existen grandes discrepancias entre las arquitecturas y capacidades de las redes neurales artificiales y biológicas. A parte, el flujo de conocimiento e inspiración entre los sistemas biológicos y artificiales ha sido casi exclusivamente unidireccional [Goldberg, 1989].

El reciente entusiasmo en las redes neuronales artificiales se basa en que han tenido éxito en donde otros tipos de programación han fallado. Por ejemplo, los algoritmos tradicionales de programación son eficientes en cálculos aritméticos rápidos, así como en realizar tareas específicas para los cuales fueron programados; en cambio, fallan cuando se presentan con datos ruidosos ó datos del entorno, en el paralelismo masivo, tolerando fallas, así como en adaptarse a las circunstancias. Es en éstos puntos, en donde la programación tradicional falla, donde las redes neuronales se fortalecen. Éstos sistemas han sido efectivos donde no se puede formular una solución algorítmica, donde se puede extraer grandes cantidades de datos del comportamiento que se requiere, en los sistemas de datos altamente ruidosos, y donde se necesita extraer el comportamiento de ciertos datos [Smith, 1996].

Los sistemas neuronales artificiales han sido aplicados con éxito en áreas como: [Neurodimension Inc. (www.neurodimension.com), 1997]:

- Control de Calidad
- Predicciones Financieras
- Predicciones Económicas
- Reconocimiento del habla así como de patrones
- Instrumentación Biomédica
- Modelaje y Gerencia de Procesos
- Investigaciones de laboratorios
- Exploración de petróleo y gas
- Predicción de bancarrotas
- Diagnóstico de Maquinarias

Los sistemas neuronales artificiales, o redes neuronales, son sistemas celulares físicos, los cuales pueden adquirir, almacenar, y utilizar conocimiento experimental.

Personas y animales son mejores y más rápidos que las computadoras más poderosas en ciertos procesamientos, como por ejemplo el reconocimiento de imágenes. Aunque las computadoras superan a los sistemas neuronales biológicos y artificiales en tareas basadas en operaciones aritméticas precisas y rápidas, los sistemas neuronales artificiales representan la prometedora nueva generación de redes de procesamiento de información. Ciertos avances se han llevado a cabo en aplicar tales sistemas a problemas intransables o demasiado difíciles para la programación tradicional [Zurada, 1992].

La habilidad de las redes neuronales artificiales de realizar cómputos se basa en la esperanza de reproducir cierta flexibilidad y poder del cerebro humano en términos artificiales. La programación neuronal es realizada por una densa red de nodos computacionales y conexiones. Éstas operan colectiva y simultáneamente en la mayoría ó en la totalidad de los datos a procesar y de las entradas al sistema.

Los elementos de procesamiento básicos de una red neuronal son llamados neuronas artificiales ó simplemente neuronas. Generalmente se les llaman simplemente nodos. En ciertos casos pueden ser consideradas como unidades

de umbral que emiten una salida cuando la entrada total excede cierto valor. Las neuronas usualmente operan en paralelo y son configuradas en arquitecturas regulares. Son generalmente organizadas en capas y permiten conexiones con otras neuronas de su misma capa y con otras de capas posteriores. Cada fuerza de la conexión es expresada por un valor numérico llamado peso, el cual puede ser modificado [Zurada, 1992].

Los sistemas neuronales artificiales funcionan como redes computacionales distribuidas paralelamente. Su característica más básica es una arquitectura particular. Las redes neuronales difieren entre ellas según el método de aprendizaje y su arquitectura. Existe una variedad de métodos de aprendizaje las cuales establecen cuándo y dónde los pesos entre las conexiones cambian. Las redes neuronales artificiales (RNA) también exhiben diferentes velocidades y eficiencias en el aprendizaje. Como resultado también difieren en la habilidad de responder acertadamente a los datos introducidos como entradas [Zurada, 1992].

En contraste con los algoritmos tradicionales, los cuales son programados para realizar tareas específicas, la mayoría de las RNA deben ser entrenadas ó enseñadas. El aprendizaje corresponde al cambio de parámetros dentro de la estructura (generalmente los pesos, como se verá posteriormente). Las reglas de aprendizaje y los algoritmos usados para el entrenamiento experimental remplazan a la programación requerida en los algoritmos convencionales. Los usuarios de las RNA no especifican el algoritmo a ser ejecutado por cada nodo computacional como lo haría un programador de un algoritmo tradicional. En cambio, el usuario selecciona con su criterio la mejor arquitectura, especifica las características de las neuronas y los pesos iniciales, y elige el método de entrenamiento de la red. Luego son aplicadas *entradas* (*inputs*) apropiados a la red con el propósito de que adquiera conocimiento del entorno. Como resultado de ésta exposición, la red asimila información que luego puede ser obtenida por el usuario [Zurada, 1992].

A pesar de las discrepancias entre los sistemas biológicos y artificiales, a continuación se presentará brevemente cómo funcionan los sistemas neuronales

biológicos de los cuales los artificiales tuvieron un aire de inspiración, para luego describir sus contrapartes artificiales.

III.1. La Neurona [Müller y Reinhardt, 1990]

Las investigaciones detalladas de la estructura interna de las células neuronales, especialmente después de la invención del microscopio electrónico, han revelado que todas las neuronas están constituidas de las mismas estructuras básicas, independientemente de su tamaño y forma (Figura 1). La parte central abultada es llamada cuerpo de la célula ó *soma*: de éste punto se proyectan varias extensiones llamadas *dendritas*, así como una fibra tubular llamada *axón*, la cual se divide en su extremo formando diversas ramas. El tamaño del *soma* de una neurona típica es de 10 a 80 μm, mientras las dendritas y el axón tienen un diámetro de pocos μm. Las dendritas sirven de receptores de las señales emitidas por las neuronas adyacentes, el propósito del axón es transmitir la actividad neuronal generada hacia otras células nerviosas ó hacia fibras musculares.

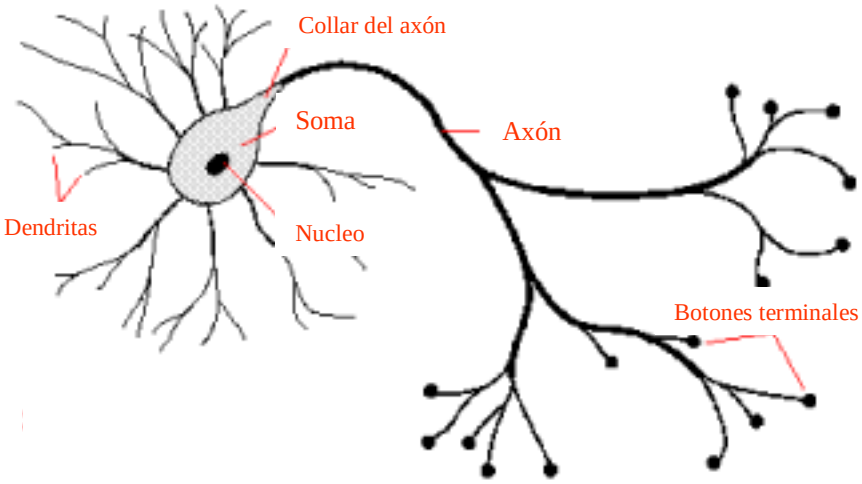


Figura 1. Neurona Biológica [Fraser, 1998]

La unión entre el final de una rama del axón y otra neurona o músculo es llamada *sinapsis*. La sinapsis puede ser localizada directamente en el cuerpo de la célula ó en las dendritas de la neurona subsiguiente; su fuerza de influencia generalmente decrece con la distancia desde el cuerpo celular. La longitud de las neuronas presenta grandes variaciones: de 0.01 mm para las interneuronas del cerebro humano, hasta 1m para neuronas en las extremidades.

Las señales nerviosas son transmitidas por señales eléctricas o químicas. Las transmisiones eléctricas prevalecen en el interior de la neurona, mientras que los mecanismos químicos operan entre las diferentes neuronas, i.e. en las sinapsis. La transmisión eléctrica se realiza mediante una descarga que empieza en el cuerpo de la célula y luego viaja a través del axón hasta las diferentes conexiones sinápticas. En estado de inactividad el interior de la neurona, el *protoplasma*, permanece cargado negativamente en contra del líquido neuronal que la rodea. Éste potencial de reposo es soportado por la acción de la membrana celular, la cual es impenetrable por los iones Na^+, causando una deficiencia de iones positivos en el protoplasma (Figura 2).

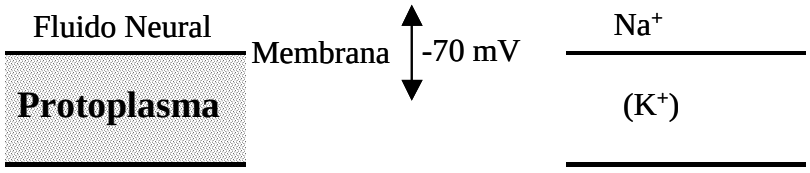


Figura 2. Estructura del axón. [Müller y Reinhardt,1990]

Las señales que llegan de las conexiones sinápticas provenientes de otras neuronas, producen una debilitación temporal, o *despolarización*, del potencial de reposo. Cuando éste es reducido a menos de –60 mV, la membrana pierde su impermeabilidad en contra de los iones Na^+, los cuales entran al protoplasma, neutralizando la diferencia de potencial. Ésta descarga suele ser tan poderosa que el interior de la neurona adquiere un pequeño potencial positivo en contra de su medio. Luego la membrana recupera gradualmente sus propiedades originales y regenera el potencial de reposo en un período de varios

milisegundos. Durante éste período de recuperación, la neurona permanece incapaz de ser excitada. Cuando la recuperación se ha completado, la neurona está en estado de reposo y puede “disparar” de nuevo.

La descarga, la cual ocurre inicialmente en el cuerpo de la neurona, se propaga a través del axón a las sinapsis. Ya que las partes despolarizadas de la neurona se encuentran en estado de recuperación y no pueden ser activadas inmediatamente, el pulso de actividad eléctrica siempre se propaga hacia una sola dirección: alejándose del cuerpo celular. La intensidad de la señal transmitida no decae mientras se propaga a través de la fibra nerviosa, ya que la descarga de cada nuevo segmento del axón es siempre completa.

La señal de descarga viajando a través del axón, se detiene cuando llega a la sinapsis, ya que allí no existe un puente conductor a la próxima neurona o fibra muscular. La transmisión de la señal a través del estrecho sináptico es efectuada por mecanismos químicos. En la transmisión química, cuando el pico de la señal llega al nervio terminal pre-sináptico, unas sustancias especiales llamadas *neurotransmisores* son liberadas en pequeñas cantidades desde vesículas contenidas en el botón terminal. Las moléculas de los neurotransmisores viajan a través del estrecho sináptico, como se muestra en la figura 3, alcanzando a la neurona post-sináptica o una fibra nerviosa. Al llegar éstas sustancias a receptores especiales, modifican la conductancia de la membrana post-sináptica para algunos iones, los cuales fluyen hacia adentro o hacia afuera de la neurona, causando una polarización o despolarización del potencial post-sináptico local.

Si el potencial de polarización inducido δU es positivo, i.e. si la fuerza total del potencial de reposo es reducida, la sinapsis es llamada excitatoria, ya que la influencia de la sinapsis tiende a activar la neurona post-sináptica. Si δU es negativo, la sinapsis es llamada inhibitoria, ya que contrarresta la excitación de la neurona.

La influencia de una sinapsis dada depende de varios aspectos: la fuerza inherente de su efecto despolarizante, su localización respecto al cuerpo de la célula, y la razón de repetición de las señales que llegan. Existe gran cantidad

de evidencia que indica que la fuerza inherente de una sinapsis no es fijada de una vez para siempre. La fuerza de una conexión sináptica puede ser ajustada, si su nivel de actividad cambia [Hebb, 1949]. Éste mecanismo de *plasticidad sináptica* en la estructura de la conectividad neuronal, conocido como la regla de Hebb, aparenta jugar un papel dominante en el complejo proceso de aprendizaje.

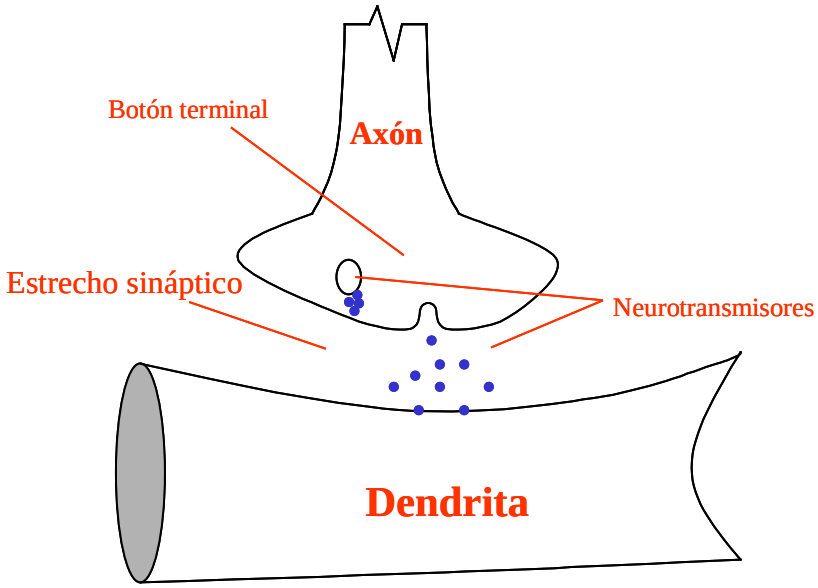


Figura 3. Esquema de la sinápsis. [Fraser, 1998]

III.2. La Corteza Cerebral [Müller y Reinhardt, 1990]

La increíble complejidad del sistema nervioso central humano, específicamente del cerebro humano, se fundamenta no sólo en la complejidad y diversidad de la célula nerviosa, la cual es algo limitada, sino que también en el vasto número de sus unidades constituyentes, i.e. de las neuronas y sus mutuas conexiones. Existen muchos tipos diferentes de neuronas presentes en el cerebro, distinguidas por su tamaño y grado de ramificación del árbol dendrítico y la longitud del axón, entre otros detalles estructurales.

En la corteza cerebral, se estima que cada neurona recibe un average de 10.000 sinapsis. Por otro lado, cada célula descarga su salida a varios cientos de otras

neuronas, y generalmente un número grande de sinapsis conectan a una sola célula nerviosa.

El número total de neuronas en la corteza cerebral es inmenso y sólo puede ser estimado. Existen estimaciones que cuentan 10^{11} el número de neuronas en el sistema nervioso central humano. Combinado con un número promedio de sinapsis por neurona, se alcanza una cifra de 10^{15} conexiones sinápticas en el cerebro humano, mayoría de las cuales se desarrollan unas semanas después del nacimiento.

III.3. Red Neuronal Artificial [Zurada, 1992]

Una vez que se tiene en mente el funcionamiento de los sistemas neuronales biológicos, se procederá a continuación a hacer una descripción general de los sistemas neuronales artificiales. No es difícil notar los principios más básicos de funcionamiento de los sistemas biológicos en los sistemas artificiales (Figura 4), aunque como ya se ha mencionado, sólo han funcionado como forma de inspiración en los mecanismos y arquitecturas.

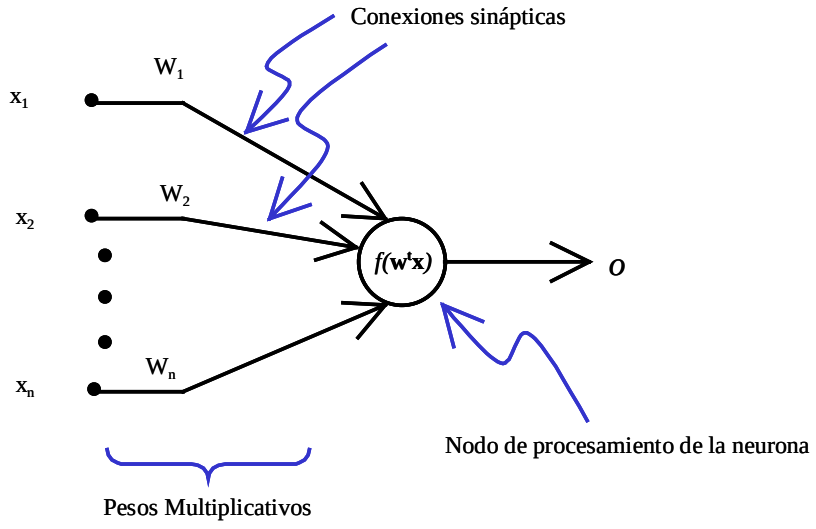


Figura 4. Esquema general de una neurona artificial. [Zurada, 1992]

Cada modelo neuronal consiste de un elemento procesador con conexiones sinápticas de entrada y una sola salida. El flujo de señales neuronales de entrada, x_i, es considerado unidireccional como es indicado por el sentido de las flechas, al igual que el flujo de transmisiones de salida de una neurona biológica. La simbología general de una neurona artificial es mostrada en la figura 4.

En ésta representación simbólica, se muestra una serie de *pesos* (W_j) y la unidad de procesamiento de la neurona ó *nodo*. La señal de salida de la neurona es dada por la siguiente relación:

$$o = f\left(\mathbf{w}^t\mathbf{x}\right), \text{ ó}$$

$$o = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{1}$$

donde **w** es el vector de pesos definido como:

$$\mathbf{w} = \left[\mathrm{w}_1 \;\; \mathrm{w}_2 \;\; \ldots \;\; \mathrm{w}_\mathrm{n}\right]^t \tag{2}$$

y **x** es el vector de entradas:

$$\mathbf{x} = \left[\mathrm{x}_1 \;\; \mathrm{x}_2 \;\; \ldots \;\; \mathrm{x}_\mathrm{n}\right]^t \tag{3}$$

(Todos los vectores definidos son vectores columna, el superíndice *t* denota una transposición). La función $f(\mathbf{w}^t\mathbf{x})$ es frecuentemente referida como función de activación ó función de transferencia. Su dominio es el conjunto de valores de activación, "*red*", del modelo neuronal; generalmente se denota como *f(red)*. La variable *red* es definida como el producto escalar del los vectores peso y entradas:

$$red = \mathbf{w}^t\mathbf{x} \tag{4}$$

El argumento de la función de activación, la variable *red*, es análogo al potencial de la membrana en una neurona biológica. En esta definición, no está usado explícitamente el término umbral. Se ha asumido temporalmente que la neurona modelada tiene realmente *n-1* conexiones sinápticas que provienen de variables de entrada $x_1, x_2, ..., x_{n-1}$. Se ha asumido también que $x_n = -1$ y $w_n = U$, donde U es el valor umbral de la neurona. Éste valor umbral generalmente viene expresado por el valor *bias*, que es un valor de entrada a la neurona.

A partir de éste momento, cada vez que se cite el término *neurona* se refiere a modelos de neuronas *artificiales*; y cada vez que se cite el término *redes neuronales* o *red neuronal* se refiere a redes neuronales artificiales, las cuales consisten de modelos neuronales.

Se observa por la ecuación (1), que la neurona como nodo de procesamiento realiza la operación de suma de las entradas *"pesadas"*, o el producto escalar para obtener el argumento *red*. Luego de obtener el argumento de la función de activación, el nodo realiza el cálculo de la función al argumento.

En el estudio de las redes neuronales se emplean diversos tipos de funciones de activación, entre las más usadas se encuentran [Demuth y Beale., 1992]:

- Hard limit
- Hard limit simétrica
- Sigmoidal logarítmica
- Lineal positiva
- Lineal saturable
- Lineal simétrica saturable
- Sigmoidal tangente hiperbólica

En el presente trabajo se usó la función de activación sigmoide, la cual viene expresada por la siguiente fórmula:

$$f(red) = \frac{1}{1 + e^{(-\lambda \, \mathrm{red})}} \qquad (5)$$

usando λ=1, produce la gráfica mostrada en la figura 5.

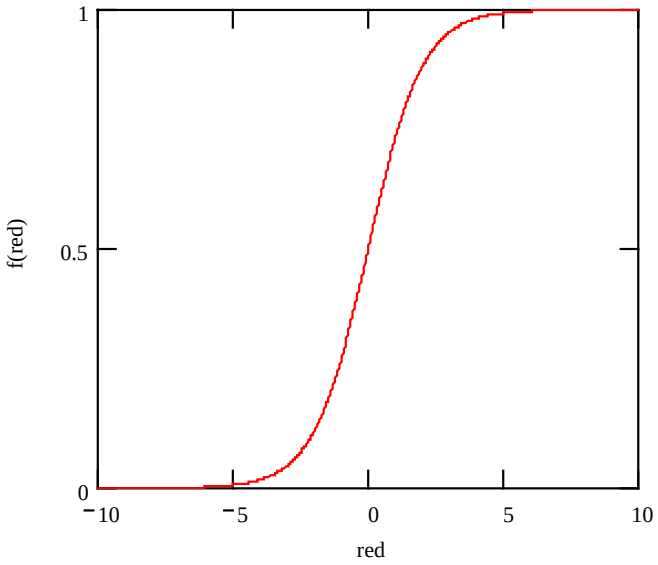


Figura 5. Función Sigmoide. [NN Tool 2000® Handbuch]

el término λ es siempre mayor que cero, y es proporcional a la ganancia de la neurona, determinando la inclinación de la función continua *f(red)* cerca de *red* = 0.

La función de activación generalmente es fijada por los requerimientos de la arquitectura de la red neuronal. Las funciones por lo general son monótonamente ascendentes, y sus rangos están acotados entre [-1, 1]. A continuación se presentará la definición de un perceptrón, el cual es la unidad básica de las redes neuronales artificiales.

III.4. Perceptrón

Las funciones de activación pueden ser discretas o continuas. Un ejemplo de función discreta es la siguiente:

$$f(red) = \begin{cases} +1, & red > 0 \\ -1, & red < 0 \end{cases} \quad \text{bipolar discreta} \tag{6}$$

Ésta función es llamada bipolar discreta; bipolar, ya que la respuesta de la función puede ser tanto positiva como negativa.

La ecuación (6) representa una función continua. Si escalamos y trasladamos su gráfica, podremos obtener una función continua bipolar, como:

$$f(red) = \frac{2}{1+e^{(-\lambda \, \mathrm{red})}} - 1 \tag{7}$$

En el caso de ésta función, el modelo usado se muestra en la figura 6.

La neurona es una amplificación saturable de alta ganancia de la sumatoria, la cual amplifica la señal de entrada $\mathbf{w}^t\mathbf{x}$.

Los modelos en las figuras 6 y 7, pueden ser llamadas perceptrons continuo y discreto respectivamente. El perceptrón discreto fue la primera máquina en aprender.

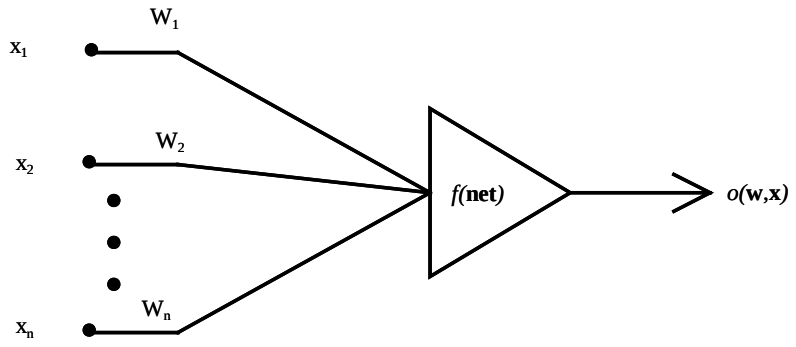


Figura 6. Esquema del Perceptrón Contínuo. [Zurada, 1992]

Cuando se usa la ecuación (6), el diagrama neuronal se suele representar de la forma que se muestra en la figura 7.

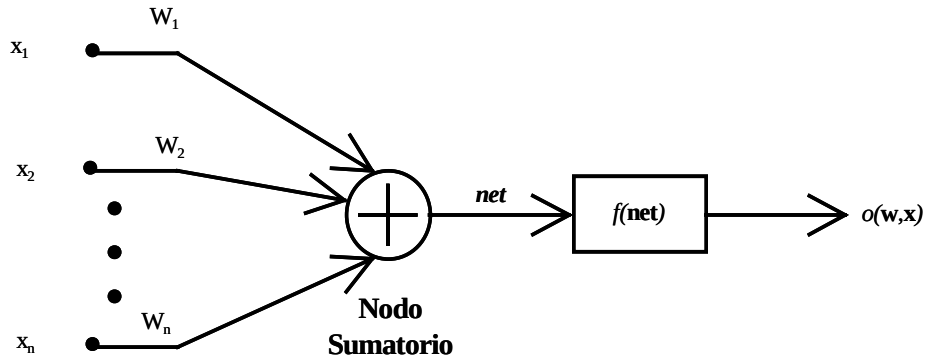


Figura 7. Perceptrón binario. [Zurada, 1992]

III.5. Modelos de Redes Neuronales Artificiales

Las redes neuronales artificiales pueden ser definidas como una interconexión de neuronas (como las definidas en el punto anterior), en la cual cada salida de las neuronas constitutivas están conectadas, a través de pesos, a todas las demás neuronas, incluyendo ella misma; conexiones con retraso y sin retraso son permitidas. El retraso es un factor de tiempo el cual demora la señal de la neurona.

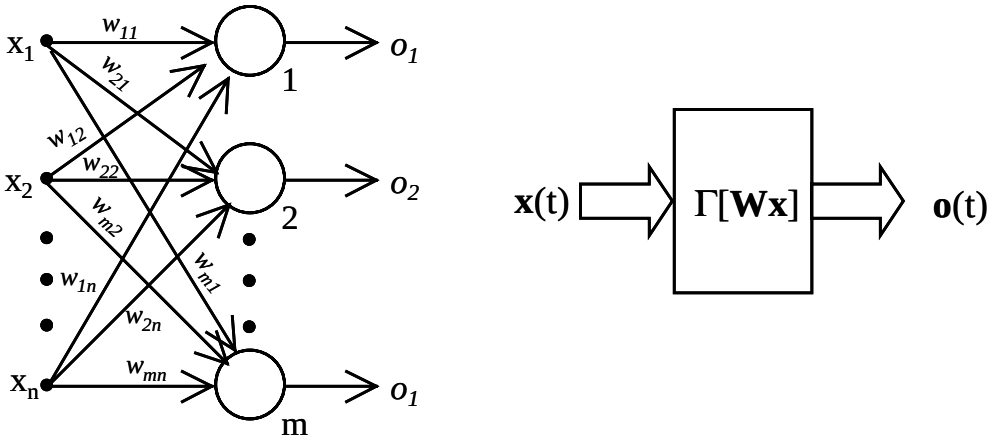
III.6. Red Feedforward

Una arquitectura *feedforward* elemental se presenta en la figura 8 (a), conteniendo *m* neuronas recibiendo *n* entradas; en la figura 8 (b) se muestra el diagrama de bloque. Sus vectores de salida y de entrada son respectivamente:

$$\mathbf{o} = [o_1 \; o_2 \; \ldots \; o_m]^t \tag{8}$$

$$\mathbf{x} = [x_1 \; x_2 \; \ldots \; x_n]^t \tag{9}$$

El peso W_{ij} conecta la i-esima neurona con la j-esima entrada. El doble subíndice denota que el primero y el segundo representan los nodos de destino y salida



(a) (b)

Figura 8. Esquema de una red neuronal artificial feedforward.
(a) Con Interconexiones, (b) En diagrama de bloque. [Zurada, 1992]

respectivamente. Se puede escribir el valor del argumento de activación *red* para la i-esima neurona como:

$$red = \sum_{j=1}^{n} w_{ij} x_j \text{ , para i} = 1, 2, \ldots, \text{m} \qquad (10)$$

La transformación no lineal de la ecuación (11) involucrando la función de activación $f(red_i)$, para i=1, 2, ..., m, completa el procesamiento de la señal de entrada **x**. La transformación realizada por cada una de las *m* neuronas en la red, es un mapeo altamente no lineal, expresado como:

$$o_i = f(\mathbf{w}_i^{\text{t}}\mathbf{x}) \text{ , para i} = 1, 2, \ldots, \text{m} \qquad (11)$$

donde el vector de pesos $\mathbf{W}_i$ contiene los pesos que se dirigen hacia el i-esimo nodo, y es definido como:

$$\mathbf{w}_i = [w_{i1} \;\; w_{i2} \;\; \ldots \;\; w_{in}]^t \tag{12}$$

Introduciendo el operador no lineal de matrices $\mathbf{\Gamma}$, el mapeo de las señales de entrada "*x*" hacia el de salidas "*o*" implementado por la red, puede ser expresado como:

$$\mathbf{o} = \Gamma[\mathbf{W}\mathbf{x}] \tag{13}$$

donde **W** es la matriz de pesos, también llamada matriz de conexión:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \tag{14}$$

Los vectores **o** y **x** son generalmente llamados *patrones* de entrada y de salida respectivamente. Las redes neuronales feedforward presentadas son del tipo instantáneas, ya que no involucran retraso de tiempo entre la entrada **x**, y la salida **o**. Se puede re-escribir la ecuación (13) en forma explícita involucrando retraso de tiempo:

$$\mathbf{o}(t) = \Gamma[\mathbf{W}\mathbf{x}(t)] \tag{15}$$

La figura 8 (b), presenta el diagrama de bloques de una red neuronal feedforward. Como puede verse, la red feedforward es caracterizada por la ausencia de retroalimentación. En este tipo de redes, la salida de una capa es la entrada de la siguiente capa.

Todas las redes neurales poseen una capa de entrada, al menos una (a veces más) capa escondida, y una capa de salida. Cada capa es esencial para el éxito de la red neuronal. En definitiva, una red neural en ojos de un ajente externo, puede ser vista como una caja negra a la cual se envía una entrada específica a todos los nodos de la capa de entrada. La red neuronal procesa ésta información a través de las interconexiones entre los nodos (toda la etapa de procesamiento es escondida del usuario). Finalmente, la red neuronal produce una salida final, la cual resulta de los nodos de la capa de salida. Los objetivos de cada capa pueden ser descritos como: [Quantrille y Liu, 1991]

- Capa de entradas: Recibe la información de la fuente externa, y la transmite a la red neuronal para su procesamiento.
- Capa escondida: Recibe la información de la capa de entradas, y procesa toda la información. Todo el paso de procesamiento no se muestra.
- Capa de salida: Recibe la información procesada desde la red y envía el resultado a un receptor externo.

Cuando la capa de entradas recibe información de una fuente externa, ésta se "activa" y emite señales a sus vecinos. Los vecinos reciben la excitación de la capa de entradas y, contrariamente, éstos emiten ouputs a sus vecinos. Lo que resulta es una activación en patrones, que eventualmente se manifiesta en la capa de salida.

Dependiendo de la fuerza de las conexiones, las señales pueden excitar ó inhibir a los nodos, al igual que en una neurona biológica.

La red neuronal mantiene información a través de:

- las magnitudes de las señales pasadas a través de la red, y
- las conexiones de los nodos con los vecinos

Para operar una red neuronal, y lograr el comportamiento que se desea, se debe pasar por las siguientes etapas:

- la etapa de entrenamiento
- la etapa de re-llamado

- la etapa de generalización

En la etapa de entrenamiento, se presenta repetidamente a la red neuronal series de patrones entradas-salidas, se ajustan los pesos de todas las interconexiones entre los nodos, hasta que una entrada especificada llegue a la salida deseada. A través de estas acciones, la red neuronal "aprende" el correcto comportamiento entrada – salida de respuesta.
En el desarrollo de la red neuronal, la etapa de entrenamiento es típicamente la más larga y la que más tiempo requiere.
Luego de la etapa de entrenamiento, se realizan las etapas de re-llamado y generalización. En la etapa de re-llamado, se somete a la red neuronal a un completo arreglo de patrones de entradas vistos en la etapa de entrenamiento, y se hacen ajustes para hacer el sistema más confiable y robusto. Durante la etapa de generalización, se somete la red a nuevos patrones de entradas, y se espera que la red produzca salidas satisfactorias.

III.7. Entrenamiento con Retro-propagación del Errror

El entrenamiento de las redes neuronales se realiza con patrones de conjuntos entrada-salida, los cuales se quiere reproducir. Supongamos que se tiene una red neuronal *feedforward*, con una capa escondida, como la que se muestra en la figura 9. En este caso suponemos que el último elemento de la capa de entrada, y de la capa escondida proporcionan los valores umbrales, y sus pesos se modifican al igual que el resto de la red neuronal. Las entradas las llamaremos **z**, las salidas de la capa escondida **y**, y las salidas **o**. Los pesos de la capa de entrada los denotaremos con la letra **v**, y los de la capa escondida **w**. Como se ha mencionado previamente, los componentes de una matriz, se denotan con el primer subíndice indicando el nodo de llegada, y el segundo indicando el nodo de partida.

El objetivo del entrenamiento es minimizar el error producido entre la salida generada por la red neuronal y el valor deseado. Los patrones *"blanco"* los cuales se tienen para el entrenamiento, se les denominarán con la letra **d**.

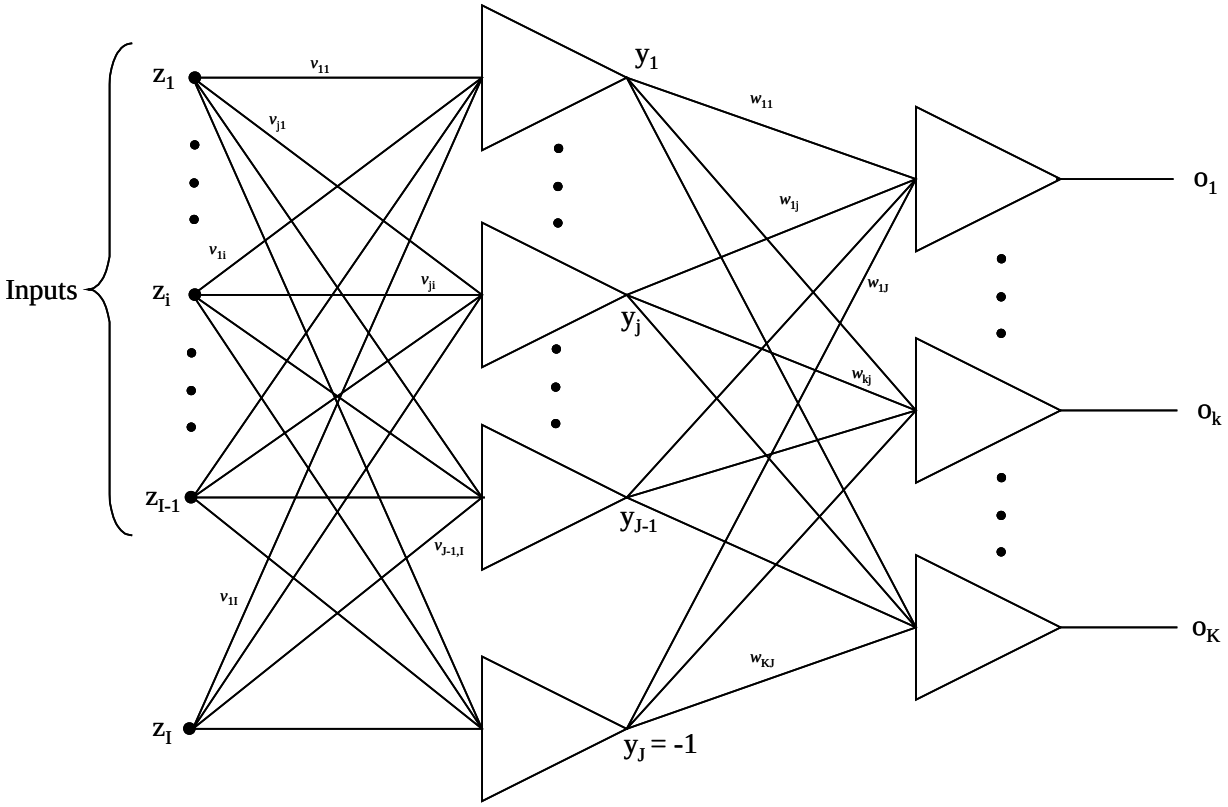


Figura 9. Red Neuronal Feedforward con dos capas. [Zurada, 1992]

Dados *P* pares de entrenamiento:

$$\{z_1, d_1, z_2, d_2, \ldots, z_p, d_p\}$$

donde $\mathbf{z}_i$ es (*I* x 1), $\mathbf{d}_i$ es (*K* x 1), y *i* = 1, 2, . . . , P. Como se mencionó anteriormente, los últimos elementos de cada capa proporcionan los valores umbrales, y se le asignarán un valor fijo de –1. El vector **y** tiene dimensión (*J* x 1) y **o** tiene (*K* x 1).

El primer paso es escoger un valor de η mayor que cero, y de E_{max}, el cual representa el error máximo permitido para el entrenamiento. η es una constante positiva usada en el algoritmo.

Los pesos **W** y **V** son inicializados en valores al azar pequeños; **W** es de dimensión (*K* x *J*), y **V** es de (*J* x *I*).

Se le suministra a la red el primer vector de entrada, y se calcula la salida de cada capa:

$$y_j \leftarrow f(\mathrm{v}_j^t \mathrm{z}), \text{para } \mathrm{j} = 1, 2, ...\mathrm{J} \tag{16}$$

donde v_j, un vector (columna), es la j-ésima fila de **V**, y

$$o_j \leftarrow f(\mathrm{w}_k^t \mathrm{y}), \text{para } \mathrm{k} = 1, 2, ...\mathrm{K} \tag{17}$$

donde w_k, un vector (columna), es la k-ésima fila de **W**.

Luego el error es calculado:

$$E \leftarrow \frac{1}{2}(d_k - o_k)^2 + E, \text{ para } k = 1, 2, ...\mathrm{K} \tag{18}$$

El error es acumulativo, es decir, se suma por cada serie de datos de entrenamiento.

Una vez que se tiene el valor del error, se calculan los vectores de error. El término del error para la capa de salida es:

$$\delta_{\mathrm{o\,k}} = (d_k - o_k)(1 - o_k)o_k, \text{ para } k = 1, 2, ...\mathrm{K} \tag{19}$$

Y el término para la capa escondida es:

$$\delta_{yj} = y_j(1-y_j)\sum_{k=1}^{K}\delta_{\text{ok}} w_{kj},\ \text{para}\ j=1,2,...\text{J} \qquad (20)$$

El vector δ_o es de dimensión (K + 1), y δ_y es de (J + 1).

Luego los pesos de la capa de salida son ajustados:

$$w_{kj} \leftarrow w_{kj} + \eta\delta_{ok} y_j, \text{para}\ j=1,2,...\text{J}\ \text{ y }\ \text{para}\ k=1,2,...\text{K} \qquad (21)$$

y los de la capa escondida:

$$v_{ji} \leftarrow v_{ji} + \eta\delta_{yj} z_i, \text{para}\ j=1,2,...\text{J}\ \text{ y }\ \text{para}\ i=1,2,...\text{I} \qquad (22)$$

Si existen más series de entrenamiento, se vuelve a reiniciar el proceso. Por el contrario, si las series de entrenamiento se han completado, se verifica que el error acumulado E es menor que el error máximo permitido E_{max}; si lo es, la etapa de entrenamiento se ha completado. Si el error acumulado es mayor que el error máximo, se reinicia el proceso de entrenamiento, con valores iniciales de los pesos iguales a los últimos obtenidos en la etapa de entrenamiento previa.últimos obtenidos en la etapa de entrenamiento previa.

III.8. Propiedades, Ventajas y Limitaciones de las Redes Neuronales

Las propiedades de las redes neuronales pueden ser extensas, pero se pueden englobar con un punto de vista externo como [Quantrille y Liu, 1991]:

- La información es distribuida sobre un campo de nodos
- Las redes neuronales tienen la habilidad de aprender
- Las redes neuronales se adaptan bien en el procesamiento de datos ruidosos, incompletos, o inconsistentes
- Las redes neuronales imitan el mecanismo de actividad neuronal biológico básico

Entre las ventajas de las redes neuronales encontramos [Quantrille y Liu, 1991]:

- Comportamiento adaptativo
- Propiedades de reconocimiento de patrones
- Capacidad de filtración: poco sensitiva a información ruidosa o incompleta
- Abstracción automática
- Potencial para uso en línea

Los sistemas donde pueden ser aplicadas con éxito las redes neuronales son incontables, aunque se debe tener en cuenta el compromiso entre las ventajas y las limitaciones de éstas.

Entre las limitaciones de las redes neuronales encontramos [Quantrille y Liu, 1991]:

- Tiempos largos en la etapa de entrenamiento
- Gran cantidad de datos para el entrenamiento
- No garantiza resultados óptimos
- No garantiza 100% de confianza

Con el avance de la computación actual, los tiempos de entrenamiento con sistemas automatizados han ido disminuyendo progresivamente. Tanto hoy en día, como en el futuro, los tiempos de entrenamiento no son ni serán limitantes para la aplicación de las redes neuronales en sistemas ingenieriles. Es importante resaltar que el usuario de las redes neuronales artificiales debe tener presente sus limitaciones y saber cuándo y porqué sus resultados se alejan de los esperados; solo así, las redes neuronales se pueden aplicar satisfactoriamente.

III.9. Modelaje y Simulación

La simulación de procesos en la ingeniería química ayuda a entender mejor el desempeño de los procesos, y es una herramienta poderosa para la optimización de plantas en la etapa de diseño, así como en la etapa operacional [Ingham, et. al, 2000].

Una etapa esencial en el desarrollo de cualquier modelo, es la formulación de las ecuaciones de balances de masa y energía apropiadas. A éstas se les debe añadir las ecuaciones cinéticas apropiadas para las velocidades de reacciones químicas, velocidades de transferencia de masa y energía, así como las ecuaciones que representan los cambios de propiedades del sistema, equilibrio de fase, y las del control aplicado. La combinación de éstas relaciones, provee una base para la descripción cuantitativa del proceso y abarca el modelo matemático básico. El modelo resultante puede variar de un simple caso de relativamente pocas ecuaciones, a modelos de gran complejidad. Mientras mayor sea la complejidad del modelo, mayor será la dificultad en identificar el mayor número de valores de parámetros. Por ello, una de las habilidades del modelaje es derivar el modelo más simple posible, capaz de hacer una representación realista del proceso [Biegler, et al., 1997].

Adicionalmente, uno de los rasgos más importantes del modelaje, es la necesidad de una constante evaluación a la teoría básica (modelo físico), así como las ecuaciones matemáticas que representan el modelo físico (modelo matemático), con el fin de obtener concordancia entre la predicción del modelo y el comportamiento real del proceso (datos experimentales).

En general, las herramientas de simulación de procesos son esenciales para el análisis y evaluación de diagramas de flujo propuestas en la etapa de diseño. Incluso son valiosas en el desarrollo y mantenimiento de sistemas de control integrados.

Las simulaciones son muy usadas en la toma de decisiones en ingeniería. Es una herramienta popular para el modelaje, ya que permite una representación del sistema a ser manipulado, en especial cuando manipular el sistema real es imposible o muy costoso. Las simulaciones permiten cambiar el sistema de coordenadas espaciales ó temporales a uno más conveniente. Además permiten el cambio de la representación del sistema para lograr una mejor comprensión del sistema real; esto requiere que el modelo sea una representación realista del sistema [Ingham, et al., 2000].

Por otra parte, las simulaciones permiten al analista controlar cualquiera de los parámetros del modelo, variables, ó condiciones iniciales, algo que no es posible con el sistema real, también pueden ser usadas para evaluar la respuesta del sistema a condiciones que pudieran no haber ocurrido en el pasado.

III.10. Fundamentos de la Programación

La programación moderna rompe la manera tradicional de escribir código, y cambia la manera de enfocar los problemas presentados en el proceso de codificación. Para obtener una valoración de éste tipo de programación, se debe dejar atrás viejos hábitos e intuiciones acerca de escribir código, y repasar la evolución de la programación enfocada a objetos (OOP por sus siglas en inglés). [Pattison, 1999]
Se presentará a continuación una pequeña introducción a éste tipo nuevo de programación, el cual fue aplicado en el presente trabajo de grado.

III.10.1. Objetos, métodos y eventos, clases [Microsoft Visual Basic 6.0 Programmer´s Guide]

Un objeto es un término general usado para describir un sistema, el cual realiza acciones. En Visual Basic®, un objeto es una combinación de código y data que puede ser tratado y controlado como una unidad.
Una hoja de Microsoft Excel® es un ejemplo de un objeto conteniendo otros objetos como hojas de trabajo, tablas y celdas.
Para controlar un objeto, se usa sus propiedades y métodos. Las propiedades son la data, settings y atributos de un objeto. Métodos son los procedimientos que operan en el objeto, ó los que el objeto realiza en data.
Los objetos son encapsulados, esto significa que contienen su código y su data, haciéndolos más fáciles de mantener que las formas tradicionales de escribir código.

Los objetos en Visual Basic® son creados a partir de clases, por lo tanto se dice que un objeto es una *instancia de una clase*.

Para usar un objeto, se debe mantener una referencia hacia él en una variable de objeto. El tipo de lazo determina la velocidad en que los métodos del objeto son accesados usando la variable del objeto.

Por otro lado, una interfase es un conjunto de propiedades y métodos. La interfase por defecto de un objeto de Visual Basic® es una interfase dual la cual soporta todas las formas de lazos.

Además de las interfaces por defecto, los objetos en Visual Basic® pueden implementar interfases extras para proporcionar polimorfismo. El polimorfismo permite manipular diferentes clases de objetos sin importar de qué tipo es cada uno.

III.10.2. Envolvente de una clase [Rogerson, 1995]

Un envolvente es una clase que contiene un objeto al cual la clase proporciona una interfase. Un envolvente puede "envolver" funciones en C, funciones exportadas de una librería de enlace dinámico (dll), clases de C++, una interfase OLE (Object Linking and Embedding), un control OLE, un objeto COM (Component Object Model), ó cualquier otro grupo de funciones (ver figura 10).

Un beneficio importante de las clases es que permiten re-usar código. Una vez que la clase es escrita, se puede usar en diferentes sitios dentro de la aplicación. Por lo tanto, las clases permiten reducir o eliminar código redundante dentro de una aplicación. También facilitan el mantenimiento del código. Se puede modificar o eliminar propiedades o métodos que no son visibles públicamente. Se puede incluso cambiar implementaciones públicas de métodos siempre y cuando la sintaxis para invocar el método no sea alterada. Cuando la implementación de un método en una clase es mejorada, cualquier cliente que use esa clase va ser beneficiario del cambio.

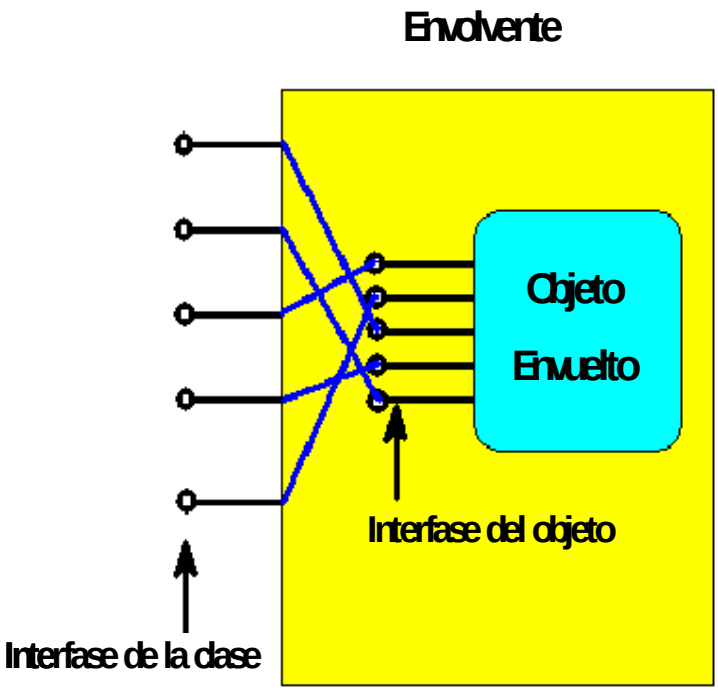


Figura 10. Un envolvente [Rogerson, 1995]

III.10.3. DLLs [Sarma, 1996]

A continuación se presentará la definición de dll, sus ventajas y desventajas, y el porqué de su creación.

III.10.3.1. Descripción

Dll es acrónimo de dynamic-link library (librería de enlace dinámico). Un dll es un componente de software al cual una aplicación se enlaza cuando ésta está siendo ejecutada.

Un programa de Windows es un archivo ejecutable que generalmente crea una o varias ventanas y usa un loop de mensaje para recibir información del usuario. Las librerías de enlace dinámico por lo general no pueden ser ejecutadas

directamente, y no reciben mensajes. Los dll son archivos separados que contienen funciones que pueden ser llamadas por programas o otros dlls para realizar ciertos cómputos.
Como indica el nombre, los dll son enlazados dinámicamente a una aplicación. Para entender esto, se explicará a continuación la tradicional manera de enlace (el enlace estático), para luego explicar brevemente el enlace dinámico.

III.10.3.2. Enlace Estático

En lenguajes de programación de alto nivel como C, Fortran o Pascal, el código de una aplicación es compilado y enlazado a varias librerías para crear un archivo ejecutable. Éstas librerías contienen archivos de objetos de funciones pre-compiladas las cuales son llamadas para ejecutar ciertas tareas, como calcular la raíz cuadrada de un número o alocar memoria. Cuando éstas librerías de funciones son enlazadas a una aplicación, se convierten en parte permanente del archivo ejecutable de la aplicación. Todas las llamadas a las librerías de funciones son hechas en tiempo de enlace; por ello el nombre enlace estático.

III.10.3.3. Enlace Dinámico

El enlace dinámico provee un mecanismo para enlazar las aplicaciones a las librerías en tiempo de corrida. Las librerías residen en sus propios archivos ejecutables y no son copiados a los archivos ejecutables de la aplicación como en el enlace estático. Éstas librerías son llamadas *librerías de enlace dinámico* (dlls) para enfatizar que son enlazadas a la aplicación cuando ésta es llamada y ejecutada, en vez de cuando se es enlazada. Cuando una aplicación usa un dll, el sistema operativo carga el dll a memoria, resuelve las referencias a funciones en el dll para que puedan ser invocadas por la aplicación, y descarga el dll cuando no se necesita más. Éste mecanismo de enlace dinámico puede ser realizado explícitamente por las aplicaciones o implícitamente por el sistema operativo.

Como se mencionó anteriormente, los dll no son programas ejecutables. Para entender esto, a continuación se presentan ciertas diferencias entre las aplicaciones de Windows® y los dll.

III.10.3.4. Diferencias entre aplicaciones y DLLs

A pesar de que los DLLs y las aplicaciones son ambos módulos de programas ejecutables, éstos difieren de varias maneras. Para el usuario final, la diferencia más obvia es que los DLLs no son programas que son ejecutados directamente del Program Manager ó otros programas cobertores. Desde el punto de vista del sistema, existen dos diferencias fundamentales entre aplicaciones y DLLs:

- Una aplicación puede tener varias instancias de sí mismo corriendo en el sistema simultáneamente; en cambio, un DLL sólo puede tener una. Cada instancia de la aplicación tiene su propio espacio automático de datos, pero todas las instancias de la aplicación comparten una sola copia del código ejecutable y recursos. Por otra parte, no importa cuántas veces un DLL es cargado, éste tiene exactamente una sola instancia. Para un sistema operativo de 32 bits, cada carga de un DLL tendrá su propia instancia; por lo tanto, compartir segmentos de datos entre múltiples procesos no es recomendado para DLLs de 32 bits.
- Una aplicación puede “poseer” cosas, en cambio un DLL no. Sólo los procesos son capaces de “propiedad”, y sólo las instancias de aplicaciones son procesos. En un sistema operativo de 32 bits, los DLL se adhieren a los procesos, sólo los procesos individuales pueden poseer memoria. Los DLLs no son más entidades por ellos mismos.

En otras palabras, los DLLs son módulos de programas separados de las aplicaciones. En disco, residen en sus propios archivos ejecutables, los cuales podrían contener código, datos, y recursos (datos de sólo lectura) como bitmaps y cursores. Cuando un proceso carga un DLL, el sistema mapea el código y datos del DLL en el espacio para direcciones del proceso. En un sistema

operativo de 32 bits, los DLLs no pasan a ser parte del sistema operativo, más bien pasan a ser parte del proceso que carga el DLL. Cualquier llamada para fijar memoria hechas por funciones en el DLL hacen que la memoria sea fijada en el espacio de direcciones del proceso; ningún otro proceso tiene acceso a esta memoria fijada. Las variables globales y estáticas fijadas por un DLL tampoco son compartidas entre múltiples mapeos del DLL.

III.10.3.5. Entendiendo cómo el DLL y la aplicación son localizados en la memoria [D'Souza, et al., 1999]

Uno de los cambios más importantes a los DLL para Win32® es la localización en memoria donde el código y datos del DLL reside. En Win32®, cada aplicación corre entre el contexto de su propio espacio de dirección linear 32-bit. De esta manera, cada aplicación tiene su espacio con dirección privada el cual sólo puede ser accesado desde código entre éste proceso. (En Win32®, cada espacio con dirección privada de cada aplicación es referido como el *proceso* para ésa aplicación). Todo el código, datos, recursos y memoria dinámica de una aplicación también reside entre el proceso de la aplicación. Más aun, no es posible para una aplicación referir datos o código que resida fuera de su propio proceso. Debido a esto, cuando un dll es cargado debe de alguna manera residir en el proceso de la aplicación que cargó al DLL; si más de una aplicación carga el mismo DLL, éste debe residir en el proceso de cada aplicación.
Entonces, para satisfacer lo anteriormente dicho – que sea reentrante (accesible de más de una vía a la vez) y que tenga sólo una copia del DLL físicamente cargada en memoria – Win32® usa mapeo de memoria. A través del mapeo de memoria, Win32® es capaz de cargar el DLL una vez al grupo global y luego mapear el rango de la dirección del DLL al espacio de direcciones de cada aplicación que lo carga.

III.10.3.6. Desventajas de los DLLs [Sarma, 1996]

La idea de Microsoft® de re-usar código mediante la creación de los dll, no ha venido sin problemas asociados. A continuación se presentan los dos tipos más comunes de conflictos con los dll.

III.10.3.6.1. DLL Hell

Los sistemas operativos y aplicaciones modernas se construyen de varios componentes. Un componente es una entidad de software auto-contenida, la cual ofrece un grupo de funciones que pueden ser usadas extensamente por una variedad de aplicaciones. Por otra parte, debido a que los componentes individuales son usados por más de una aplicación, es esencial compartir los componentes.

Windows ha adoptado el concepto de compartir código desde su concepción. Todos los sistemas operativos equilibran la necesidad de proveer un abanico robusto y completo de servicios con las limitaciones de recursos del hardware para el cual el sistema operativo fue diseñado. Hasta hace poco, el uso del CPU y el espacio en disco fueron recursos bastante limitados en la plataforma PC. Una manera obvia de acomodar el sistema operativo y el código de las aplicaciones en un pequeño espacio ha sido el compartir código lo más posible. Entre muchos otros beneficios, el compartir código mejora la limitación de los recursos del hardware y minimiza el frente expuesto que se debe comprobar para asegurar la calidad.

Para tener éxito en compartir componentes se requiere que todo componente compartido funcione exactamente como las versiones previas de ese componente. En la práctica, 100 por ciento de compatibilidad es difícil si no imposible de obtener, debido a la dificultad de comprobar todas las configuraciones en las cuales el componente a ser compartido podría ser usado. Aplicaciones viejas y más nuevas terminan usando el mismo componente, y con

el tiempo, reparar y mejorar los componentes pasa a ser una tarea de gran dificultad.

También, la funcionalidad práctica de un componente no es fácilmente definida. Las aplicaciones pueden convertirse en dependientes de efectos colaterales no intencionados los cuales no son parte de la función esencial del componente. Por ejemplo, una aplicación puede pasar a ser dependiente de una falla en el componente, y cuando el creador del componente decide reparar el fallo, la aplicación deja de funcionar, lo cual ha pasado a llamarse "DLL hell". Mientras más aplicaciones usen ese componente, más se agudiza el problema.

Esta falta de compatibilidad puede resultar en la incapacidad de desplegar una nueva aplicación sin afectar otra ya desplegada, o comprometer la funcionalidad de la nueva aplicación. Cualquier aplicación nueva requiere una versión del componente diferente a la que esta desplegada originalmente.

Compartir componentes conlleva a que las aplicaciones se conviertan en interdependientes entre ellas, introduciendo un elemento de fragilidad. Hacer cambios en un componente podría producir efectos no intencionados en otros componentes.

Típicamente una aplicación podría pasar a ser dependiente de una versión particular de un componente compartido; otra aplicación podría ser instalada con una versión más nueva (ó mas vieja) del componente compartido, resultando en que la primera aplicación sufra debido al cambio. En casos extremos, aplicaciones que alguna vez trabajaron bien, empiezan misteriosamente a funcionar en forma inapropiada, ó incluso a fallar. Esta condición es frecuentemente llamada "DLL hell".

III.10.3.6.2. Conflicto de dirección base del DLL

Todos los DLLs del sistema son usualmente cargados a la misma dirección virtual sin importar el proceso. A veces podría haber un conflicto en la dirección base del DLL y se podría presentar el siguiente mensaje cuando el DLL es cargado por la aplicación que está siendo usada:

Dll xxxx.dll base 10000000 relocated due to collision with
yyyy.dll

Esta re-localización causa una penalidad en la aplicación, pero ya que es de una sola ocurrencia, la penalidad puede que no sea de importancia. Cada vez que un DLL es re-localizado, los fix-ups del mismo tienen que ser re-calculados. Esto lo realiza el sistema operativo y no puede ser controlado por aplicaciones que usen éste DLL.

IV. Metodología

Para la realización del presente trabajo de grado se siguió la siguiente metodología:

IV.1. Investigación y creación de las diversas interfases del envolvente

Con el objetivo de poder conectar a *SimWrap* con los diversos programas usados (como ACM y NN Tool 2000), fue necesario conocer las interfases que éstos ofrecen en el ambiente Windows, y saber lo que era posible lograr con éstas. Antes de poder hacer la conexión de los programas, se necesitó investigar dichas interfases, y conocer el tipo de codificación que éstas requerían.

Una vez conocidas las interfases, se procedió a codificar los comandos que permitieron el uso de las mismas a lo largo de los algoritmos.

IV.2. Desarrollo de los diversos algoritmos de generación de datos

Una vez creadas las interfases, se procedió a desarrollar los diversos algoritmos de generación de data, que permitieron crear las colecciones de datos (compuestas de series entradas – salidas), que luego fueron usadas en el entrenamiento de las redes neuronales.

IV.3. Desarrollo de un algoritmo de pronósticos

Con el objetivo de evaluar el desempeño de las redes neuronales creadas, se desarrolló un algoritmo que generaba valores para las entradas de las redes neuronales, y obtenía automáticamente los valores pronosticados por las mismas.

IV.4. Desarrollo de la interfaz de usuario

Posteriormente se desarrolló la interfaz usuario – máquina en Visual Basic, con el fin de permitir una amigable interacción entre el usuario y el algoritmo, de forma tal que se pudiera introducir fácilmente los datos requeridos por éste último para su corrida.

IV.5. Uso de los modelos para evaluar el desempeño de los diversos algoritmos

Se usaron tres modelos básicos de la ingeniería química para validar el programa, así como para evaluar el desempeño de los algoritmos de selección de datos y de las redes neuronales artificiales.

V. Resultados

A continuación se presentan los resultados obtenidos en el presente trabajo, los cuales fueron agrupados de la siguiente manera: resultados en el software y resultados en los modelos.

V.1. Resultados en el software

El programa desarrollado en el presente trabajo fue denominado *SimWrap*, y el mismo está basado en la creación de datos que eventualmente son usados para entrenar una serie de redes neuronales.

Debido a que las simulaciones se basan en el envío de ciertos valores de entradas para obtener los pertinentes valores de las salidas para esa precisa serie de valores de entradas, las colecciones de datos se estructuran en series de valores entradas – salidas (ver tabla 1).

Tabla 1. estructura de una Colección de datos

	entrada 1	entrada 2	...	entrada n	salida 1	saliida 2	...	salida n
serie 1	valor	valor	...	valor	valor	valor	...	valor
serie 2	valor	valor	...	valor	valor	valor	...	valor
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
serie n	valor	valor	...	valor	valor	valor	...	valor

Dichas estructuras de colecciones de datos dependen del número de entradas y salidas que el usuario halla especificado para la simulación.

Estos conceptos y otros aspectos de la estructura y funcionamiento de *SimWrap* serán clarificados en las secciones siguientes.

V.1.1. SimWrap

El programa desarrollado denominado *SimWrap*, conecta a Aspen Custom Modeler®, envía datos extraídos a Excel para su visualización, y selecciona datos representativos de una superficie. Luego, conectándose al software modelador de redes neuronales: Neural Network Tool 2000®, crea unas redes neuronales, las cuales son entrenadas con los datos previamente seleccionados.

V.1.1.1. Componentes de SimWrap

SimWrap es un programa que genera datos de una simulación. El simulador comercial usado es el Aspen Custom Modeler® (ACM). El modelo usado debe estar previamente escrito y compilado en éste.
SimWrap crea valores para las entradas de la simulación en ACM, para luego enviárselos. Si la simulación cumple los requisitos debidos para su posible corrida (por ejemplo, que tenga cero grados de libertad), entonces *SimWrap* realiza la corrida con el propósito de extraer los valores de las salidas para ese preciso conjunto de valores de entradas. De ésta manera *SimWrap* genera los datos que serán usados para el proceso de entrenamiento de la red neuronal. Las colecciones de datos que genera el sistema, son enviadas a Excel con el objeto que el usuario pueda visualizar y luego darle uso a los datos que el programa está manejando.
En general, *SimWrap* tiene los siguientes componentes:

1. Generación de datos: Es donde la primera colección de entradas-salidas es generada. Ésta primera colección es llamada "*colección original*", y es la que eventualmente se refina. Existen dos maneras posibles de generar los datos de la colección original: Usando una rejilla al azar, ó una rejilla fijada.

2. Refinamiento: Es de uso opcional, y funciona sólo con la generación de datos por rejilla al azar. Es donde *SimWrap* selecciona puntos de la

colección original, los cuales tienen gradiente mayor a un valor crítico introducido por el usuario. El valor crítico es llamado también valor umbral. Luego el algoritmo genera nuevos puntos al azar en la proximidad de éstos puntos de "alto gradiente". Éstos nuevos puntos son generados en el espacio de entradas, esto es, cada punto es la combinación de los valores para todas las entradas de la simulación. Luego *SimWrap* envía cada una de éstas entradas al simulador, con el propósito de generar los valores de las salidas para cada "punto". La colección refinada tiene las series de la colección original, más las nuevas series generadas por el algoritmo de refinación. El objetivo de refinar las zonas de alto gradiente, es obtener un mejor resultado con las redes neuronales cuando el pronóstico se realiza en éstas zonas.

3. Creación de la Red Neuronal: Es de uso opcional, y es donde *SimWrap* envía los datos a NN Tool 2000® para la creación de la red neuronal. Son enviadas dos colecciones de datos: la colección refinada, y la colección no refinada.

4. Pronóstico: Es de uso opcional, y es donde *SimWrap* genera valores para las entradas (de la misma manera que son generados en el punto 1) y los envía a la red neuronal de manera de obtener el valor de la salida pronosticada por la red neuronal. Luego el programa envía éstas series de entradas a ACM con el objeto de obtener los valores "reales" de la salida, y luego calcula la suma del cuadrado del error entre la salida pronosticada por la red neuronal y el valor real (valores obtenidos mediante el uso del simulador). En ésta etapa *SimWrap* evalúa el desempeño del algoritmo de refinación, haciendo una comparación entre las redes neuronales entrenadas con la colección refinada y la no refinada.

V.1.1.2. Descripción detallada de cada componente

A continuación se realiza una descripción detallada de cada uno de los componentes de *SimWrap*.

V.1.1.2.1. Generación de datos

SimWrap usa las propiedades y métodos de *Aspen Custom Modeler*® (ACM), con el objeto de obtener los valores de las salidas para cada serie de entradas, generadas en la creación de la colección original. *SimWrap* primero genera el valor de cada entrada, y luego envía la serie a ACM. Si la simulación tiene cero grados de libertad, *SimWrap* la corre, y recoge el valor de cada salida para luego guardarlos junto con su serie de entradas.

SimWrap tiene que ser inicializado desde una hoja de Excel® llamada "*initialization sheet*", la cual contiene el nombre de las entradas, salidas y constantes (si existe alguna). También debe contener el valor de cada constante (si existe alguna), y los valores del límite superior e inferior para cada entrada; éstos últimos valores indican el rango en el cual cada entrada varía.

El usuario selecciona el tipo de algoritmo de generación de datos, los cuales pueden ser: algoritmo de rejilla al azar, y algoritmo de rejilla fijada. Los algoritmos generan valores en línea recta desde el límite inferior hasta el límite superior de cada entrada. Entre los valores introducidos en la inicialización del algoritmo, se encuentra el número de cálculos, el cual representa el número total de series de datos (entradas – salidas) generados. Para cada serie, debe ser hecha una corrida del simulador. Es importante notar aquí, que los datos generados en ésta etapa crean la *colección original de datos*: "original", ya que *SimWrap* puede cambiarla a lo largo de la corrida del algoritmo (refinarla, por ejemplo). La colección original de datos contiene los datos de todas las entradas, y todas las salidas generadas por uno de los dos métodos mencionados anteriormente (usando una rejilla al azar ó una rejilla fija).

V.1.1.2.2. Algoritmo de rejilla fija

Para la rejilla fija, el número de cálculos que se selecciona, debe ser igual ó mayor que dos (2), ya que los cálculos comienzan en el límite inferior, y terminan en el superior (un cálculo en cada límite, y si el número de cálculos es mayor que dos, las que restan van a ser hechas entre éstas). Los valores de las entradas son cambiados en un salto incremental (fijado por rango y el número de cálculos) en cada cálculo. Cuando el rango de la primera variable es completado (desde el límite inferior, hasta el superior), la siguiente entrada es cambiada en un solo salto incremental, y luego la primera variable vuelve a cambiar desde el límite inferior, hasta el superior. Por lo tanto, el número total de cálculos realizados por el algoritmo viene dado por la ecuación 23

$$N_c = \left(Nc_{\,por\ cada\ entrada}\right)^{Numero\ de\ entradas} \qquad (23)$$

Donde N_c es el número de cálculos.

Éste número de cálculos es llamado "*número de cálculos exponencial*" dentro del programa, ya que incrementa en razón exponencial; éstos van a ser el número de series en la colección original.

Por ejemplo, para una simulación que contenga 3 entradas, se podrían establecer los límites señalados en la tabla 2.

Tabla 2. Límites de las entradas

	Límite inferior	Límite superior
Entrada 1	0	1
Entrada 2	1	2
Entrada 3	2	3

Donde los valores generados para éste caso serían los reportados en la tabla 3.

Tabla 3. Valores de las entradas de la rejilla fija

Entrada 1	Entrada 2	Entrada 3
0	1	2
0.5	1	2
1	1	2
0	1.5	2
0.5	1.5	2
1	1.5	2
0	2	2
0.5	2	2
1	2	2
0	1	2.5
0.5	1	2.5
1	1	2.5
⋮	⋮	⋮
1	2	3

La rejilla en el espacio (si consideramos cada entrada como una coordenada) sería algo semejante a la mostrada en la figura 11.

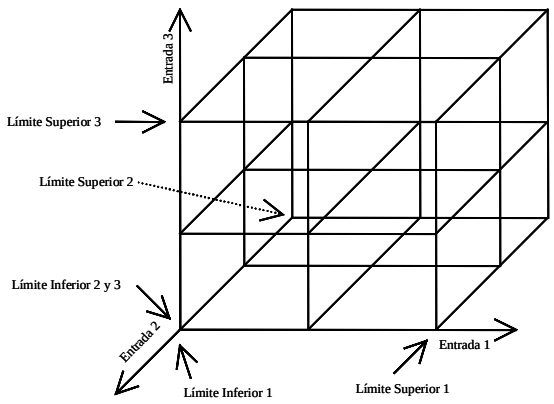


Figura 11. Rejilla fija de valores

La ventaja de éste algoritmo es que “barre” todo el rango de cada entrada, creando una colección de datos normalmente distribuida. Mientras más cálculos se hagan, más refinada va a ser la rejilla, mejorando la colección para el entrenamiento. Los puntos son generados sin importar el tipo de superficie de la simulación.

V.1.1.2.3. Algoritmo de rejilla al azar

Para cada cálculo, el algoritmo genera puntos al azar entre el límite inferior y superior para cada entrada. De ésta forma se genera una selección de puntos totalmente al azar entre los límites lineales de la rejilla. Los límites son los mismos que los del algoritmo de rejilla fija, pero los puntos generados no.
Para éste tipo de algoritmo, está disponible otro tipo de *número de cálculos*. El tipo *normal* de *número de cálculos*, representa realmente el número indicado en el parámetro; eso quiere decir que el parámetro no es exponencial. Por ejemplo, si el usuario selecciona 50 cálculos, entonces 50 cálculos van a ser ejecutados, y éste va a ser el número de series en la colección original.
El usuario puede seleccionar en éste algoritmo qué tipo de *número de cálculos* va a usar, si el *número de cálculos exponencial*, ó el *número de cálculos normales*. Esto no es posible en el algoritmo de rejilla fija, el cual siempre necesita de cálculos exponenciales.

V.1.1.2.4. Algoritmo de Refinación

El propósito del algoritmo es generar puntos al azar en la región de la superficie n-dimensional de entradas, donde el gradiente de la superficie generada por todas las entradas y una sola salida es mayor que un valor crítico. La llamamos superficie, ya que podría no existir una sola función relacionando las entradas con las salidas (esto depende del tipo de simulación); en cambio, se puede tener toda una simulación (series de ecuaciones) relacionándolas, y ésta simulación, genera puntos en la

dimensión [número de entradas + número de salidas], los cuales pueden ser conectados por una superficie. El algoritmo calcula una aproximación del gradiente para una superficie generada por todas las entradas y una sola salida.

El usuario puede seleccionar el número de puntos (nodos) de la colección original, para los cuales el gradiente va a ser calculado (los nodos son la combinación de valores para cada input; suponga una coordenada para cada entrada). Esto tiene significado cuando la colección original tiene un tamaño significativo. Los puntos seleccionados para el cálculo del gradiente son seleccionados al azar de la colección original (usando el algoritmo de selección). El número de puntos para el cálculo del gradiente indica cuán bien la colección original es cubierta para la búsqueda de los altos gradientes. El valor representa un compromiso entre la rapidez del análisis y el detalle de la búsqueda.

Una vez que el algoritmo tenga los puntos en los cuales el gradiente va a ser calculado, éste "conecta" cada uno de éstos puntos con el resto de ellos (figura 12). El término "conectar", indica que el algoritmo calcula la distancia entre los puntos conectados.

Con el objeto de calcular la aproximación del gradiente para un punto en la superficie, el algoritmo necesita algunos vecinos en la proximidad de éste. El usuario selecciona el número de éstos vecinos, el cual tiene que ser mayor que el número de entradas en la simulación. Usualmente, a medida que aumenta el número de vecinos, la aproximación del gradiente va a aproximarse más al valor real. Éstos vecinos, son los puntos más cercanos (en término de distancia) en la proximidad del punto al cual el gradiente está siendo calculado.

Una vez que el algoritmo haya seleccionado los puntos donde el gradiente es mayor que un valor crítico, este genera puntos al azar (valores para cada entrada) entre éste punto y sus vecinos, y luego envía a estos puntos al simulador y corre la simulación con el objeto de obtener el valor de la salida de la cual el gradiente está siendo calculado. La rejilla refinada contendrá éstos valores, más los valores de la colección original de datos. Éstos pasos son

repetidos una vez para cada salida. El usuario introduce el número de nuevos puntos a generar entre los vecinos.

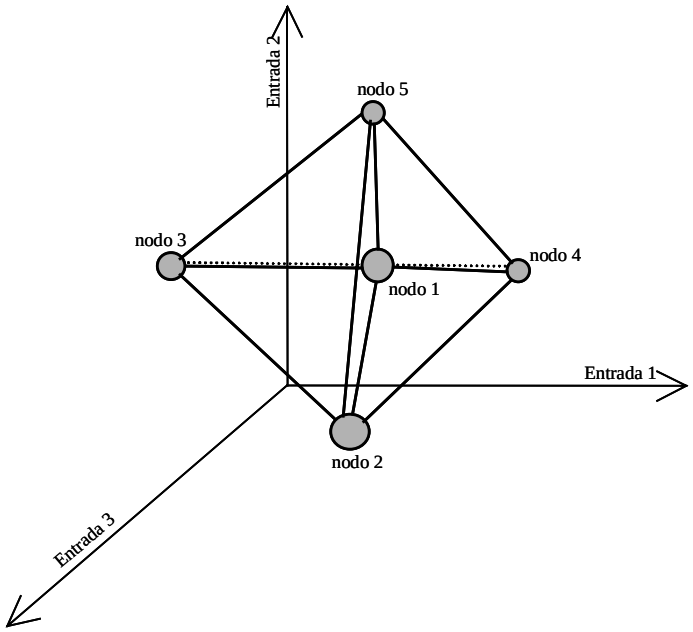


Figura 12. Conexión de nodos

V.1.1.2.4.1. Gradiente

El gradiente es calculado para la superficie generada por todos las entradas, y una sola salida. Esto representa que primero el algoritmo separa colecciones conteniendo todas las entradas y una sola salida, por lo cual el refinamiento va a ser llevado a cabo una vez por cada colección de éstas que se tenga. El algoritmo puede encontrar una región de alto gradiente para la primera salida, y ésta región puede que sea diferente a la región de alto gradiente para la segunda salida. Cada salida es tratada diferente.

Suponga que se tiene una simulación con 3 entradas, y 4 vecinos más próximos para calcular la aproximación del gradiente (figura 12). Si se está calculando el gradiente para el nodo 1, y los nombres de las entradas son: x, y, z; la salida va a ser F, se tiene el sistema:

$$
\begin{aligned}
dF_{1-2} &= \frac{\partial F}{\partial x}.dx_{1-2} + \frac{\partial F}{\partial y}.dy_{1-2} + \frac{\partial F}{\partial z}.dz_{1-2} \\
dF_{1-3} &= \frac{\partial F}{\partial x}.dx_{1-3} + \frac{\partial F}{\partial y}.dy_{1-3} + \frac{\partial F}{\partial z}.dz_{1-3} \\
dF_{1-4} &= \frac{\partial F}{\partial x}.dx_{1-4} + \frac{\partial F}{\partial y}.dy_{1-4} + \frac{\partial F}{\partial z}.dz_{1-4} \\
dF_{1-5} &= \frac{\partial F}{\partial x}.dx_{1-5} + \frac{\partial F}{\partial y}.dy_{1-5} + \frac{\partial F}{\partial z}.dz_{1-5}
\end{aligned}
\qquad (24)
$$

donde:

dF_{1-n} : Es la diferencia en la salida entre el nodo 1 y el vecino " n" (n = 2, 3, 4, 5)

$dx_{1-n}, dy_{1-n}, dz_{1-n}$: son las diferencias en la entrada entre el nodo 1 y el vecino " n"

$\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}$: son las derivadas parciales de la función respecto a las entradas.

Otra forma de expresar el sistema anterior es:

$$
\begin{pmatrix} dF_{1-2} \\ dF_{1-3} \\ dF_{1-4} \\ dF_{1-5} \end{pmatrix} = \begin{pmatrix} dx_{1-2} & dy_{1-2} & dz_{1-2} \\ dx_{1-3} & dy_{1-3} & dz_{1-3} \\ dx_{1-4} & dy_{1-4} & dz_{1-4} \\ dx_{1-5} & dy_{1-5} & dz_{1-5} \end{pmatrix} . \begin{pmatrix} \frac{\partial F}{\partial x} \\ \frac{\partial F}{\partial y} \\ \frac{\partial F}{\partial z} \end{pmatrix} \qquad (25)
$$

donde el número de ecuaciones es fijada por el número de vecinos, y el número de incógnitas es fijada por el número de entradas. Llamemos al número de ecuaciones (el número de vecinos) " n" , y al número de incógnitas (el número de entradas) " m" .

La ecuación 25, puede ser escrita en forma matricial como:

$$b = A \cdot x \tag{26}$$

Aquí el punto denota multiplicación de matrices, **b** el vector a la izquierda del símbolo igual (vector de términos independientes), **A** la primera matriz a la derecha del signo igual (matriz de coeficientes), y **x** el segundo vector a la derecha del signo de igualación (vector solución, es decir, vector gradiente). Si n = m entonces existen igual número de ecuaciones que incógnitas, por lo cual hay un gran chance de resolver el sistema para una única solución de series de $\frac{\partial F}{\partial Input}$. Sin embargo, analíticamente, se puede fallar en buscar una sola solución si una de las n ecuaciones es una combinación lineal de las otras, una condición llamada *degeneración de filas*, ó si todas las ecuaciones contienen alguna variable en exactamente la misma combinación lineal, condición llamada *degeneración de columnas* [Press, et al., 1999]. No obstante, éstos sistemas son poco posibles que ocurran ya que el algoritmo genera los datos al azar.

Si n < m, ó si n = m pero las ecuaciones están degeneradas, entonces existen efectivamente menos ecuaciones que incógnitas. En éste caso, puede no haber solución, ó más de una solución del vector **x** [Press, et al., 1999]. Éstos sistemas son evitados forzando al usuario a seleccionar más vecinos que entradas tenga el sistema.

En el caso opuesto, existen más ecuaciones que incógnitas n > m. Cuando esto ocurre, no hay generalmente una solución para el vector **x**, y el sistema de ecuaciones se dice que está sobre-determinado. Aunque, se podría buscar la mejor solución “comprometida”, es decir, la que mejor se acerque a satisfacer a todas las ecuaciones simultáneamente. Si el acercamiento es definido en términos de mínimos cuadrados, i.e., que la suma de los cuadrados de las diferencias entre el lado derecho e izquierdo de la ecuación 25 sea minimizado, entonces el problema lineal sobre-determinado se reduce (usualmente) a un problema lineal que se puede resolver, y es llamado problema lineal de mínimos cuadrados [Press, et al., 1999].

El sistema de ecuaciones reducido a resolver puede ser escrito como un sistema de m x m ecuaciones

$$\left(A^T.A\right).x = \left(A^T.b\right) \qquad (27)$$

donde A^T denota la matriz A transpuesta. Las ecuaciones involucradas en la expresión 27 son llamadas las ecuaciones normales del problema lineal de mínimos cuadrados.

El sistema de ecuaciones resultante de la expresión 27 es resuelto, en el algoritmo, usando una rutina Gauss Jordan.

V.1.1.2.4.2. Algoritmo de Selección

Cuando la colección original es demasiado grande, ó cuando el tiempo para resolver tiene que ser corto, el usuario puede elegir usar un algoritmo de selección el cual selecciona series al azar de la colección original para los cuales el gradiente va a ser calculado. El número introducido por el usuario, el cual representa el número de series a elegir, controla el detalle de la búsqueda en la colección original por las regiones de alto gradiente.

El algoritmo selecciona series al azar de la colección original. Cada una de éstas series proporciona un juego completo de valores entradas – salidas. Todos los puntos seleccionados son diferentes de los otros (el algoritmo no repite puntos previamente seleccionados). Por ejemplo, si se tiene una colección original con 100 elementos, y el usuario determina elegir sólo 5 de ésta, por lo que el algoritmo selecciona al azar las series 2, 4, 5, 57 y 98, tal como se muestra en la tabla 4, donde las flechas indican las series seleccionadas. Puede verse aquí que las series son filas en la colección original de datos y las series seleccionadas son nodos para los cuales el gradiente va a ser calculado.

Tabla 4. Series seleccionadas en la colección original

	Entrada 1	Entrada 2	...	Entrada n	Salida 1	Salida 2	...	Salida n
Serie 1	Valor	Valor	...	Valor	Valor	Valor	...	Valor
→ Serie 2	Valor	Valor	...	Valor	Valor	Valor	...	Valor
Serie 3	Valor	Valor	...	Valor	Valor	Valor	...	Valor
→ Serie 4	Valor	Valor	...	Valor	Valor	Valor	...	Valor
→ Serie 5	Valor	Valor	...	Valor	Valor	Valor	...	Valor
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
→ Serie 57	Valor	Valor	...	Valor	Valor	Valor	...	Valor
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
→ Serie 98	Valor	Valor	...	Valor	Valor	Valor	...	Valor
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Serie 100	Valor	Valor	...	Valor	Valor	Valor	...	Valor

V.1.1.2.4.3. Conexión de Nodos

Los nodos son conectados en el espacio n-dimensional de entradas. Por ejemplo, en la figura 12, existen 5 nodos, en un espacio de tres dimensiones (3 entradas), que originan las distancias:

- Para el nodo 1: 1-2, 1-3, 1-4, 1-5.
- Para el nodo 2: 2-1 (igual a 1-2), 2-3, 2-4, 2-5.
- Para el nodo 3: 3-1 (igual a 1-3), 3-2 (igual a 2-3), 3-4, 3-5.
- Para el nodo 4: 4-1 (igual a 1-4), 4-2 (igual a 2-4), 4-3 (igual a 3-4), 4-5.
- Para el nodo 5: 5-1 (igual a 1-5), 5-2 (igual a 2-5), 5-3 (igual a 3-5), 5-4 (igual a 4-5).

Por otra parte, las coordenadas de los nodos son colecciones de valores para cada entrada, ejemplo para una simulación con 3 entradas:

- Nodo 1:[valor entrada 1, valor entrada 2, valor entrada 3]
- Nodo 2: [valor entrada 1, valor entrada 2, valor entrada 3]

y así sucesivamente...

Cada nodo es conectado con *todos* los otros nodos, y todas las distancias entre los otros cuentan en la selección de los vecinos más cercanos. Por ejemplo, los vecinos más cercanos para el nodo 1 en orden decreciente son: 2, 4, 5, 3; para el nodo 2: 1, 3, 4, 5; para el nodo 3: 2, 1, 4, 5; etc. La matriz de conexiones entre los nodos para un sistema de 6 nodos sería algo como se muestra en la tabla 5:

Tabla 5. Matriz de distancias

Nodos	Nodo1	Nodo2	Nodo3	Nodo4	Nodo5	Nodo6
Nodo1						
Nodo2						
Nodo3						
Nodo4						
Nodo5						
Nodo6						

Las celdas grises representan que existe distancia entre los dos nodos especificados, y los espacios en blanco indican que no hay conexión entre un nodo y sí mismo. Se puede notar que las distancias sobre la diagonal son las mismas que las de abajo de la diagonal (simetría con la diagonal como eje).

El algoritmo genera un sistema $\left(A^T.A\right).x = \left(A^T.b\right)$ por cada nodo; por lo tanto, habrá un vector gradiente por cada nodo.

V.1.1.2.4.4. Distancias entre nodos

La distancia entre los nodos es calculada con la raíz cuadrada de la suma de los cuadrados de las diferencias en las entradas (coordenadas) de cada nodo:

$$d_{1-n} = \sqrt{\Delta x_{1-n}{}^2 + \Delta y_{1-n}{}^2 + \Delta z_{1-n}{}^2} \qquad (28)$$

Donde:

Δx_{1-n} : es la diferencia en la entrada x entre el nodo 1 y el vecino "n".

x, y, z: son las entradas del sistema.

V.1.1.2.4.5. Selección de los vecinos más cercanos

El algoritmo selecciona un número de vecinos más cercanos (valor que el usuario especifica) para cada nodo. Una vez que el algoritmo tiene todas las distancias entre un nodo y sus vecinos, éste selecciona las menores distancias. Las distancias son seleccionadas en orden creciente; esto quiere decir, la menor distancia primero, después la 2da menor, la 3ra menor, y así sucesivamente.

V.1.1.2.4.6. Rutina Gauss Jordan

Una vez que el algoritmo tiene los valores del vector b y la matriz A, éste multiplica a cada uno de ellos por la transpuesta de la matriz A: A^T, y luego aplica una rutina de eliminación Gauss Jordan para calcular una aproximación del vector gradiente x. La rutina Gauss Jordan con pivoteo ha sido seleccionada ya que es un método estable y eficiente, compacto y comprensible [Press 1999]. La solución del sistema Gauss Jordan (un vector) es la aproximación del gradiente.

V.1.1.2.4.7. Selección de las regiones a refinar

El algoritmo luego calcula la magnitud del vector gradiente:

$$\left|\overline{\nabla F}\right|_{node_n} = \sqrt{\left(\frac{\partial F}{\partial x}\right)^2 + \left(\frac{\partial F}{\partial y}\right)^2 + \left(\frac{\partial F}{\partial z}\right)^2} \qquad (29)$$

Donde:

$\left|\nabla F\right|_{node_n}$: es la magnitud del vector gradiente del nodo n.

$\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}$: son los componentes del vector gradiente.

y calcula el promedio de las magnitudes de todos los nodos:

$$\left\langle \left| \nabla F \right| \right\rangle_{\text{todos los nodos}} = \frac{\left| \nabla F \right|_{nodo_1} + \left| \nabla F \right|_{nodo_2} + ... + \left| \nabla F \right|_{nodo_n}}{\text{número de nodos}} \quad (30)$$

Donde:

$\left\langle \left| \nabla F \right| \right\rangle$: es el promedio de todas las magnitudes.

Para seleccionar los nodos a refinar, el algoritmo calcula el valor de:

$$\frac{\left| \overline{\nabla F} \right|_{nodo_n}}{\left\langle \left| \nabla F \right| \right\rangle_{\text{todos los nodos}}} \quad (31)$$

y selecciona los nodos que tengan el valor previo mayor que un parámetro crítico (valor umbral de refinación) especificado por el usuario:

$$\frac{\left| \overline{\nabla F} \right|_{nodo_n}}{\left\langle \left| \nabla F \right| \right\rangle_{\text{todos los nodos}}} > parámetro \quad (32)$$

V.1.1.2.4.8. Generación de nuevos puntos

Una vez que el algoritmo tenga los nodos los cuales van a ser refinados, éste genera una cantidad (especificada por el usuario) de valores al azar. Éstos valores al azar son generados entre los valores de las entradas del nodo a refinar y sus vecinos más cercanos. El algoritmo crea una nueva matriz de valores de entradas con éstos valores, y luego calcula (con el uso del simulador) el valor de la salida para cada nueva serie de valores de entradas. Para crear la colección refinada, el algoritmo inserta estas series de valores entradas – salida en la matriz de la colección original, y luego genera una red neuronal con ésta colección.

El algoritmo de refinación será probado comparando el desempeño de una red neuronal entrenada con la colección refinada contra otra entrenada con la colección original.

Se puede visualizar que la colección refinada tiene más series de datos que la colección original. Para comparar la red neuronal entrenada con la colección refinada, con la entrenada con la colección "no refinada", ambas colecciones necesitan tener la misma cantidad de series de datos. Con el propósito de que las dos colecciones tengan el mismo número de series de datos, el algoritmo genera series de datos al azar para las entradas (usando el algoritmo de rejilla al azar) y calcula el valor de las salidas para cada una de éstas series usando el simulador; luego el algoritmo inserta éstas series en la colección original de datos. Ésta nueva colección va a ser llamada "colección original expandida", y por lo tanto tiene el mismo número de series que la colección refinada.

V.1.1.2.5. Creación de la red neuronal

Las redes neuronales son creadas automáticamente conectando Visual Basic® y NN Tool 2000®. El programa primero crea un archivo .pat, el cual contiene todos los datos que van a ser usados en la etapa de entrenamiento, es decir, todas las series que contienen los datos de las entradas y una salida; y luego un archivo .cob, el cual contiene todos los parámetros necesarios para la creación de la red neuronal. Éstos parámetros son seleccionados por el usuario, e incluyen:

- mínimo número de nodos
- máximo número de nodos
- mínimo número de iteraciones en la etapa de entrenamiento
- máximo número de iteraciones en la etapa de entrenamiento
- pasos en la etapa de entrenamiento
- valor umbral para seleccionar una mejor red neuronal, entre otros.

Una vez que el algoritmo ha creado éstos archivos, el programa ejecuta el archivo .cob, e inmediatamente NN Tool 2000® se ejecuta, creando la red neuronal, y la cual es guardada en una carpeta especificada por el usuario.

V.1.1.2.6. Pronósticos

El algoritmo de pronósticos, genera valores para todos las entradas entre los límites de éstas, y luego envía cada serie a la red neuronal, con el objeto de obtener el valor de salida producido por la red neuronal que está siendo estudiada.

Existen dos tipos de generación de pronósticos de los cuales el usuario puede escoger: pronóstico con rejilla fija, y pronóstico con rejilla al azar. Éstos valores para el pronóstico son generados exactamente igual que las entradas cuando se crea la colección original con rejilla fija ó rejilla al azar.

Una vez que los valores para el pronóstico son generados, el programa envía éstos valores de entradas a ACM con el objetivo de obtener el valor "real" de la salida para esa serie de entradas, y luego calcula la suma del cuadrado de los errores absolutos entre los valores de la salida pronosticada, y el valor calculado por ACM (valor real):

$$\textit{error absoluto} = \textit{valor real} - \textit{salida de la red neuronal} \qquad (33)$$

donde:

valor real: es el valor obtenido usando el simulador

salida de la red neuronal: es el valor obtenido usando la red neuronal (valor pronosticado por ésta)

V.1.1.2.7. Estrategias en caso de singularidad

En el programa se construyeron dos estrategias en caso de que el simulador sea incapaz de encontrar solución al modelo presente: *homotopía* y *saltos en las entradas*.

Homotopía

La homotopía proporciona una manera de moverse desde una solución que ha convergido hasta otra solución, con diferentes valores de las series de entradas. Ésta es una técnica conveniente cuando es difícil obtener convergencia para una especificación particular, pero requiere que se tenga una solución previa para una especificación diferente [ACM® help topics.]

La homotopía funciona moviéndose a lo largo de un trayecto hasta la nueva solución, y resolviendo un número de puntos interinos a lo largo del trayecto.

Si la simulación tiene previamente una solución que ha convergido (HOM1), y no se encontró solución para HOM2, el próximo valor dado al simulador es:

$$Valor\ de\ \text{homotopia} = HOM1 + \theta.(HOM2 - HOM1) \qquad (34)$$

donde θ: es un valor seleccionado por el usuario, y varía de 0 a 1. Indica cuántos pasos la simulación debe hacer en el rango entre HOM1 y HOM2.

Cuando una solución a la homotopía es encontrada en el trayecto, a ésta se le suma el *valor de homotopía* y se repite el proceso, así sucesivamente hasta que HOM2 es alcanzado. Éstos saltos incrementales son hechos para todas las entradas en la simulación.

Si la estrategia de homotopía falla, el programa se detiene.

Los valores obtenidos de la homotopía no son insertados a la matriz de la colección original; la estrategia es usada solamente para intentar conseguir un valor que no converge.

Saltos en las entradas

La estrategia es añadir un valor incremental seleccionado por el usuario, a los valores de todas las entradas en la serie. Los datos obtenidos son insertados en la matriz de la colección original. La estrategia es conveniente para saltar de una solución irresoluble, por ejemplo una división por cero.

V.1.1.3. Implementación

Los principales componentes de SimWrap son: Hoja de Inicialización (Hoja de Microsoft Excel®), Aspen Custom Modeler®, NN Tool 2000®, y opcionalmente: Hoja de Interfase (Hoja de Microsoft Excel®).

V.1.1.3.1. Hoja de Inicialización

SimWrap debe ser inicializado desde una hoja de Excel llamada "*hoja de inicialización*". Los parámetros especificados en ésta deben estar en concordancia con el modelo en el simulador. La hoja debe contener:

- nombre de las variables de entrada, y sus nombres de exportación
- nombre de las variables de salida, y sus nombres de exportación
- nombre de las variables fijas, y sus nombres de exportación (si existe alguna)
- valores del límite superior e inferior de las variables de entrada
- valores de las variables fijas (si existe alguna)

Los nombres de las variables deben ser los mismos a como están escritos en el simulador, y deben ser precedidos por el número de bloque y un punto. Por ejemplo, para la variable T en el bloque B1 se debe expresar como: B1.T

Los nombres de exportación pueden ser elegidos por el usuario, y son con fines de presentación.

La hoja de inicialización contiene también el nombre y la vía (localización en el disco) del archivo ACM donde el modelo está escrito.

La estructura de la hoja de inicialización se propone como la mostrada en la figura 13:

ACM File	D:\My Simulations\Modeler\Non Linear3.acmf

	Export Name	Simulation Name	lower bound	upper bound
Input				
	friction coeficient of the pulp into the refiner	B1.Mt	0.5	1
	specific refining energy	B1.E	1000000	9000000
	angular speed	B1.w	150	350
Output				
	residence_time	B1.T		
Fixed			**Value**	
	refining consistency	B1.c	0.08	
	internal radius	B1.r1	0.08	
	external radius	B1.r2	0.15	
	friction coeficient between the wet wood and the steel	B1.Mr	0.15	

Figura 13. Hoja de Inicialización

La hoja de inicialización debe presentar las siguientes especificaciones:

1. La celda que contiene el nombre y vía del archivo ACM debe tener el nombre: ACM_filename
2. La celda donde está escrito "Input" debe tener el nombre: Input
3. La celda donde está escrito "Output" debe tener el nombre: Output
4. La celda donde está escrito "Fixed" debe tener el nombre: Fixed

SimWrap encontrará los parámetros en la región de la hoja mediante los nombres de exportación. Por lo tanto SimWrap no encontrará un parámetro para el cual no esté escrito el nombre de exportación.

Para los parámetros requeridos (entradas y salidas), SimWrap buscará desde la celda que está al lado del título del parámetro ("Input" ó "Output") un nombre de exportación, de arriba hacia abajo, hasta que uno sea encontrado. Si no se

encuentra ningún nombre 10 filas abajo, la inicialización fallará. Para los parámetros fijos, SimWrap sólo buscará en las 6 primeras filas debajo de la celda de al lado de la llamada “Fixed”; una vez que ésta celda es alcanzada, SimWrap considerará que no existen parámetros fijos en la simulación.
Si existe más de una entrada, salida ó parámetro fijo, no pueden presentar filas de por medio, ya que SimWrap detiene la búsqueda de los parámetros pertinentes cuando encuentra una celda vacía.
Para las entradas, el límite inferior debe ser menor que el límite superior.

V.1.1.3.2. Hoja de interfase

Cuando SimWrap comienza, aparece una ventana la cual insta al usuario a seleccionar la manera de inicializar el algoritmo. Si el usuario selecciona inicializar el algoritmo usando una hoja de Excel, éste debe especificar la vía y nombre del archivo.
El archivo de Excel (hoja de interface) debe presentar las siguientes especificaciones:
Debe tener dos hojas con los siguientes nombres:
1) Set up: La cual contiene:

- El nombre del dll y la clase en la cual el componente funcional se encuentra, separados ambos por un punto, por ejemplo: “SimpleFunction.acmFunc”. El dll debe estar registrado previamente en el sistema de la computadora usada. La celda en la cual se especifica éste parámetro debe llamarse: progid. El componente funcional es el dll el cual contiene el código para correr la simulación y recoger los datos de inicialización de Excel®.
- La vía y el nombre de la hoja de inicialización. La celda debe tener por nombre: initstr
- El nombre del dll y la clase en la cual el algoritmo de selección se encuentra. separados ambos por un punto; por ejemplo:

" SelectionAlgorithm.SelAlgo1" . El dll debe estar registrado previamente en el sistema de la computadora usada.

La figura 14 muestra un ejemplo de la *hoja de interfase: run*:

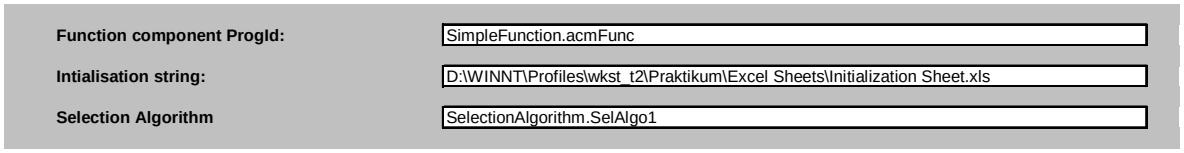


Figura 14. Hoja de Interfase: Setup

2) AlgorithmSetUp: La cual contiene los parámetros mostrados en la tabla 6:

Tabla 6. Parámetros en la hoja AlgorithmSetUp

Parametro	Nombre de la celda
Calculations (normal): Número de cálculos normales para realizar en el algoritmo de generación de datos. El parámetro sólo puede ser usado en el algoritmo de rejilla al azar.	Calculations
Calculations (exponential). Número de cálculos exponenciales para realizar en el algoritmo de generación de datos. Puede ser usado en ambos algoritmos de generación de datos.	NumC
Algorithm for data generation: Nombre del algoritmo de generación de datos. Adepta los valores: " random" cuando se quiere usar el algoritmo de rejilla al azar, y " fixed" cuando se quiere usar el algoritmo de rejilla fija.	strAlgorithm
Increment step in case of singularity: Salto incremental en caso de singularidad.	IncrNoSol
Homotopy parameter: Parámetro para la homotopía (θ)	Homotopy
Use refinement algorithm?: Pregunta si se quiere usar el algoritmo de refinación. Parámetro booleano, el cual acepta " yes" (sí), ó " no"	bRefAlgo
Si el usuario selecciona usar el algoritmo de selección, los siguientes parámetros deben ser indicados:	

Parametro	Nombre de la celda
Algorithm for data refinement: El usuario debe escribir " all connections"	refAlgo
Number of points to select from the collection: Número de puntos a seleccionar de la colección original para los cuales se calculará el gradiente. Debe ser un valor positivo, menor ó igual al número de series en la colección original.	numRandom
Number of nearest neighbors to select: Número de vecinos más cercanos a seleccionar para el cálculo del gradiente.	nNst
Number of new points to generate between bounds: Número de nuevos puntos a generar entre los límites de las entradas. Debe ser un valor entero.	NumNewPoints
Threshold value to perform refinement: Valor umbral para realizar refinamiento.	criter
Create Neural Network?: Pregunta al usuario si desea crear una red neuronal. Parámetro booleano, y acepta " yes" (sí), ó " no"	bCreateNN
Si el usuario selecciona crear una red neuronal, los siguientes parámetros deben ser indicados:	
Path of the .Pat file to be created: Vía del archivo .Pat que será creado. Se tiene que especificar el disco y directorio de la misma manera como se hace en DOS. Los espacios en blanco no están permitidos.	FilePath
Name of the file to be created: Nombre del archivo a ser creado. Los espacios en blanco no están permitidos.	nameFile
Path and name of the NN Tool: Vía y nombre del archivo ejecutable de la herramienta que usará el algoritmo para crear la red neuronal. En la presente tesis se usó NN Tool 2000, nombre del archivo: nn.exe	strNNTool
Minimum number of nodes: Número mínimo de nodos que presentará la red neuronal. Número entero mayor ó igual a 1.	byMinNodes
Maximum number of nodes: Número máximo de nodos que presentará la red neuronal. Número entero menor ó igual a 32.	byMaxNodes
Minimum number of iterations: Número mínimo de iteraciones para el entrenamiento. Número entero mayor ó igual a 10.	byMinIter
Maximum number of iterations: Número máximo de iteraciones para el entrenamiento. Número entero menor ó igual a 100.	byMaxIter
Number of steps in training phase: Número de etapas en la fase de entrenamiento.	byStepsTrain
Number of steps to test set selection: Número de pasos en la colección original para selección de serie para la validación (test).	byStepsData
Threshold value to perform refinement: Valor umbral para realizar refinamiento.	byPercent

Parametro	Nombre de la celda
Prognosis: Tipo de algoritmo de generación de datos para realizar el pronóstico. Puede ser “random” para generar una rejilla al azar, ó “fixed” para generar una rejilla fija. Si la celda permanece en blanco, el pronóstico no se reliza.	Prognosis
Number of prognosis between bounds (exponential): Número de pronósticos entre los límites (exponencial). El valor es el mismo en su tipo que el número de cálculos (exponenciales) en el algoritmo de generación de datos.	NumberOfPrognosis
Number of prognosis between bounds (normal): Número de pronósticos entre los límites (normal). El valor es el mismo en su tipo que el número de cálculos (normal) en el algoritmo de generación de datos.	NumP
Create a new Excel file?: Pregunta al usuario si desea crear un nuevo archivo Excel en donde la data será mostrada. Parámetro booleano, y acepta “yes” (sí), ó “no”	bExCreate
Path and name of the file to be created: Vía y nombre del archivo a ser creado.	ExcelName
Status of the simulation: Sólo “steady” es aceptado.	state

Si el usuario selecciona no crear otro archivo para enviar los datos, el archivo Excel: *hoja de Interfase*, debe contener otra hoja llamada “run”, y es donde todos los datos van a ser enviados. Si el usuario selecciona crearla, SimWrap creará un archivo Excel con el nombre especificado, conteniendo una hoja llamada “run”, donde todos los datos van a ser mostrados. El archivo va a ser guardado en la vía especificada.

La figura 15 muestra un ejemplo de la *hoja de Interfase*: *AlgorithmSetUp*.

Calculations (normal)	40
Calculations (exponential)	

Algorithm for data generation :	random
Increment step in case of singularity :	0.001
Homotopy parameter :	0.5

Use Refinament Algorithm ? :	yes
Algorithm for data refinament :	all connections
Number of points to select from the collection :	40
Number of nearest neighbors to select :	6
Number of new points to generate between bounds :	1
Threshold value to perform refinament :	1.5

Create Neural Network ? :	yes
Path of the .Pat file to be created :	c:\test
Name of the file to be created :	test
Path and name of the NN Tool :	d:\program files\nn\nn.exe

Neural Network Configuration

Minimum number of nodes :	1
Maximum number of nodes :	16
Minimum number of iterations :	10
Maximum number of iterations :	100
Number of steps in the training phase :	10
Number of steps to test set selection :	5
Threshold value to perform refinement :	2

Prognosis :	random
Number of prognosis between bounds (exponential) :	
Number of prognosis (normal) :	40

Create a New Excel File ? :	no
Path and name of the file to be created :	D:\TestPraktikum

Status of the simulation :	steady

Figura 15. Hoja de Interfase: AlgorithmSetUp

V.1.1.3.3. Aspen Custom Modeler® (ACM)

Las versiones de ACM que SimWrap acepta son: v 10.1 y 10.2. El modelo debe estar escrito previamente en éste. Todos los parámetros (variables y constantes) deben ser fijados como libres (“free”). Esto puede ser hecho haciendo doble click en el icono del bloque, y cambiando la especificación de la variable a “free”, ó en el editor de texto del bloque (donde el modelo está escrito).

SimWrap fijará los parámetros en ACM en concordancia a los tomados en la *hoja de inicialización*.

V.1.1.3.4. NN Tool 2000®

SimWrap usa entrenamiento automático para entrenar la red neuronal. Ésta separa series para el entrenamiento, y series para el "test". Ésta selección es hecha con el parámetro "Number of steps to test set selection" (Número de pasos en la colección original para la selección de serie para el test). Por ejemplo, si el usuario selecciona 5 en "Number of steps to test set selection", NN Tool 2000® tomará una serie para el "test" de cada 5 series para el entrenamiento.

Es importante mencionar que NN Tool 2000® crea una red neuronal para cada número de nodos que prueba (desde el mínimo número de nodos, hasta el máximo), y una red neuronal para cada número de iteraciones, y luego selecciona la que proporciona el menor error en el pronóstico de las series de test; la que selecciona pasa a ser la que se utiliza, y las demás son desechadas.

Las redes creadas con NN Tool 2000® tienen una capa escondida, y la función de transferencia es sigmoide. El entrenamiento es *feedforward*.

SimWrap crea un archivo .pat, el cual contiene toda la información para el entrenamiento (esto es: nombre de las entradas y salidas, y sus valores), y un archivo .cob, el cual contiene la información para la inicialización de NN Tool 2000®, como el número mínimo y máximo de nodos, iteraciones, etc. Éstos valores son seleccionados por el usuario.

Luego SimWrap crea un archivo .bat, el cual contiene la vía y el nombre del archivo .exe que ejecuta a NN Tool 2000®, y la vía y el nombre del archivo .cob que ha sido creado previamente. La estructura del archivo .bat debe ser del estilo:

"d:\program files\nn\nn.exe" "d:\winnt\profiles\wkst_t2\praktikum\test.cob"

V.1.1.4. Ventanas de Inicialización de SimWrap

Una vez que el programa comienza, aparece en la pantalla la ventana mostrada en la figura 16:

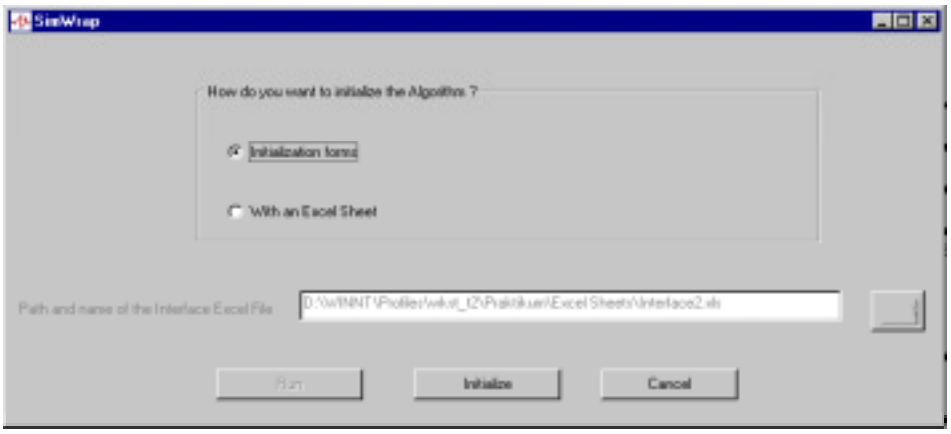


Figura 16. Primera ventana de SimWrap

El usuario puede seleccionar inicializar el algoritmo desde una hoja de Excel (hoja de Interfase) ó hacerlo mediante las ventanas del programa.

Si el usuario selecciona inicializar el algoritmo usando las ventanas, el diálogo mostrado en la figura 17 aparece en la pantalla.

Por el contrario, si el usuario selecciona inicializar el algoritmo desde la hoja Excel, debe especificar en el recuadro que se activa, la vía y el nombre del archivo de la hoja de Interfase.

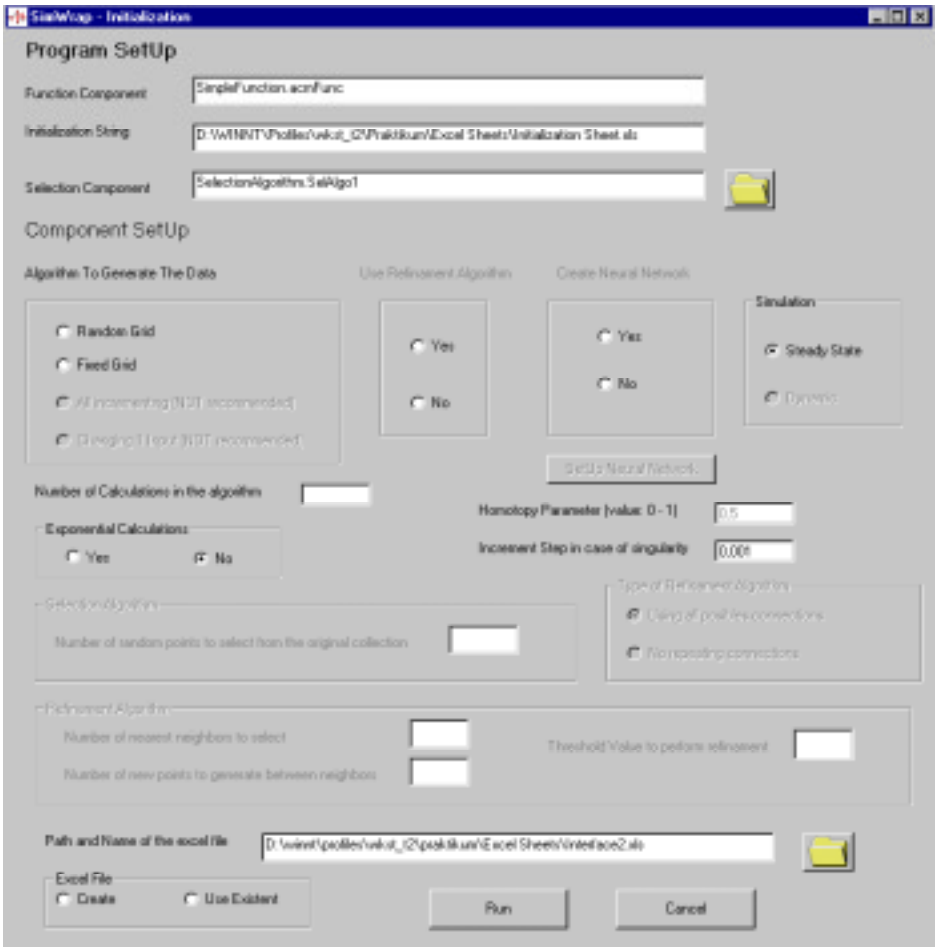


Figura 17. Ventana de inicialización del algoritmo

El usuario puede introducir los parámetros deseados en un ambiente Windows® amigable, y hacer las diversas selecciones que se presentan en la utilización de los algoritmos. Los parámetros son los mismos que los introducidos en la *hoja de interfase*.

Existen ciertas opciones que no están disponibles (no es posible seleccionarlas), y es debido a la evolución del algoritmo.

Si el usuario selecciona crear una red neuronal, el botón de inicialización de ésta (SetUp Neural Network) se activa, y será capaz de presionarlo. Una vez que el usuario presiona dicho botón, la ventana mostrada en la figura 18 aparece en la pantalla, la cual contiene todos los parámetros necesarios para la inicialización de la red neuronal, además de los pertinentes para el pronóstico. La ventana

aparece en la pantalla con los valores de inicialización pre-determinados, los cuales pueden ser cambiados. Los parámetros de la inicialización de la red neuronal son los mismos que los presentados en la *hoja de Interfase*.

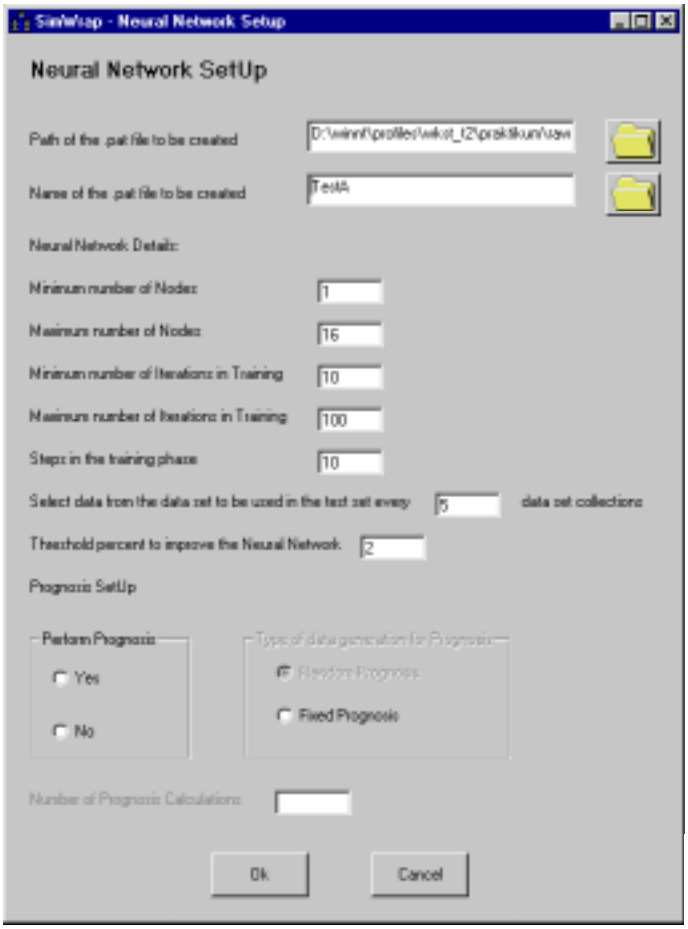


Figura 18. Ventana de inicialización de la red neuronal

Una vez que el usuario ha introducido todos los parámetros necesarios para la corrida, el botón "run" se activa. Presionando dicho botón, se inicia la corrida, y la barra de progreso mostrada en la figura 19 aparece a continuación. Con esta barra, se indica al usuario que SimWrap está realizando los cálculos pertinentes.

Figura 19. Barra de Progreso de la corrida

Cuando SimWrap ha calculado todos los valores de los gradientes de la salida que está siendo calculada, éste los reporta en una ventana mostrada en la figura 20.

Figura 20. Ventana de Valores de gradientes

En la figura 20 por ejemplo, se muestran los valores de los gradientes para la variable (salida) T del bloque B1.

También se muestra en la parte inferior de la ventana el valor promedio de los gradientes calculados. Con éste valor y viendo los demás valores de los gradientes, se le proporciona al usuario una idea de los valores manejados, y luego se le da la opción de cambiar el valor umbral, y el número de puntos nuevos generados entre los vecinos, mediante las ventanas mostradas en las figuras 21 y 22 respectivamente.

Figura 21. Ventana preguntando al usuario si desea cambiar el valor umbral

Figura 22. Ventana preguntando al usuario si desea cambiar el número de nuevos puntos a generar

SimWrap calcula los gradientes de todas las salidas que se inicializaron, por lo cual, las figuras 19, 20, 21 y 22, se muestran una vez por cada salida que exista. Una vez que la corrida ha concluido sin errores, aparece en la pantalla la ventana mostrada en la figura 23, la cual indica que la corrida ha concluido satisfactoriamente, y el programa ha finalizado.

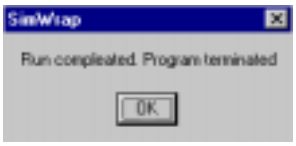


Figura 23. Ventana final. La corrida se ha completado satisfactoriamente

V.1.1.5. Uso de la red neuronal en Excel®

Una vez que el programa ha creado la red neuronal, éste la guarda en un archivo .dsc, el cual se localiza en la misma carpeta donde el archivo .pat ha sido guardado. El programa crea una red neuronal entrenada con la colección original expandida (no refinada), y otra entrenada con la colección refinada, para cada salida. Luego el programa guarda la red neuronal entrenada con la colección original expandida utilizando el mismo nombre que fue especificado para el archivo .pat, incluyendo la terminación "real" seguido del número de la salida; la red neuronal entrenada con la colección refinada es guardada con el nombre del archivo .pat seguido del número de la salida. Por ejemplo, se especifica el nombre del archivo .pat como *Prueba*, y supongamos que el modelo tiene dos salidas. El programa guardará el archivo de la red neuronal entrenada con la colección original expandida como: *Prueba_real1* para la primera salida, y *Prueba_real2* para la segunda salida; en el caso de la red neuronal entrenada con la colección refinada, el nombre del archivo será: *Prueba_1* para la primera salida, y *Prueba_2* para la segunda salida.

NN Tool 2000® permite el uso de la red neuronal creada en Excel®. Esto permite la utilización de la red neuronal creada sin la necesidad de abrir ni de tener instalado en el sistema el software NN Tool 2000®. Las aplicaciones de ésta posibilidad son amplias, y se puede mencionar el uso de una red neuronal creada con el uso del simulador para la representación de un modelo, prescindiendo del simulador y con la posibilidad de obtener resultados mucho más rápido que el simulador, y la posibilidad de uso online.

V.1.1.6. Comportamiento del sistema durante el uso de SimWrap

Durante la corrida, la fase que más tiempo consume es la obtención de los resultados mediante el uso del simulador, ya que se requiere una corrida del simulador por cada serie de datos que se le envía.

La figura 24 muestra el comportamiento del sistema Pentium® III durante la corrida de la ecuación no lineal para ω y c, la cual se hizo con 40 cálculos normales y 40 pronósticos.

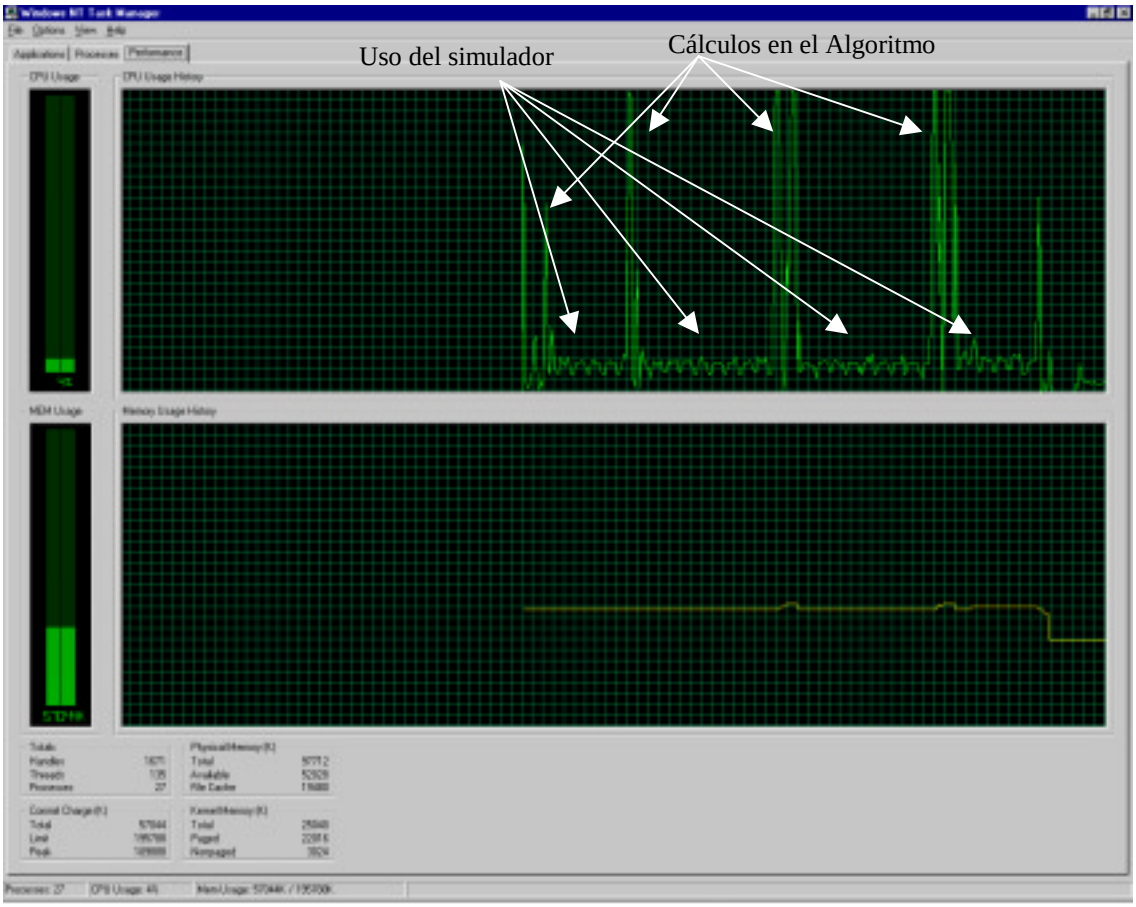


Figura 24. Comportamiento del sistema durante la corrida

El recuadro superior derecho de la figura 24 muestra la gráfica del uso del cpu (eje vertical) en contra del tiempo (eje horizontal). La ventana es disponible en Windows NT (Windows NT Task Manager).

En general, todas las corridas que se hicieron en el presente trabajo de grado, no tardaron más de 10 minutos en un procesador Pentium® III; pero se trataron de modelos sencillos con máximo dos salidas y dos entradas, y con un número de cálculos pequeño (40 y 50 cálculos normales).

Adicionalmente la figura 24 muestra que la etapa en el uso del simulador es la que menos recursos del sistema consume (uso del CPU) pero es la más larga, mientras que el cálculo en los algoritmos (como la creación de la rejilla de series de entrada, y los cálculos en el algoritmo de refinación) es la etapa que más recursos del CPU consume. Es recomendable no tener otros programas ejecutando mientras la corrida de SimWrap es realizada.

Por otra parte, el recuadro inferior derecho muestra el uso de la memoria del sistema (eje vertical) en contra del tiempo (eje horizontal). Se aprecia una disminución del uso de memoria una vez que la corrida ha concluido. Esto es debido a que la memoria usada durante la corrida es liberada una vez que ésta ha concluido.

Aunque la memoria no fue un factor limitante en corrida hecha en la figura 24, lo podría ser cuando las colecciones de datos manejadas sean extensas en dimensión.

Por lo tanto, es recomendable que los sistemas en que las corridas sean hechas, tengan un mínimo de 16 Mb de memoria RAM.

V.2. Resultados en los Modelos

Tres sistemas fueron usados en el estudio: Un modelo de fermentador en estado estacionario, una ecuación altamente no linear que predice el tiempo de residencia de la pulpa de la madera en una refinería de hojuelas de madera, y una ecuación que genera dos picos en su representación tridimensional.

1) *Modelo del Fermentador:*

Un fermentador con una entrada y una salida de flujo en estado estacionario, puede ser modelado con dos ecuaciones no lineales (ecuación 35 y 36): [Kuhlmann, 1997]

$$\mu(s)X - XD = 0 \tag{35}$$

$$-\frac{\mu(s)}{Y}X + (S_f - S)D = 0 \tag{36}$$

donde X es la concentración celular, S_f es la concentración del sustrato de entrada, Y es el coeficiente de producción, y *D* la razón de dilución, la cual, en este caso es: *F/V*, donde *F* es el flujo de entrada, y *V* es el volumen del reactor. Los valores nominales de *Y* y S_f son tomados como 0.4 g/g y 1 g/l respectivamente.

$\mu(s)$ es la tasa de crecimiento específico, la cual es modelada por una relación cinética Monod, que se muestra en la ecuación 37.

$$\mu(s) = \mu_{max}\frac{S}{K+S} \tag{37}$$

donde *S* es la concentración del sustrato, μ_{max} es la tasa máxima de crecimiento específico, y *K* es la constante de saturación del sustrato. Los parámetros nominales para el modelo Monod son *K* = 0.05 g/l y μ_{max} = 0.4 1/h.

Si se sustituye la ecuación 37, en 35 y 36, y se despeja *S* en la ecuación 35 y *X* en la ecuación 36, se obtiene las ecuaciones 38 y 39.

$$S = \frac{FK}{(\mu_{max} V) - F} \tag{38}$$

$$X = \frac{(S_f - S)FY}{\mu_{max} V\left(\frac{S}{K+S}\right)} \tag{39}$$

Las ecuaciones 38 y 39 son las usadas en el modelo ACM®.

2) *Ecuación no lineal:*

El modelo es una ecuación altamente no lineal que predice el tiempo de residencia de la madera en una refinería de hojuelas de madera [Miles, 1991] y es dada por la ecuación 40.

$$\tau = \frac{\mu_r}{\mu_t} \frac{aEc}{\varpi^3\left(r_2^{\,2} - r_1^{\,2}\right)} \ln\left(\frac{r_2}{r_1}\right) \tag{40}$$

Donde τ es el tiempo de residencia en la refinería, μ_r el coeficiente de fricción entre la madera mojada y el acero (0,15 – 0,25), μ_t el coeficiente de fricción de la pulpa dentro de la refinería (0,5 – 1,0), *a* es una constante (fijada en 1), *E* la energía de refinamiento específica (1.10^6 – 9.10^6 J kg^{-1}), *c* la consistencia de refinamiento (0,08 – 0,12), ω la velocidad angular (150 – 350 rad s^{-1}), r_1 el radio interno (0,08 – 0,12 m) y r_2 el radio externo (0,15 – 0,45 m).

3) *Ecuación de dos picos tridimensionales:*

La expresión 41 es una ecuación no lineal la cual genera dos picos en su representación tridimensional.

$$z(x,y) = S_1 e^{-\left[\frac{(x-a_x)^2+(y-a_y)^2}{\sigma_1}\right]} + S_2 e^{-\left[\frac{(x-b_x)^2+(y-b_y)^2}{\sigma_2}\right]} \quad (41)$$

donde:

S_1: es la altura del primer pico

S_2: es la altura del segundo pico

a_x: es la primera coordenada del centro del primer pico

a_y: es la segunda coordenada del centro del primer pico

b_x: es la primera coordenada del centro del segundo pico

b_y: es la segunda coordenada del centro del segundo pico

σ_1: es la amplitud del primer pico

σ_2: es la amplitud del segundo pico

x: es la primera coordenada

y: es la segunda coordenada

Éste modelo fue usado en el estudio ya que genera dos zonas de alto gradiente, donde el resto de la superficie es de comportamiento suave; comportamiento necesario para validar el estudio.

V.2.1. Selección de puntos en zonas de alto gradiente

El estudio se limitó a simular los modelos con una y dos entradas debido principalmente a que el ser humano sólo es capaz de visualizar hasta 3 dimensiones, además fue un criterio de acotamiento en el estudio.

Todas las figuras mostradas como resultados, son representaciones gráficas de la colección original; los puntos generados por el algoritmo de refinación son mostrados de forma aislada del gráfico principal (como puntos sobre la superficie).

V.2.1.1. Modelo del Fermentador

La figura 25 fue obtenida graficando la colección original (x azules) y los puntos generados por el algoritmo de refinación (círculos azules).
Datos introducidos para la corrida:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 50 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 45
- Número de vecinos usados para el cálculo del gradiente: 3
- Valor umbral: 1
- Nodos generados entre vecinos para refinar: 1
- Variable de entrada y rango de variación: F entre 0 y 2
- Variables de salida: X y S

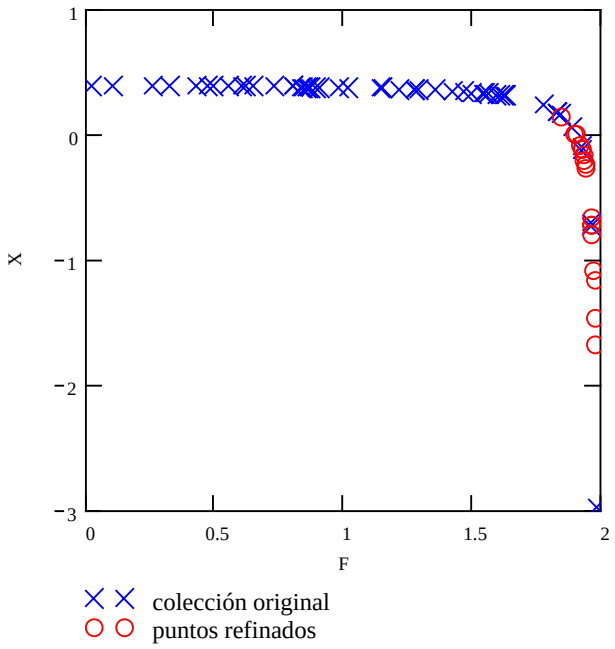


Figura 25. Modelo del Fermentador para X. Caso 1

En la figura 25, se puede observar que existe un comportamiento asintótico en F=2. Es importante resaltar que el valor de X pasa a ser menor que cero en ciertas partes de la figura; aunque no existe interpretación real de concentraciones menores que cero, ésta fue considerada sólo por propósitos matemáticos.

En lo que respecta a S, la figura 26 muestra un comportamiento similar al anteriormente descrito para X.

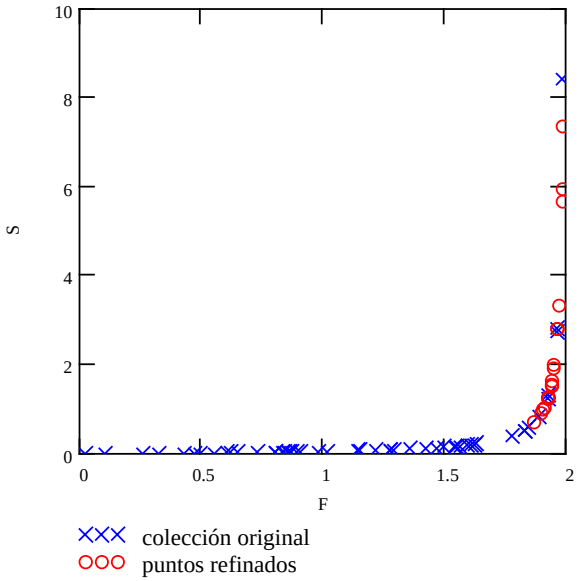


Figura 26. Modelo de Fermentador para S. Caso 1

Para obtener las figuras 27 y 28 se usaron los siguientes datos en la corrida:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 20 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 20
- Número de vecinos usados para el cálculo del gradiente: 3
- Valor umbral: 2
- Nodos generados entre vecinos para refinar: 1
- Variable de entrada y rango de variación: F entre 0 y 5
- Variables de salida: X y S

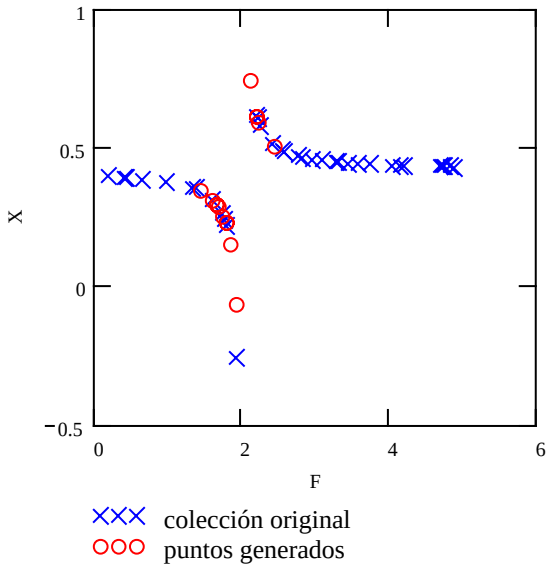


Figura 27. Modelo del Fermentador para X. Caso 2

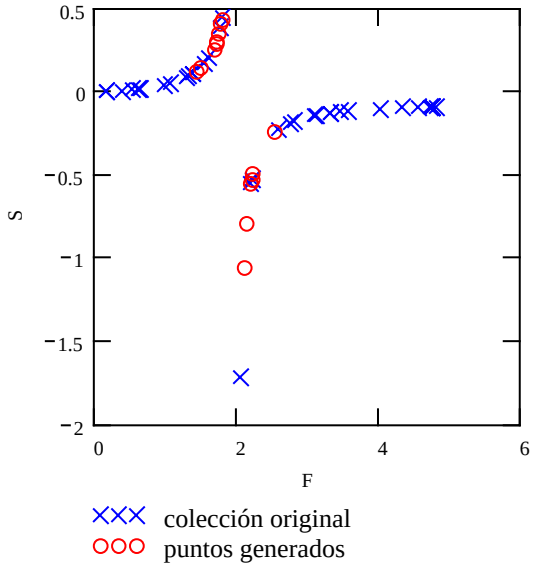


Figura 28. Modelo del Fermentador para S. Caso 2

Como se puede observar, tanto para X como para S, los puntos son generados (círculos rojos) en la región de máxima pendiente.

Las próximas dos figuras (29 y 30) fueron generadas graficando la colección original (superficie) y los puntos generados por el algoritmo de refinación (puntos rojos). La figura 29 muestra el modelo para la salida X, y la figura 30 para la salida S. Las figuras muestran la variable de salida graficada en contra de las variables de entrada.

Los datos introducidos para la corrida fueron los siguientes:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 30 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 30
- Número de vecinos usados para el cálculo del gradiente: 5
- Valor umbral: 2
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: F entre 0 y 2; V de 5 a 10

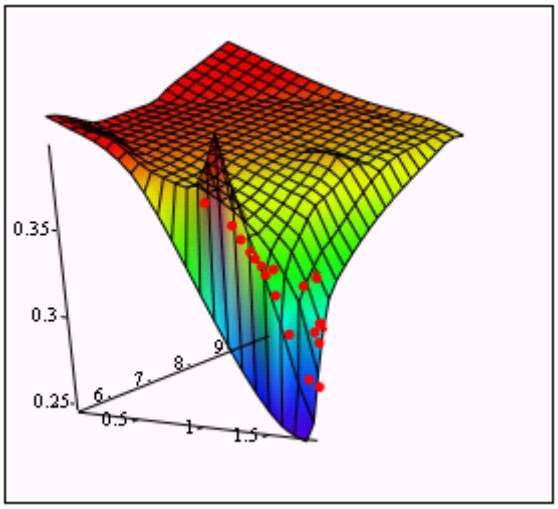


Figura 29. Modelo del Fermentador para X. Caso 3

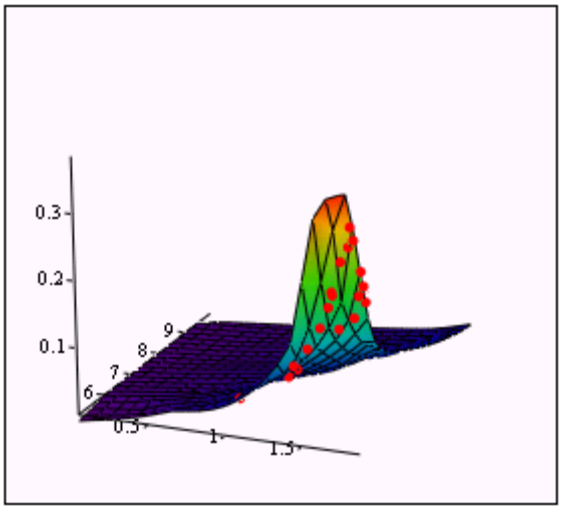


Figura 30. Modelo de Fermentador para S. Caso 3

Nuevamente los gráficos muestran que los puntos son generados (puntos rojos) en las regiones de máximo gradiente.

V.2.1.2. Ecuación no lineal para el cálculo del tiempo de residencia

La siguiente figura (número 31) fue hecha graficando la variable de salida en contra de las variables de entrada. La colección fue generada con los siguientes parámetros:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 40 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 40
- Número de vecinos usados para el cálculo del gradiente: 5
- Valor umbral: 1
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: ω de 150 a 350; *c* de 0,08 a 0,28
- Variable de salida: τ

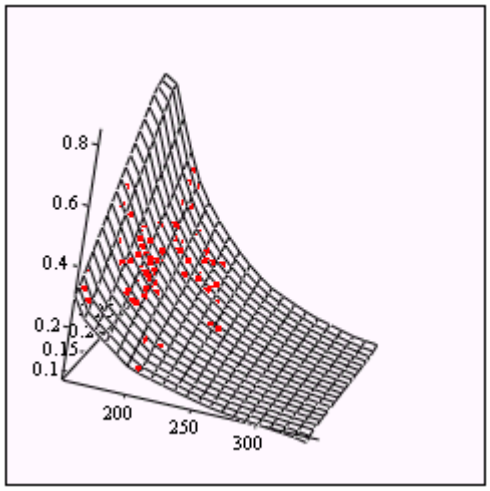


Figura 31. Tiempo de residencia contra ω y c

En éste caso, los puntos fueron generados (puntos rojos) mayormente en una región donde el gradiente es mayor que el valor medio.

V.2.1.3. Ecuación de dos picos

La figura 32 fue obtenida graficando la colección original (superficie), y los puntos generados por el algoritmo de refinación (puntos rojos). Las figuras muestran la variable de salida graficada en contra de las variables de entrada. La colección de datos fue generada con los siguientes valores:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 50 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 50
- Número de vecinos usados para el cálculo del gradiente: 6
- Valor umbral: 1
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: *x* de 0 a 30; *y* de 0 a 30
- Variable de salida: z

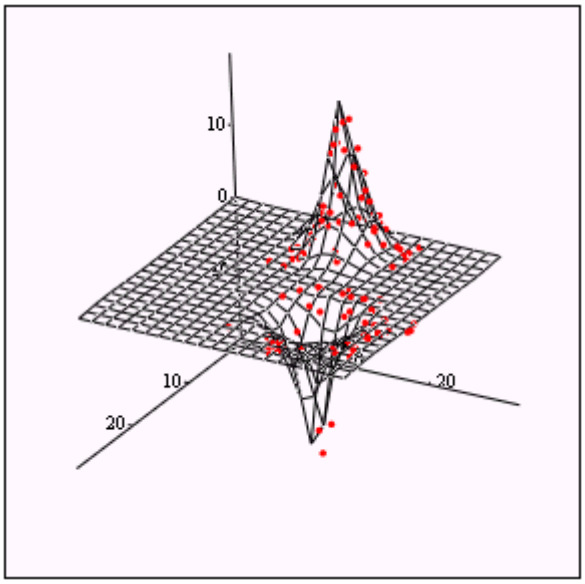


Figura 32. Modelo de dos picos Tridimensionales con los puntos refinados (rojo)

V.2.2. Colección Refinada

Fue estudiado el desempeño de las redes neuronales entrenadas con la colección refinada, relacionándolo con las áreas donde los nuevos puntos (refinados) fueron generados. También fue estudiada la influencia de los parámetros usados en el algoritmo de refinación (el número de vecinos y el valor umbral) en el desempeño de las RN. Éstos parámetros fueron estudiados, ya que son influyentes en el número de nuevos puntos a generar. Los estudios fueron realizados variando un parámetro a la vez; es decir, se mantiene el número de vecinos fijo, y se varía el valor umbral, luego se incrementa el número de vecinos, y se vuelve a variar el valor umbral. Los estudios fueron realizados con 4, 5 y 6 vecinos, y valores umbrales de 1; 1,5; 2 y 2,5. Hubo casos en los que el algoritmo no encontró regiones para refinar, por lo cual la variación del valor umbral se trunca hasta el último valor para el cual el algoritmo encontró regiones a refinar.

En todos los casos estudiados fue calculada la suma del cuadrado de los errores absolutos (SCEA) entre los valores pronosticados por la red neuronal (entrenadas tanto con la colección refinada como con la colección original expandida) y los valores reales.
Los valores numéricos se muestran en los anexos 1.

V.2.2.1. Modelo del Fermentador

Los resultados de éste modelo fueron obtenidos variando *F* (flujo de entrada) de 0,01 a 1,9, y S_f (la concentración de substrato de entrada) de 1 a 5; por lo tanto, se tiene el sistema mostrado en las ecuaciones 42 y 43.

$$S(F) = \frac{FK}{(\mu_{max} V) - F} \tag{42}$$

$$X(F, S_f) = \frac{(S_f - S)FY}{\mu_{max} V \left(\frac{S}{K + S} \right)} \tag{43}$$

las variables restantes fueron fijadas en sus valores nominales (fijadas constantes).
Las corridas fueron hechas usando los siguientes valores:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 40 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 40
- Número de vecinos usados para el cálculo del gradiente: 4, 5 y 6
- Valor umbral: 1; 1,5; 2 y 2,5
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: *F* de 0,01 a 1,9; S_f de 1 a 5
- Variable de salida: X, S
- Tipo y número de pronóstico: al azar con 60 series normales

El algoritmo no encontró regiones a refinar para valores umbrales mayores que 1,5.

Los resultados obtenidos, en cuanto a la suma del cuadrado de los errores absolutos (SCEA) para X y S, y el número de puntos generados por el algoritmo de refinación, son mostrados en las figuras 33, 34 y 35 cuando se seleccionaron 4 vecinos; en las figuras 36, 37 y 38 para 5 vecinos, y en las figuras 39, 40 y 41 para 6 vecinos.

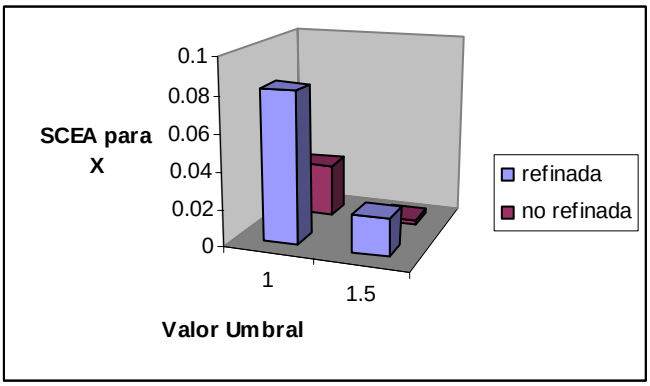


Figura 33. SCEA para X en el modelo del Fermentador. 4 vecinos

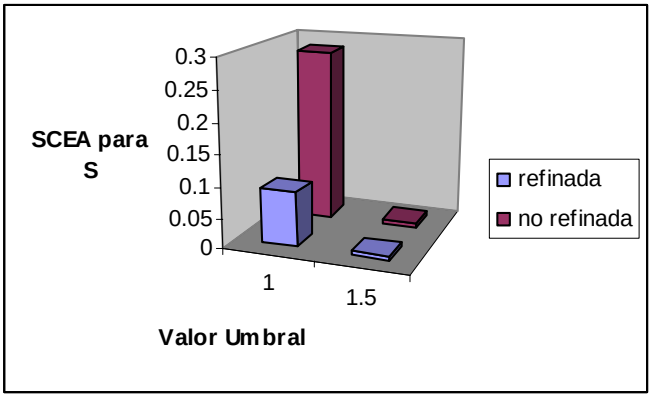


Figura 34. SCEA para S en el modelo del Fermentador. 4 vecinos

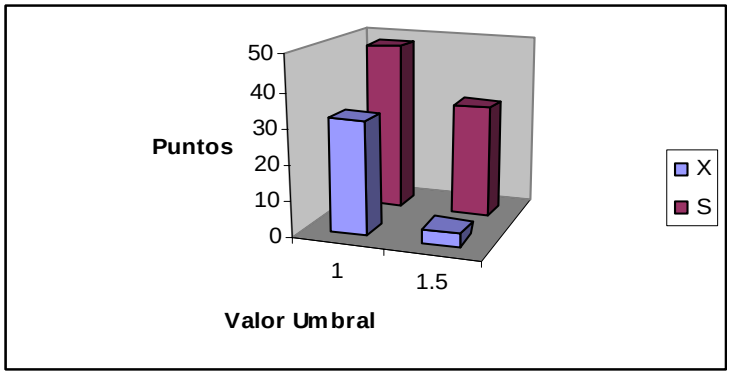


Figura 35. Puntos generados por el algoritmo de refinación en el modelo del Fermentador. 4 vecinos

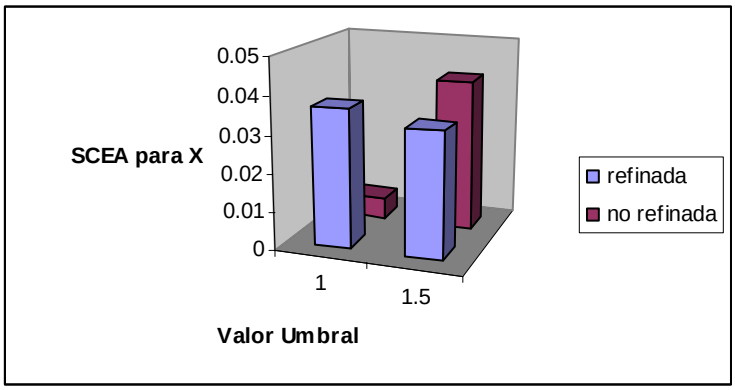


Figura 36. SCEA para X en el modelo del Fermentador. 5 vecinos

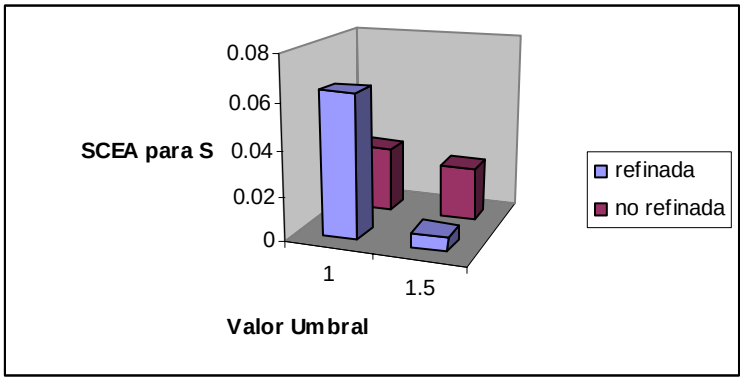


Figura 37. SCEA para S en el modelo del Fermentador. 5 vecinos

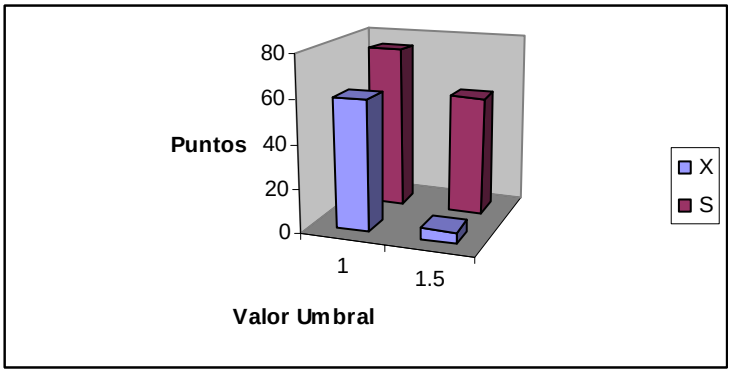


Figura 38. Puntos generados por el algoritmo de refinación en el modelo del Fermentador. 5 vecinos

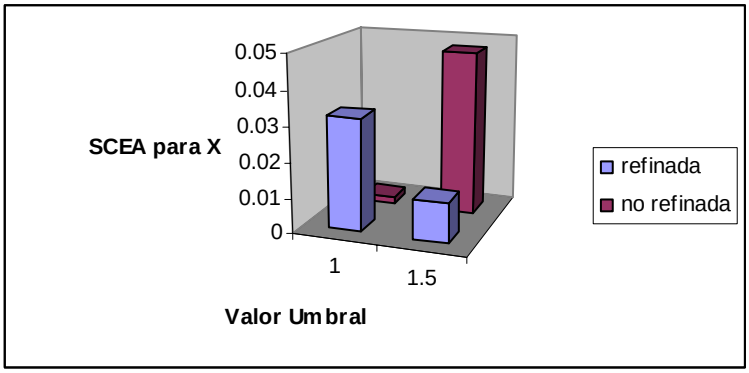


Figura 39. SCEA para X en el modelo del Fermentador. 6 vecinos

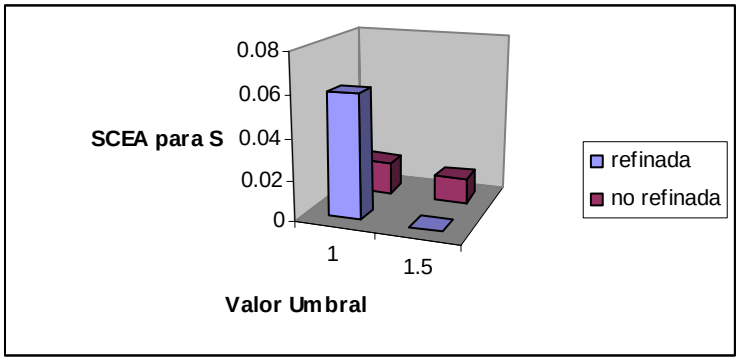


Figura 40. SCEA para S en el modelo del Fermentador. 6 vecinos

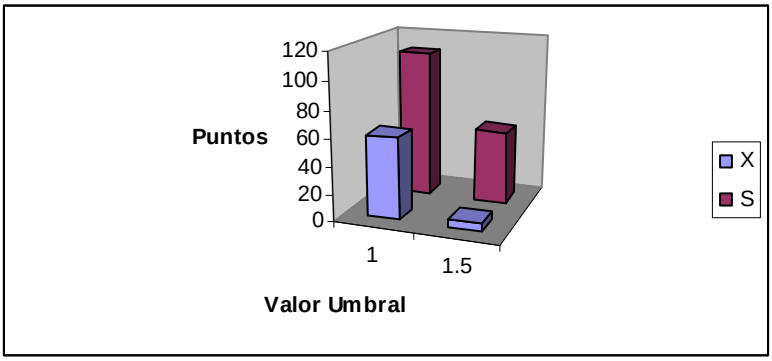


Figura 41. Puntos generados en el algoritmo de refinación en el modelo del Fermentador. 6 vecinos

En todos los casos, la suma del cuadrado de los errores absolutos (SCEA) para las colecciones refinadas disminuyen cuando el valor umbral aumenta.

En dos casos (5 y 6 vecinos para X) el SCEA para la colección no refinada aumenta.

El número de puntos generados siempre disminuyó al aumentar el valor umbral.

De 6 casos, 5 veces (83,3%) se obtuvo que la colección refinada tenía un mayor SCEA para el primer valor umbral (1), pero luego se obtuvo 5 veces (83,3%) menor error para el último valor umbral (1,5).

No se nota una variación del error cuando el número de vecinos cambia.

Las redes neuronales entrenadas con la colección refinada tuvieron 6 veces de 12 casos (50%) menor SCEA que las redes neuronales entrenadas con la colección original expandida.

V.2.2.2. Ecuación no lineal para el cálculo del tiempo de residencia

Tres sistemas fueron estudiados con esta ecuación, usando como entradas las siguientes variables, cuyos valores fueron cambiados durante el estudio:

1. μ_t y μ_r,
2. r_1 y r_2, y
3. ω y c

Los otros parámetros de la ecuación fueron fijados en sus límites inferiores. En cada sistema el gradiente fue calculado con 4, 5 y 6 vecinos, y valores umbrales de 1, 1,5, 2 y 2,5.

V.2.2.2.1. Cambiado μ_t y μ_r:

El valor de μ_t (coeficiente de fricción de la pulpa en la refinería) fue cambiado entre 0,5 y 1, y μ_r (coeficiente de fricción entre la madera mojada y el acero) fue cambiado entre 0,15 y 0,25. La ecuación es:

$$\tau(\mu_t, \mu_r) = \frac{\mu_r}{\mu_t} . \frac{a.E.c}{\varpi^3 . \left(r_2^{\,2} - r_1^{\,2}\right)} . \ln\left(\frac{r_2}{r_1}\right) \qquad (44)$$

La salida es el tiempo de residencia τ. Las otras variables fueron fijadas en sus límites inferiores (constantes).

Los siguientes parámetros fueron usados para las corridas:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 40 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 40
- Número de vecinos usados para el cálculo del gradiente: 4, 5 y 6
- Valor umbral: 1; 1,5; 2 y 2,5
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: μ_t de 0,5 a 1; μ_r de 0,15 a 0,25
- Variable de salida: τ
- Tipo y número de pronóstico: al azar con 60 series normales

El algoritmo encontró regiones a refinar sólo para valores umbrales de 1 en todos los casos (ver tabla 7).

Tabla 7. Resultados en los errores para la ecuación no lineal, entradas μt y μr

		Valores para τ
4 vecinos		
Valor Umbral		
1	SCEA de la RN entrenada con la colección refinada	1.29377E-05
	SCEA de la RN entrenada con la colección original expandida	1.79844E-06
	Puntos generados	72
5 vecinos		
1	SCEA de la RN entrenada con la colección refinada	1.08416E-06
	SCEA de la RN entrenada con la colección original expandida	0.026248562
	Puntos generados	100
6 vecinos		
1	SCEA de la RN entrenada con la colección refinada	4.91528E-05
	SCEA de la RN entrenada con la colección original expandida	3.70704E-06
	Puntos generados	78

Las redes neuronales entrenadas con las colecciones refinadas sólo obtuvieron en 1 caso de 6 estudiados (16.6%) menor SCEA que las entrenadas con la colección original expandida. No hubo una notable variación del error con el número de vecinos seleccionados.

V.2.2.2.2. Cambiando r_1 y r_2:

El valor de r_1 (radio interno) fue cambiado entre 0,08 y 0,12, y el valor de r_2 (radio externo) fue cambiado entre 0,15 y 0,45. La ecuación es:

$$\tau(r_1, r_2) = \frac{\mu_r}{\mu_t} \cdot \frac{a.E.c}{\varpi^3 . \left(r_2^{\,2} - r_1^{\,2}\right)} . \ln\left(\frac{r_2}{r_1}\right) \qquad (45)$$

La salida es el tiempo de residencia τ. Las demás variables fueron fijadas en sus respectivos límites inferiores (constantes).
Los parámetros usados para las corridas fueron:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 40 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 40
- Número de vecinos usados para el cálculo del gradiente: 4, 5 y 6
- Valor umbral: 1; 1,5; 2 y 2,5
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: r_1 de 0,08 a 0,12; r_2 de 0,15 a 0,45
- Variable de salida: τ
- Tipo y número de pronóstico: al azar con 40 series normales

Los resultados obtenidos, referentes a la suma del cuadrado de los errores absolutos y el número de puntos generados por el algoritmo de refinación, son mostrados en las figuras 42 y 43 cuando se seleccionaron 4 vecinos; en las figuras 44, 45 para 5 vecinos, y en las figuras 46 y 47 para 6 vecinos.

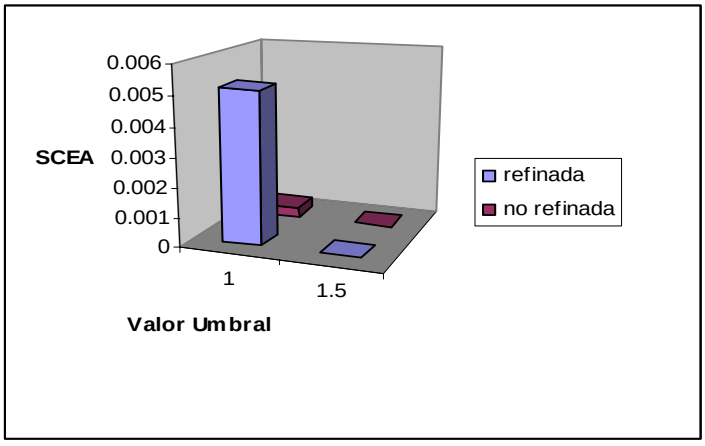


Figura 42. Suma del cuadrado de los errores absolutos para la ecuación no lineal. Entradas r1 y r2. 4 vecinos

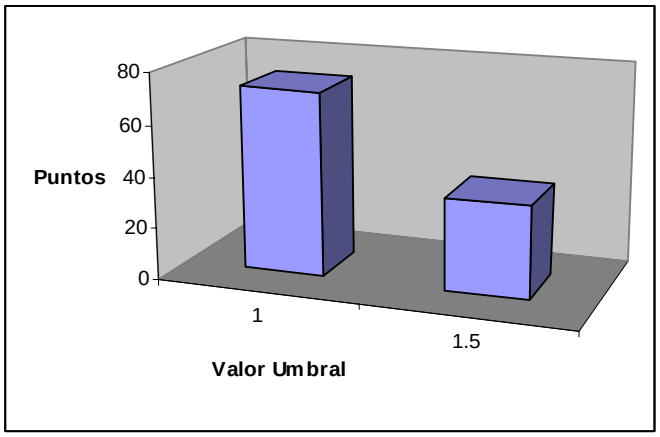


Figura 43. Número de puntos generados por el algoritmo de refinación para la ecuación no lineal. Entradas r1 y r2. 4 vecinos

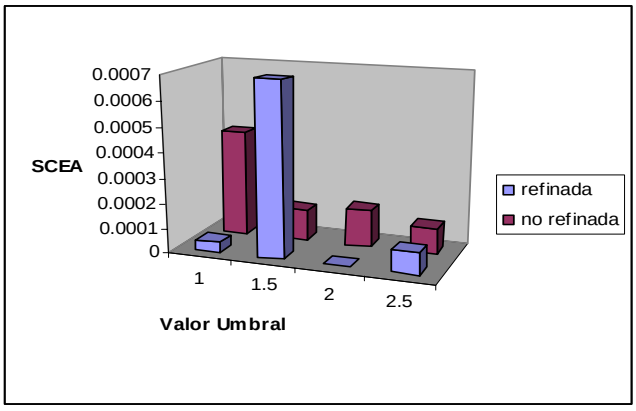


Figura 44. Suma del cuadrado de los errores absolutos para la ecuación no lineal. Entradas r1 y r2. 5 vecinos

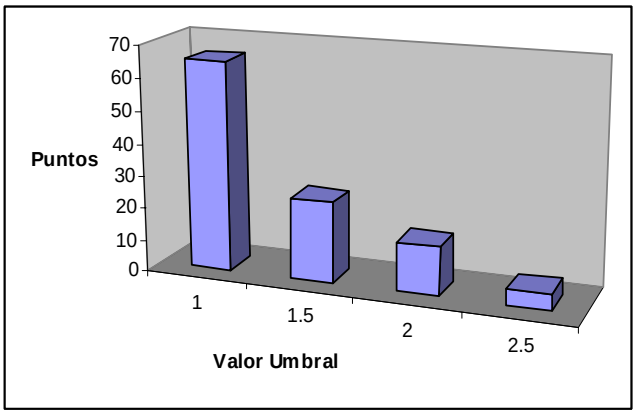


Figura 45. Número de puntos generados por el algoritmo de refinación para la ecuación no lineal. Entradas r1 y r2. 5 vecinos

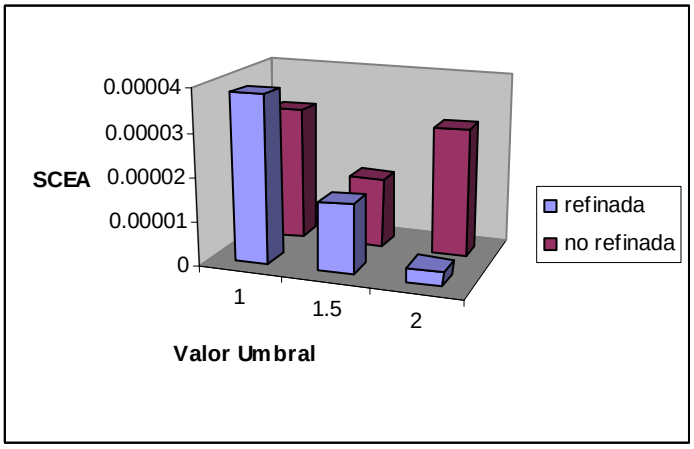


Figura 46. Suma del cuadrado de los errores absolutos para la ecuación no lineal. Entradas r1 y r2. 6 vecinos

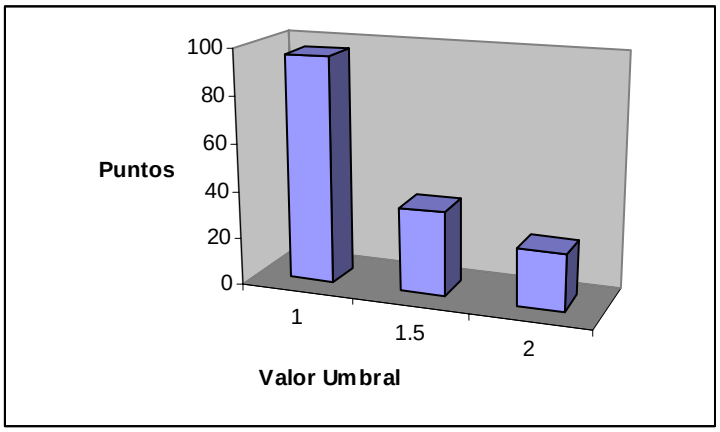


Figura 47. Número de puntos generados por el algoritmo de refinación para la ecuación no lineal. Entradas r1 y r2. 6 vecinos

De las figuras anteriores (42 a 47) se puede observar que el número de puntos generados por el algoritmo de refinación siempre disminuyen cuando el valor umbral es incrementado.

Por otro lado, el error de las redes neuronales entrenadas con los puntos refinados fueron en 5 casos de 9 estudiados (55,5%), menores que los

presentados por las redes neuronales estrenadas con la colección original expandida.

V.2.2.2.3. Cambiando ω y c

El valor de ω (velocidad angular) fue cambiado entre 150 y 350, y c (la consistencia de la refinación) fue variado entre 0,08 y 0,28. La ecuación es la mostrada en la número 46.

$$\tau(\omega, c) = \frac{\mu_r}{\mu_t} . \frac{a.E.c}{\omega^3 . \left(r_2^{\,2} - r_1^{\,2}\right)} . \ln\left(\frac{r_2}{r_1}\right) \qquad (46)$$

La salida es el tiempo de residencia τ. Las demás variables fueron fijadas es sus límites inferiores.
Los parámetros usados para las corridas fueron:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 40 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 40
- Número de vecinos usados para el cálculo del gradiente: 4, 5 y 6
- Valor umbral: 1; 1,5; 2 y 2,5
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: ω de 150 a 350; c de 0,08 a 0,28
- Variable de salida: τ
- Tipo y número de pronóstico: al azar con 60 series normales

Para el caso en que se emplearon 4 vecinos, las figuras 48 y 49 muestran la suma del cuadrado de los errores absolutos y el número de puntos generados por el algoritmo de refinación. Por el contrario, las figuras 50 y 51 muestran los

respectivos resultados obtenidos para el caso donde se consideraron 5 vecinos; y las figuras 52 y 53 para el caso de 6 vecinos.

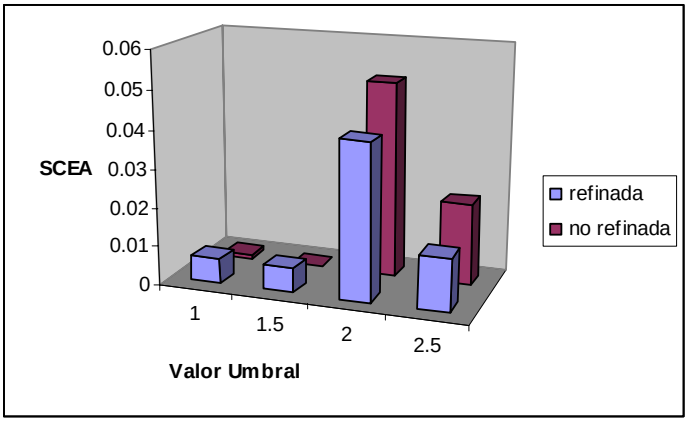


Figura 48. Suma del cuadrado de los errores absolutos para la ecuación no lineal. Entradas ω y c. 4 vecinos

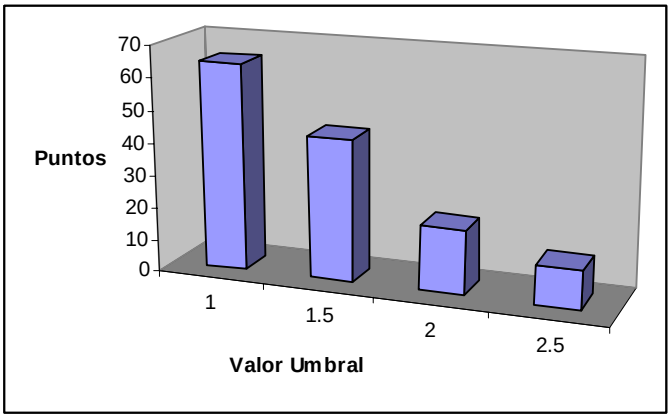


Figura 49. Número de puntos generados por el algoritmo de refinación para la ecuación no lineal. Entradas ω y c. 4 vecinos

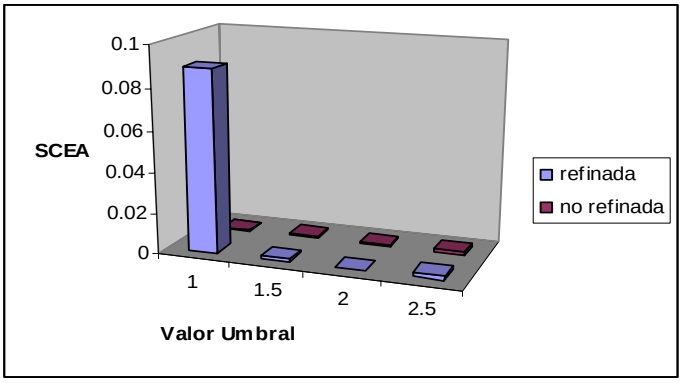


Figura 50. Suma del cuadrado de los errores absolutos para la ecuación no lineal. Entradas ω y c. 5 vecinos

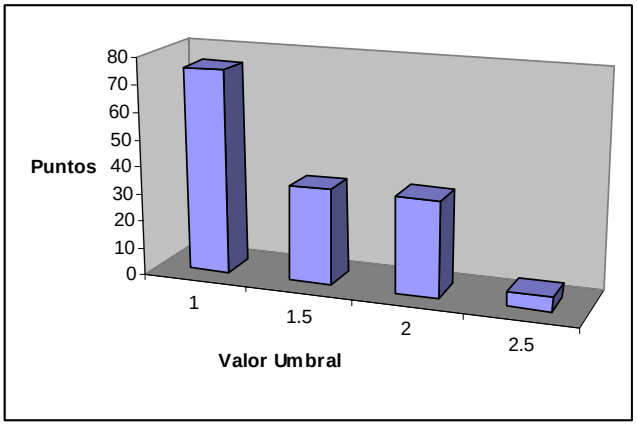


Figura 51. Número de puntos generados por el algoritmo de refinación para la ecuación no lineal. Entradas ω y c. 5 vecinos

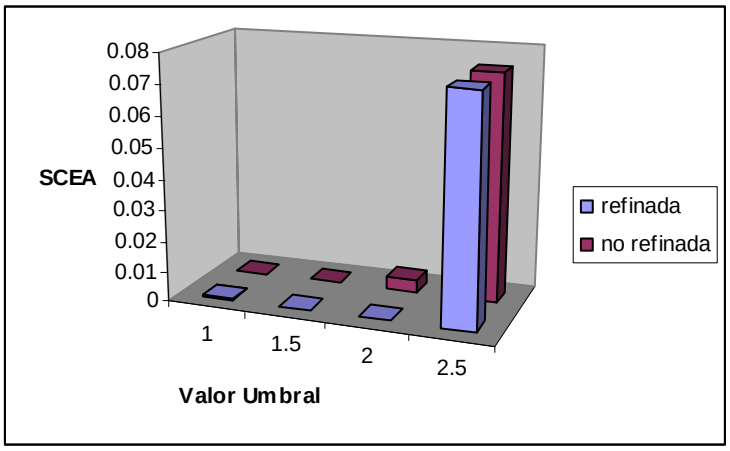


Figura 52. Suma del cuadrado de los errores absolutos para la ecuación no lineal. Entradas ω y c. 6 vecinos

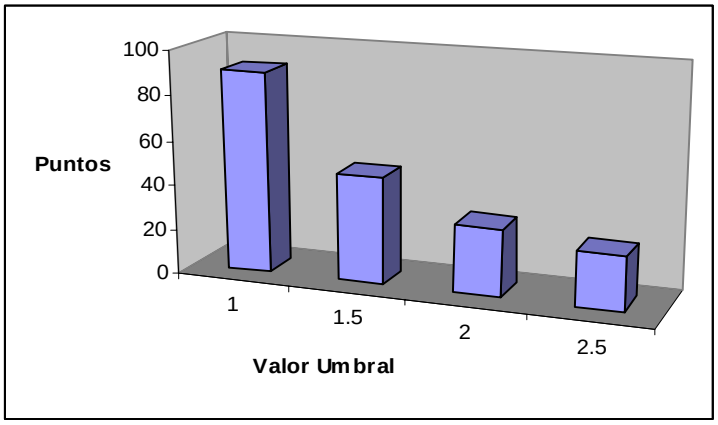


Figura 53. Número de puntos generados por el algoritmo de refinación para la ecuación no lineal. Entradas ω y c. 6 vecinos

En general, las redes neuronales entrenadas con la colección refinada dieron en 7 casos de 12 (58,33%) menor error que las redes neuronales entrenadas con la colección original expandida.
El número de puntos generados por el algoritmo de refinación decrece ó es el mismo, cuando el valor umbral es incrementado.

V.2.2.3. Ecuación de dos picos

Los resultados para este modelo fueron obtenidos cambiando las coordenadas (x e y) entre 0 y 30. Los valores de los demás parámetros son:
S_1 = 20, S_2 = -20 (pico invertido), a_x = 5, a_y = 15, b_x = 20, b_y = 20, σ_1 = 5, σ_2 = 5.
La salida es la coordenada z.
Los parámetros usados para las corridas fueron:

- Algoritmo de generación de datos: rejilla al azar
- Tipo y número de cálculos: normales con 50 cálculos
- Número de series seleccionadas para el cálculo del gradiente: 50
- Número de vecinos usados para el cálculo del gradiente: 4, 5 y 6
- Valor umbral: 1; 1,5; 2 y 2,5
- Nodos generados entre vecinos para refinar: 1
- Variables de entrada y rango de variación: *x* de 0 a 30; *y* de 0 a 30
- Variable de salida: z
- Tipo y número de pronóstico: al azar con 50 series normales

El gráfico de la ecuación se muestra en la figura 54. En ésta se muestra la variable de salida (z) en contra de las variables de entrada (x e y).

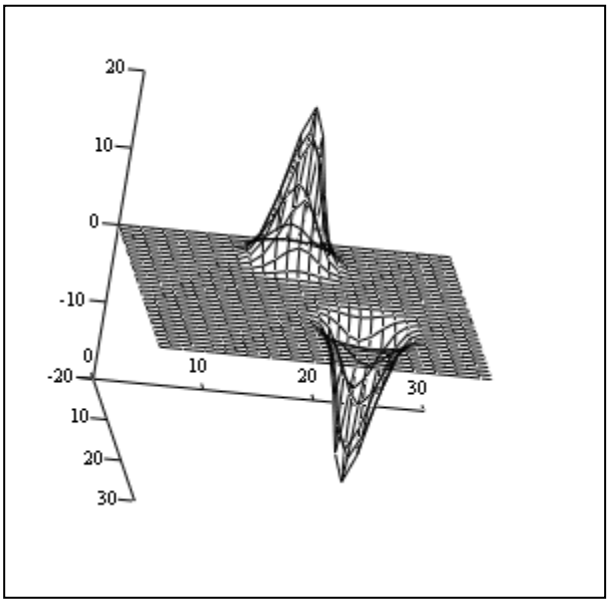


Figura 54. Ecuación de dos picos

Los resultados obtenidos en cuanto a la suma del cuadrado de los errores absolutos y el número de puntos generados por el algoritmo de refinación, son mostrados en las figuras 55 y 56 para el caso de considerar 4 vecinos; en las figuras 57 y 58 para 5 vecinos y en las figuras 59 y 60 para 6 vecinos.

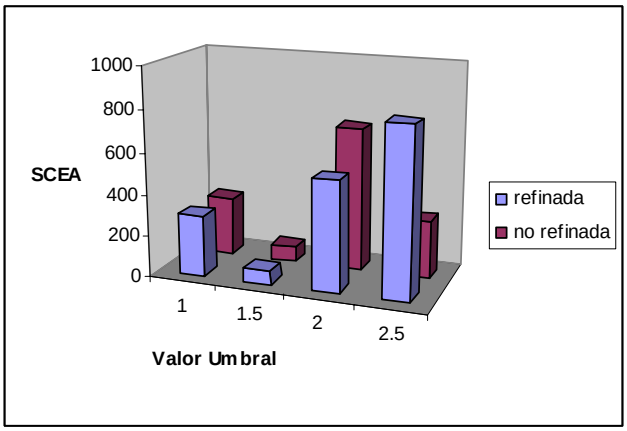


Figura 55. Suma del cuadrado de los errores absolutos para la ecuación de dos picos. 4 vecinos

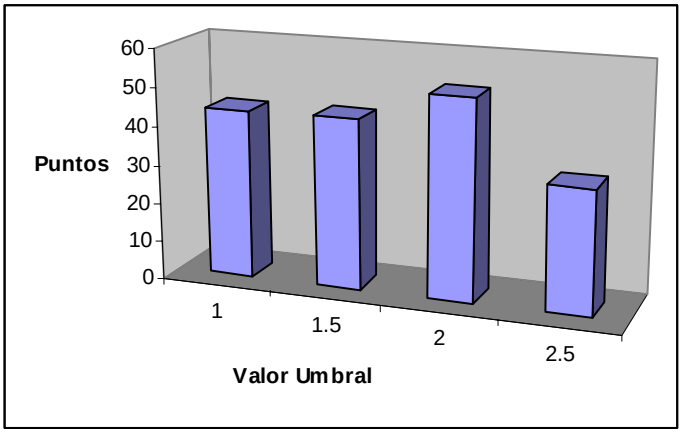


Figura 56. Número de puntos generados por el algoritmo de refinación para la ecuación de dos picos. 4 vecinos

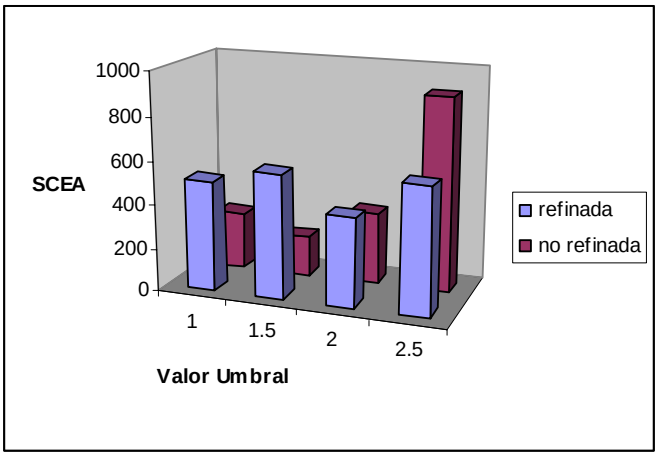


Figura 57. Suma del cuadrado de los errores absolutos para la ecuación de dos picos. 5 vecinos

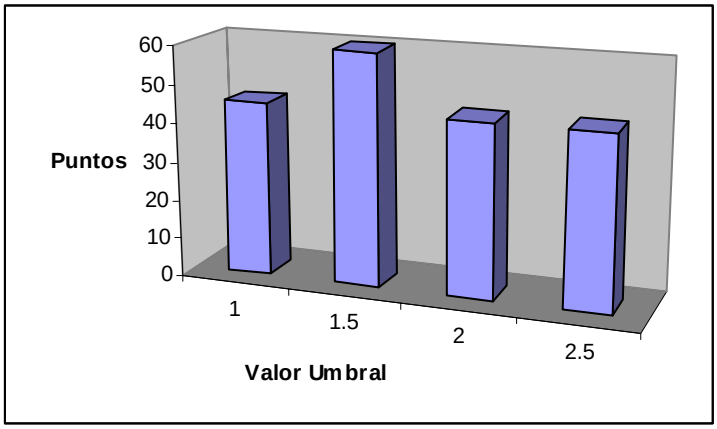


Figura 58. Número de puntos generados por el algoritmo de refinación para la ecuación de dos picos. 5 vecinos

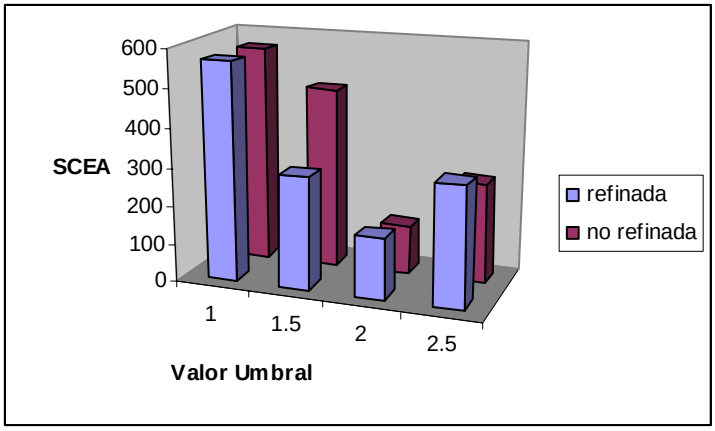


Figura 59. Suma del cuadrado de los errores absolutos para la ecuación de dos picos. 6 vecinos

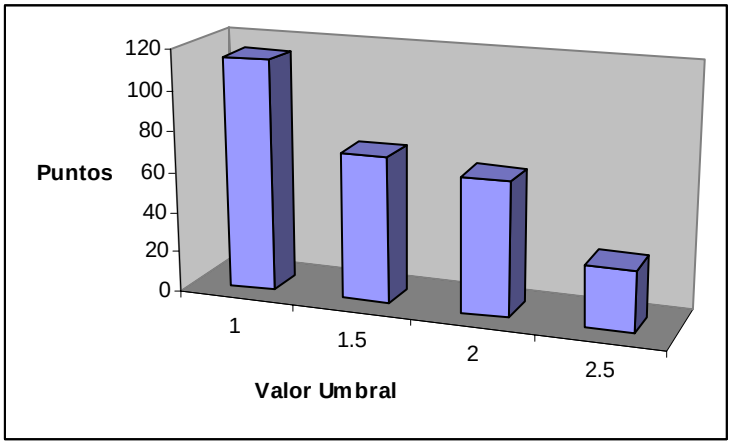


Figura 60. Número de puntos generados por el algoritmo de refinación para la ecuación de dos picos. 6 vecinos

En general, se observa que las redes neuronales entrenadas con la colección refinada dieron en 4 ocasiones de los 12 casos estudiados (33,3%) menor error que las redes neuronales entrenadas con la colección original expandida.

En resumen, haciendo un análisis de todos los casos estudiados (los tres modelos) se observa que las redes neuronales entrenadas con la colección refinada dieron en 23 ocasiones de los 51 casos (45%) menor error que las redes neuronales entrenadas con la colección original expandida.

V.2.3. Desviaciones en los pronósticos

Los pronósticos hechos por las redes neuronales siempre van a tener cierta desviación de los valores reales. La precisión de los pronósticos depende fundamentalmente de: la cantidad de datos usados para el entrenamiento, la configuración de la red neuronal, lo ruidoso que sean los datos a predecir, y el alejamiento de los datos a predecir de los valores usados para el entrenamiento.
El parámetro usado para medir la desviación debe ser escogido acorde con el sistema que se esté trabajando, y con lo que se desea lograr de éste. La suma del cuadrado de los errores absolutos fue usada previamente ya que lo que se pretendía era comparar el desempeño de dos redes neuronales entrenadas con series de datos diferentes, pero no indica desviación real de la data predicha. A continuación se presentan las tablas 8, 9, 10, 11 y 12, las cuales contienen los errores porcentuales de los datos predichos por las redes neuronales, para cada sistema estudiado.
En cada corrida hecha para el estudio, se genera una cantidad de cifras de errores, las cuales vienen dadas por el número de pronósticos que el usuario halla seleccionado. Las tablas 8, 9, 10, 11 y 12 contienen el error máximo, mínimo y el promedio de cada corrida. Éstos valores, proporcionan una idea del alejamiento de los pronósticos hechos por las redes neuronales, de los valores reales.

Tabla 8. Desviaciones en la data pronosticada por las RN en el modelo del Fermentador

Modelo del Fermentador				**RN entrenada con la colección refinada (%)**	**RN entrenada con la colección no refinada (%)**
4 vecinos	Valor Umbral	Salida	***Error porcentual***		
	1	x	máximo	28.36	24.01
			mínimo	0.02	0.00
			promedio	3.42	1.92

Continúa →

Continuación →

Modelo del Fermentador					
		s	máximo	858.32	83.80
			mínimo	0.61	0.05
			promedio	39.29	7.55
	1.5	x	máximo	6.12	3.43
			mínimo	0.00	0.02
			promedio	0.92	0.48
		s	máximo	141.08	10.46
			mínimo	0.05	0.01
			promedio	7.88	1.88
5 vecinos	1	x	máximo	9.22	4.34
			mínimo	0.02	0.00
			promedio	1.18	0.53
		s	máximo	343.63	16.51
			mínimo	0.09	0.01
			promedio	14.80	2.09
	1.5	x	máximo	42.96	45.01
			mínimo	0.01	0.00
			promedio	2.06	2.34
		s	máximo	171.72	35.82
			mínimo	0.09	0.02
			promedio	7.40	3.15
6 vecinos	1	x	máximo	14.85	2.39
			mínimo	0.01	0.00
			promedio	1.55	0.43
		s	máximo	497.62	181.35
			mínimo	0.04	0.12
			promedio	29.72	9.42
	1.5	x	máximo	7.44	19.17
			mínimo	0.02	0.04
			promedio	0.95	2.13
		s	máximo	61.86	141.75
			mínimo	0.02	0.50
			promedio	3.89	9.51

Tabla 9. Desviaciones en la data pronosticada por las RN en la ecuación no Lineal para μ_t y μ_r

Ecuación no Lineal para μt y μr				**RN entrenada con la colección refinada (%)**	**RN entrenada con la colección no refinada (%)**
4 vecinos	Valor Umbral	Salida	***Error porcentual***		
	1	τ	máximo	0.68	0.36
			mínimo	0.00	0.00
			promedio	0.14	0.04
5 vecinos	1	τ	máximo	0.23	58.57
			mínimo	0.00	0.00
			promedio	0.03	1.09
6 vecinos	1	τ	máximo	1.45	0.69
			mínimo	0.01	0.00
			promedio	0.24	0.06

Tabla 10. Desviaciones en la data pronosticada por las RN en la ecuación no Lineal para r_1 y r_2

Ecuación no Lineal para r1 y r2				**RN entrenada con la colección refinada (%)**	**RN entrenada con la colección no refinada (%)**
4 vecinos	Valor Umbral	Salida	***Error porcentual***		
	1	τ	máximo	24.32	6.76
			mínimo	0.01	0.01
			promedio	0.89	0.27
	1.5	τ	máximo	2.45	1.07
			mínimo	0.03	0.00
			promedio	0.37	0.10
5 vecinos	1	τ	máximo	1.78	7.57
			mínimo	0.00	0.01
			promedio	0.26	0.44
	1.5	τ	máximo	9.68	3.14
			mínimo	0.00	0.00
			promedio	0.38	0.27

Continúa →

Continuación →

Ecuación no Lineal para r1 y r2					
	2	τ	máximo	0.37	3.40
			mínimo	0.01	0.00
			promedio	0.08	0.29
	2.5	τ	máximo	2.74	4.87
			mínimo	0.01	0.07
			promedio	0.65	1.07
6 vecinos	1	τ	máximo	2.18	2.08
			mínimo	0.00	0.00
			promedio	0.30	0.11
	1.5	τ	máximo	6.96	2.08
			mínimo	0.00	0.00
			promedio	0.27	0.20
	2	τ	máximo	0.50	1.59
			mínimo	0.01	0.00
			promedio	0.18	0.17

Tabla 11. Desviaciones en la data pronosticada por las RN en la ecuación no Lineal para ω y c

Ecuación no Lineal para ω y c				**RN entrenada con la colección refinada (%)**	**RN entrenada con la colección no refinada (%)**
	Valor Umbral	Salida	***Error porcentual***		
4 vecinos	1	τ	máximo	10.76	2.50
			mínimo	0.01	0.01
			promedio	0.95	0.86
	1.5	τ	máximo	10.88	2.18
			mínimo	0.00	0.00
			promedio	0.80	0.30
	2	τ	máximo	20.95	20.73
			mínimo	0.00	0.01
			promedio	1.23	1.45
	2.5	τ	máximo	14.47	17.69
			mínimo	0.00	0.00
			promedio	0.75	0.59

Continúa →

Continuación →

Ecuación no Lineal para ω y c					
5 vecinos	1	τ	máximo	27.69	2.04
			mínimo	0.00	0.00
			promedio	1.69	0.58
	1.5	τ	máximo	4.76	5.68
			mínimo	0.03	0.01
			promedio	0.84	0.74
	2	τ	máximo	5.29	3.58
			mínimo	0.00	0.00
			promedio	0.66	0.60
	2.5	τ	máximo	6.01	5.23
			mínimo	0.04	0.00
			promedio	1.47	0.82
6 vecinos	1	τ	máximo	4.79	0.63
			mínimo	0.01	0.00
			promedio	0.66	0.18
	1.5	τ	máximo	2.32	0.61
			mínimo	0.00	0.00
			promedio	0.43	0.18
	2	τ	máximo	3.27	8.75
			mínimo	0.00	0.00
			promedio	0.70	1.22
	2.5	τ	máximo	28.72	28.46
			mínimo	0.01	0.01
			promedio	1.37	1.23

En general, los errores porcentuales ponderados de los promedios son:

- Modelo del Fermentador: 1,68% para X, y 17,17% para S con las colecciones refinadas; 1,31% para X, y 5,6 para S con las colecciones no refinadas.
- Ecuación no Lineal:
  - μ_t y μ_r:0,14 con las colecciones refinadas, y 0,4 con las no refinadas.
  - r_1 y r_2: 0,38% con las refinadas, y 0,32% con las no refinadas.
  - ω y c: 0,96% con las refinadas, y 0,73% con las no refinadas.
- Ecuación de dos picos: la mayor correlación fue de 0,97 con las colecciones refinadas, y de 0,91 con las no refinadas

Tabla 12. Desviaciones en la data pronosticada por las RN en la ecuación de dos Picos

Ecuación de dos picos				**RN entrenada con la colección refinada**	**RN entrenada con la colección no refinada**
	Valor Umbral	Salida	***Error***		
4 vecinos	1	τ	correlación	0.29	0.07
			error relativo	1.46	1.17
	1.5	τ	correlación	0.95	0.20
			error relativo	4.90	3.96
	2	τ	correlación	0.91	0.39
			error relativo	4.61	6.12
	2.5	τ	correlación	0.97	0.72
			error relativo	6.04	3.55
5 vecinos	1	τ	correlación	0.88	0.91
			error relativo	7.22	4.10
	1.5	τ	correlación	0.83	0.26
			error relativo	4.85	3.35
	2	τ	correlación	0.95	0.26
			error relativo	6.40	2.32
	2.5	τ	correlación	0.69	0.29
			error relativo	8.05	1.94
6 vecinos	1	τ	correlación	0.87	0.50
			error relativo	6.99	0.67
	1.5	τ	correlación	0.71	0.41
			error relativo	2.73	1.43
	2	τ	correlación	0.85	0.30
			error relativo	4.45	0.86
	2.5	τ	correlación	0.75	0.19
			error relativo	2.87	0.78

En la tabla 12 se usó la correlación y el error relativo, debido principalmente a que la ecuación de dos picos es suave en la región de z=0 (ver figura 54), por lo cual los cálculos en el error porcentual arrojaban valores no acordes (muy

grandes). Es importante resaltar que éstos valores fueron calculados usando los datos separados de las colecciones de entrenamiento para la validación (test).

V.2.4. Errores en las zonas de alto gradiente

Con el objeto de comparar el desempeño de las redes neuronales entrenadas con las diferentes colecciones en las zonas de alto gradiente, se realizó un pronóstico sólo en zonas donde el gradiente era alto. El modelo usado fue la ecuación de dos picos, y los pronósticos fueron hechos en los picos mismos (ver tabla 13).

En la tabla 13 se observa, que de los 12 casos estudiados, en 9 ocasiones (75%) las redes neuronales entrenadas con la colección refinada dieron menor error (SCEA) que las redes neuronales entrenadas con la colección original expandida.

Tabla 13. SCEA en regiones de alto gradiente

Ecuación de dos picos			**RN entrenada con colección refinada**	**RN entrenada con colección no refinada**
	Valor Umbral	***Error***		
4 vecinos	1	SCEA	2835.32	2804.00
		Indicación	1% mayor	
	1.5	SCEA	724.98	2771.99
		Indicación	73.84% menor	
	2	SCEA	785.08	2679.14
		Indicación	70.7% menor	
	2.5	SCEA	1133.35	1059.27
		Indicación	6.98% mayor	
5 vecinos	1	SCEA	1661.90	1617.45
		Indicación	2.78% mayor	
	1.5	SCEA	1670.21	2758.97
		Indicación	39.47% menor	
→	2	SCEA	1769.59	2281.75
		Indicación	22.44% menor	→

	2.5	SCEA	1412.13	2787.66
		Indicación	49.33% menor	
6 vecinos	1	SCEA	1036.32	2682.55
		Indicación	61.37% menor	
	1.5	SCEA	1774.01	2718.53
		Indicación	34.73% menor	
	2	SCEA	1490.42	2777.28
		Indicación	46.34% menor	
	2.5	SCEA	883.37	2807.51
		Indicación	68.54% menor	

VI. Discusión de Resultados

A continuación se presenta la discusión de los resultados obtenidos, la cual se realizará en concordancia con la manera que se presentaron los resultados, es decir, en base al software elaborado y los modelos estudiados.

VI.1. Resultados en el Software

Con las nuevas técnicas de programación, y la exposición de las interfaces por parte de los programas de aplicación en Windows® como los usados en el presente trabajo de grado: Aspen Custom Modeler®, Excel® y NN Tool 2000®, se abre un nuevo campo de aplicaciones de los mismos. Por ejemplo, la creación de una colección de datos de una simulación hecha en Aspen Custom Modeler®, y luego la presentación de los resultados en Excel®; así como la creación automática de redes neuronales usando NN Tool 2000®, y luego la utilización de las mismas en Excel® prescindiendo del mismo programa que las creó.

Las ventajas de tal tipo de programación son inmensas, pero la principal es el factor tiempo. Por ejemplo, con el uso de *SimWrap* se logra hacer en minutos lo que hace un lustro se hacía en días. También la obtención de resultados es mucho más rápida, por lo cual se pueden hacer un número grande de investigaciones en un tiempo corto.

Con la utilización de las redes neuronales obtenidas con *SimWrap* se logra alcanzar resultados precisos del modelo trabajado, prescindiendo del simulador y de sus largos tiempos de generación de resultados, abriendo la posibilidad de su uso en línea. Es posible por ejemplo, que al generar con *SimWrap* una red neuronal de una simulación dada; se podría usar un dispositivo especial para la introducción de los valores reales de los parámetros del proceso (medidos en el campo) a la red neuronal, y obtener al instante el valor predicho de las salidas del proceso. Si se nota una gran variación entre los valores predichos por la red neuronal, y los valores medidos en el campo (salidas), se puede saber de antemano que algo podría estar pasando en el proceso real.

Con el uso en Excel de las redes neuronales creadas, se presenta un sistema fácil de obtención de pronósticos de simulaciones, el cual puede ser usado en planta por personas que no tengan conocimientos previos en el uso de simuladores (como técnicos y peritos por ejemplo).

También con el uso de las redes neuronales creadas, se logra obtener resultados donde el modelo falla (causado por detalles matemáticos por ejemplo).

El programa creado se presenta de uso amigable y fácil de usar, permitiendo la interacción con el usuario. Las posibilidades de aplicación son grandes, y se logra hacer gran cantidad de estudios en un corto tiempo.

Adicionalmente, la estructura amigable del programa permite virtualmente a cualquier persona crear redes neuronales artificiales, sin tener un conocimiento profundo del funcionamiento de las mismas.

VI.2. Resultados en los modelos

A continuación se presentan los resultados en los modelos trabajados, según los siguientes aspectos: generación de puntos en zonas de alto gradiente, y el pronóstico realizado.

VI.2.1. Generación de puntos en zonas de alto gradiente

Ya que el refinamiento es hecho usando la colección original como espacio de búsqueda, éste estará confinado en ésta área. Por lo tanto, el número de cálculos con el algoritmo de rejilla al azar, debe ser seleccionado pensando en el grado de detalle que se requiere para la búsqueda. El algoritmo de refinación sólo busca las áreas de alto gradiente contenidas en la colección original. Es por supuesto posible que los puntos generados en el algoritmo de refinación tengan un valor de gradiente mucho más alto que los puntos en la colección original; pero el algoritmo no calcula los valores del gradiente para éstos nuevos puntos, por lo tanto ésta rama está fuera del alcance de este estudio.

El número de puntos seleccionados de la colección original para el cálculo del gradiente es un compromiso entre el detalle de la búsqueda de las regiones de alto gradiente que están contenidas en la colección original, y la rapidez de la búsqueda. Los sistemas estudiados fueron rápidos de analizar (menos de 5 segundos en un Pentium® III), ya que la colección original estaba constituida de pocas series (40 y 50 puntos en el espacio).

La selección de puntos para el cálculo del gradiente aumenta en importancia cuando el espacio de búsqueda (colección original) aumenta en dimensión. También, en cuanto mayor es el número de valores de gradiente que existen, mayor será el número de puntos que cumplirán con el criterio; por lo tanto habrán más puntos generados.

Incrementar el número de vecinos para el cálculo del gradiente conduce a mejores resultados en el valor del gradiente, y esto conduce a una mejora en la selección de puntos en las zonas de realmente alto gradiente. Un mal resultado en el gradiente, puede conducir a la selección de puntos en áreas donde el gradiente no es tan alto como se espera.

Una desventaja de aumentar el número de vecinos para el cálculo del gradiente es en la resolución del sistema de ecuaciones por Gauss Jordan, debido a posibles inestabilidades, ya que habría muchas más ecuaciones que incógnitas, y puede llevar a malos resultados. Por lo tanto la selección del número de vecinos para el cálculo del gradiente es un factor determinante en el resultado del gradiente.

Por otro lado, debido a que los puntos generados en las zonas de alto gradiente son producidos al azar entre los vecinos del punto seleccionado (aquel que cumpla el criterio de refinamiento), el número de vecinos también influye en el número de puntos generados. Si los vecinos se encuentran muy cerca (ya que por ejemplo, fue usada una colección original muy detallada, ó el espacio de búsqueda fue muy corto) esto puede causar un “apiñamiento” en los puntos generados, ya que los puntos generados también se ubican muy cerca, y no se esparcen a lo largo del área de alto gradiente. Este comportamiento puede verse en las gráficas A1, A9, A14, A37, A42 y A43.

El apiñamiento también puede presentarse cuando se seleccionan muchos puntos a ser generados entre los límites.

El valor umbral controla cuán alto debe ser el gradiente en el área de refinamiento comparado con los valores del gradiente en el resto de los puntos. Esto representa que el usuario puede seleccionar en qué áreas los puntos son especialmente seleccionados (áreas en términos de gradiente).

Ésta es la razón por la cual los valores del gradiente se presentan en la corrida, y por lo cual el usuario tiene la capacidad de cambiar el valor umbral una vez conocidos los valores del gradiente y el valor promedio.

Por ejemplo, consideremos que el gráfico que se muestra en la figura 61 es la cuenta en cajas de los valores del gradiente de una superficie ficticia.

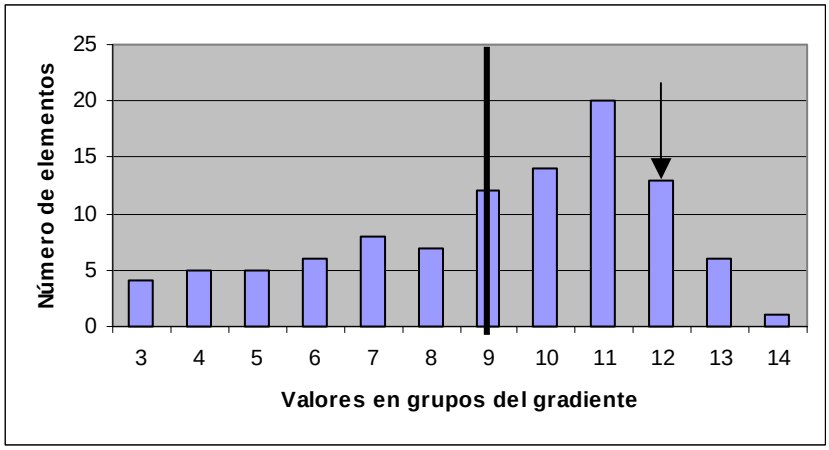


Figura 61 Cuenta en cajas

La línea gruesa indica el valor promedio de los valores del gradiente, y la flecha indica el valor umbral seleccionado por el usuario; todos los puntos a la derecha de la flecha van a ser refinados.

El usuario puede controlar la posición de ésta flecha modificando el valor umbral. Si el valor umbral es menor que 1, la flecha va a estar al lado izquierdo del valor medio (línea gruesa); si el valor umbral es mayor que 1, la flecha va a estar al lado derecho del valor medio. Por lo tanto, el usuario controla con el valor umbral cuán "fuerte" el gradiente de un punto debe ser para que éste sea seleccionado.

Por ésta razón es que en la corrida se muestran los valores del gradiente, al igual que el valor promedio, antes de hacerse el refinamiento, y se da la posibilidad al usuario de cambiar el valor umbral. Con esto, el usuario tiene una percepción de los valores de gradiente manejados, y le permite de alguna manera, seleccionar las zonas donde la refinación se va a llevar a cabo.

VI.2.2. Resultados en los pronósticos

Debido a que los puntos generados son seleccionados en zonas de alto gradiente, los valores pronosticados por las RN entrenadas con la colección refinada obtuvieron en el 75% de los casos menor error que los pronosticados por las RN entrenadas con la colección original expandida cuando los pronósticos son realizados en las áreas de alto gradiente o sus cercanías. En éste 75% de los casos, las redes neuronales entrenadas con la colección refinada dieron muchos mejores resultados que sus contrapartes (dieron menor SCEA); contrariamente, cuando las redes neuronales entrenadas con las colecciones no refinadas presentaron menor error en las áreas de alto gradiente, la diferencia entre su SCEA y el de sus contrapartes entrenadas con las colecciones refinadas era pequeño (ver tabla 12).

Cuando los pronósticos son realizados en zonas suaves de la superficie, las RN entrenadas con la colección original expandida dieron en el 55% menor error que las contrapartes entrenadas con la colección refinada, lo que indica que tienen la tendencia de producir menor error; esto ocurre ya que los puntos al azar generados para expandir la colección original son producidos a lo largo de toda la superficie, y no sólo concentrados en las zonas de alto gradiente.

En general, los errores obtenidos por las redes neuronales para todos los modelos son bastante bajos (ver tablas 7, 8, 9, 10 y 11), lo que indica que las redes neuronales artificiales son un buen método para representar modelos básicos de la ingeniería química.

VII. Conclusiones

Como resultado de los estudios, se puede concluir,

- Las RN entrenadas con la colección refinada tuvieron en el 45% de los casos menor SCEA que las RN entrenadas con la colección original expandida, lo que indica que producen mejores resultados las RN entrenadas con la colección original expandida.
- Las RN entrenadas con la colección refinada tuvieron en el 75% de los casos menor SCEA que las RN entrenadas con la colección original expandida, cuando los pronósticos son realizados en las zonas de alto gradiente. Lo que indica que las RN entrenadas con la colección refinada producen mejores resultados que sus contrapartes entrenadas con la colección original expandida cuando el pronóstico se hace en regiones de alto gradiente.
- El desempeño de las RN entrenadas con la colección refinada no siempre mejora cuando aumentan el número de series para el entrenamiento.
- El apiñamiento en los puntos generados por el algoritmo de refinación causa que la RN entrenada con la colección refinada aumente el error.
- Las redes neuronales artificiales dan una buena representación de simulaciones simples, incluso con números pequeños de series para el entrenamiento.
- El algoritmo de refinación genera puntos en zonas donde el gradiente de la superficie estudiada es mayor que un valor crítico introducido por el usuario.
- *SimWrap* se presenta como un programa amigable y fácil de usar, para crear colecciones de datos, redes neuronales y hacer pronósticos con éstas.

VIII. Recomendaciones

Es interesante cómo las Redes Neuronales entrenadas con la colección refinada obtienen menos error que las Redes Neuronales entrenadas con la colección original expandida, cuando los pronósticos son realizados en zonas de alto gradiente. Sin embargo, los pronósticos en todo el campo de estudio pueden ser mejorados ampliamente usando una combinación de ambas redes neuronales. Puede hacerse un algoritmo, el cual halle el punto más cercano entre ambas colecciones (puntos refinados y puntos al azar para expandir la colección original), y si el punto más cercano pertenece a la colección refinada, se usa la RN entrenada con la colección refinada; y si el punto más cercano pertenece a la colección expandida, usar la RN entrenada con la colección original expandida.

También podrían hacerse más corridas de refinación. El algoritmo puede generar algunos puntos en las zonas de alto gradiente en la primera corrida, y luego hacer el número de posteriores corridas de refinación que el usuario quiera, tomando en cuenta en éstos nuevos refinamientos los puntos nuevos refinados de las corridas previas.

Finalmente, sería interesante probar el desempeño de éstas redes neuronales combinadas en simulaciones reales de la ingeniería, ó en sistemas de más de 3 dimensiones.

IX. Bibliografía

IX.1. Bibliografía Citada

1) Aspen Custom Modeler® help topics.
2) Aspen Custom Modeler® user's guide.
3) Ayyub, B., McCuen, R.; "Probability, Statistics, & Reliability for Engineers". CRC Press. USA 1997.
4) Biegler, L., Grossman, I., Westerberg, A.; "Systematic Methods of Chemical Process Design"; Prentice Hall; New Jersey 1997.
5) Demuth, H., Beale, M.; "Neural Network Toolbox. User's guide". The Math Works Inc. USA. 1992
6) D'Souza, David; Whalen, BJ and Wilson, Peter; "Implementing Side-by-Side Sharing in Applications". MSDN, Microsoft Corporation. November 1999.
7) Fraser, Neil; Carleton University Web Site, 1998.
8) Goldberg, David; "Genetic Algorithms in Search, Optimization and Machine Learning"; Addison Wesley Pub Co.; 1989.
9) Hebb, D.O; "The organization of Behavior: A Neurophysiological Theory"; Wiley, New York; 1949.
10) Ingham, J., Dunn, I.J., Heinzle, E., Prenosil, J.E.; "Chemical Engineering Dynamics, An introducction to Modeling and Computer Simulation"; Wiley-VCH; Second Edition, Germany 2000.
11) Kuhlmann, C.; "Optimization of Chemical Engineering Processes Using Computational Techniques"; Doctoral Thesis, London University, 1997.
12) "Microsoft Excel Language Reference". Microsoft Press. Redmon, Washington. 1997.
13) "Microsoft Visual Basic 6.0 Programmer´s Guide". Microsoft Press. Redmon, Washington. 1998.
14) Miles, K.B. (1991). "A simplified method for calculating the residence time and refining intensity in a chip refiner". Paperi Ja Puu, 73(9), p. 852 – 857.

15) Müller, B.; Reinhardt, J.; "Neural Networks, An Introduction"; Springer-Verlag, Germany, 1990.

16) NN Tool 2000® Handbuch. Bärmann®, 2000.

17) Pattison, Ted. "Understanding Interface-based Programming". Microsoft Corporation, January 1999. MSDN.

18) Press, W.; Vetterling, W.; Teukolsky, S.; Flannery, B.; "Numerical Recipes in C". Second Edition, Cambridge University Press. Reprinted 1999.

19) Quantrille, T., Liu, Y.A.; "Artificial Intelligence in Chemical Engineering"; Academic Press; San Diego, California 1991.

20) Rogerson, Dale; "Calling COM Objects with interface Wrappers". MSDN: Microsoft Developer Network Technology Group. October, 1995.

21) Sarma, Debabrata; "Dll for Beginners". MSDN: Microsoft Development Support. November 1996.

22) Smith, Leslie; "An introduction to Neural Networks"; talk given at the NSYN meeting in Edinburgh, Scotland; 1996.

23) www.NeuroDimension.com, 1997.

24) Zurada, Jacek.; "Introduction to Artificial Neural Systems"; West Publishing Company; USA, 1992.

IX.2. Bibliografía Consultada

1) Lanoulette, R., Thibault, J., Valade, J.; "Process modeling with neural networks using small experimental datasets"; Computers and Chemical Engineering 23 (1999) 1167 – 1176.

2) www.mathsoft.com

X. Glosario

ACM	Aspen Custom Modeler®
Aspen Custom Modeler®	Simulador usado en el presente trabajo
Colección	Matriz de series de datos
Export name	Nombre de exportación en inglés
Fixed	Valores fijos o constantes
File	Archivo en inglés
Gradient	Gradiente en inglés
Homotopía	Estrategia usada en puntos de no convergencia
Initialization sheet	Hoja de inicialización
Input	Entrada en inglés
Lower bound	Límite inferior en inglés
Neural Networks	Redes neuronales en inglés
NN Tool 2000®	Programa usado en el presente trabajo para generar las redes neuronales artificiales
Nodos	(ver puntos)
Output	Salida en inglés
Prognosis	Pronóstico en inglés
Puntos	Series de valores
Refinamiento	Generación de nuevas series de datos en zonas de alto gradiente
RN	Redes neuronales
RNA	Redes neuronales artificiales
SCEA	Suma del cuadrado de los errores absolutos
Simulation name	Nombre de la simulación en inglés
SimWrap	Nombre del programa desarrollado en el presente trabajo
Upper bound	Límite superior en inglés
Values	Valores en inglés