ANEXO A

DESCRIPCIÓN DE LAS FUNCIONES UTILIZADAS EN MATLAB

1. Red tipo Perceptrón: Las siguientes son las herramientas de redes neuronales del Matlab 5.3: utilizadas en el entrenamiento de las redes neuronales correspondientes a las aplicaciones del proceso de detección de obstáculos de un robot y Control de cambio de giro de un motor trifásico.

- *newp*: crea una red tipo Perceptrón, que requiere las siguientes entradas:

  NET = NEWP(PR,S,TF,LF)

  PR : Rx2 matriz de valores máximos y mínimos para los R elementos de entrada.

  S : Número de neuronas.

  TF : Función de Transferencia, en este caso 'hardlims'.

  LF : Función de aprendizaje, para este caso 'learnp'.

- *rands*: Función simétrica que inicializa aleatoriamente los valores de pesos y ganancias de una red con valores entre -1 y 1; requiere de la estructura rands(S,PR), generando una matriz de dimensiones S x PR.

- *adapt:* Permite a una red neuronal adaptarse a los patrones de entrada, esta función tiene la siguiente sintaxis:

  [net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)

  net : Red que va a crearse

  P : Entradas a la red; deben aparecer en forma de un arreglo de matrices.

  T : Salidas esperadas de la red, si no se especifican son ceros por defecto.

  Pi : Condiciones de retardo para la entrada inicial, por defecto son ceros.

  Ai : Condiciones de retardo para la capa inicial, por defecto ceros

- *net.adaptParam.passes*: Número de iteraciones que utiliza el programa.

2. **Red tipo Adaline:** Las siguientes son las funciones de las herramientas de Redes Neuronales del Matlab utilizadas en el entrenamiento del filtro adaptivo diseñado con base en una red Adaline.

- *newlin:* Función para crea una red tipo Adaline, que requiere las siguientes entradas:

  NEWLIN(PR,S,ID,LR)

  R: Matriz de Rx2 que contiene los valores máximos y mínimos de cada uno de los R elementos de entrada.

. S : Número de neuronas

ID : Arreglo que contiene los valores de los retardos, por defecto todos sus valores son cero.

LR : Rata de aprendizaje, por defecto = 0.01

- *net.inputWeights{1,1}.delays:* Especifica los retardos iniciales
- *net.adaptParam.passes*: Número máximo de iteraciones
- *[net,y,E,pf,af]=adapt(net,p,T,pi):* Comando de entrenamiento de la red; requiere como entradas la red creada anteriormente, los patrones de entrada, las salidas esperadas y los retardos iniciales, retorna el estado final de la red, los valores obtenidos para cada patrón de entrada con sus correspondientes errores así como los valores finales de los retardos.

3. Red tipo Backpropagation: La red neuronal Backpropagation presenta una gran variedad de opciones de configuración, dependiendo de la necesidad de aprendizaje y de la aplicación que se esté desarrollando.

- *newff:* Crea una red tipo Backpropagation, requiere que le sean especificados los siguientes parámetros

  newff: (PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

  PR : Rx2 Matriz de valores máximos y mínimos de cada uno de las R neuronas de entrada.

  Si : Número de neuronas para cada una de las capas.

TFi : Función de transferencia a utilizar en cada una de las capas, por defecto utiliza *tansig*

BTF : Algoritmo de entrenamiento a utilizar, por defecto utiliza *trainlm*

BLF : Función de actualización de los pesos, por defecto utiliza *learngdm*.

PF : Función para evaluar el desempeño de la red, por defecto utiliza *mse*.

Los siguientes fueron los algoritmos de entrenamiento que se utilizaron en el ejemplo de control de voltaje por inyección de reactivos en una barra remota y en la aplicación de predicción de consumo de carga durante sus respectivos procesos de aprendizaje hasta que se encontró uno que brindara un aprendizaje óptimo, para cada uno de ellos utilizando la red Backpropagation:

3.1 Traingd: Algoritmo de pasos descendientes, que actualiza pesos y ganancias variándolos en la dirección negativa del gradiente de la función del error. Es un algoritmo de aprendizaje muy lento, que requiere de la siguiente sintaxis:

- *net.trainParam.epochs:* Máximo número de iteraciones para obtener convergencia
- *net.trainParam.goal*: Error máximo permitido
- *net.trainParam.lr:* Rata de aprendizaje
- *net.trainParam.max_fail:* Máximo número de fallas
- *net.trainParam.min_grad:* Mínimo rendimiento del gradiente
- *net.trainParam.show:* Intervalo de visualización de los resultados
- *net.trainParam.time:* Máximo tiempo de entrenamiento en segundos

Con este algoritmo el aprendizaje de la red se detendrá si el número de iteraciones excede el comando *net.trainParam.epochs*, si se alcanzó el valor del error propuesto como meta, si la magnitud del gradiente es menor que *net.trainParam.min_grad*, o si el tiempo d entrenamiento supera el valor de *net.trainParam.time.*

3.2 Traingdm: Equivale al algoritmo tradicional, más un nuevo coeficiente de momentum, que interviene en el proceso de actualización de los pesos. Si el error de la red en una iteración dada, excede el valor del error en la iteración anterior, en un valor mayor al definido por un radio de cobertura dado el que puede determinarse por medio de la función *max_perf_inc* y que está típicamente alrededor de 1.04, los nuevos pesos y ganancias son descartados y el coeficiente de momentum *mc* es fijado en cero.

La sintaxis de este algoritmo es igual a la utilizada para el algoritmo traingd, más un nuevo comando que permite modificar el coeficiente de momentum

net.trainParam.mc: Valor fijado para el coeficiente de momentum

3.2 Traingda*:* Algoritmo de Gradiente Descendiente, que emplea una rata de aprendizaje adaptiva durante el proceso de entrenamiento. La rata de aprendizaje varía entre 0.01 y 1, una rata de aprendizaje muy pequeña torna lento el aprendizaje, pero si se incrementa demasiado el aprendizaje puede tornarse inestable y crear divergencia, por esto la función *traingda* varía la rata de

aprendizaje tratando de sacar provecho de la inclinación del gradiente en cada momento; su gran desventaja es que los pesos iniciales varían muy poco así se encuentren distantes de los valores de convergencia. La sintaxis de este el algoritmo es la siguiente:

- *net.trainParam.epochs*: Máximo número de iteraciones para obtener convergencia
- *net.trainParam.goal*: Error máximo permitido
- *net.trainParam.lr:* Rata de aprendizaje inicial
- *net.trainParam.lr_inc:* Porcentaje que incrementa la rata de aprendizaje cuando el error disminuye
- *net.trainParam.lr_dec:* Porcentaje en que es decrementada la rata de aprendizaje cuando el error aumenta
- *net.trainParam.max_fail:* Máximo número de fallas
- *net.trainParam.max_perf_inc:* Máximo incremento del rendimiento
- *net.trainParam.min_grad*: Mínimo rendimiento del gradiente
- *net.trainParam.show*: Los resultados son visualizados siempre que transcurre este número de iteraciones.
- *net.trainParam.time*: Máximo tiempo de entrenamiento en segundos

3.3 Trainrp*:* Las redes multicapa, utilizan típicamente una función de transferencia sigmoidal (ver capítulo 1) en las capas ocultas, estas funciones comprimen un infinito rango de entradas, dentro de un finito rango de salidas, además se caracterizan porque su pendiente tendera cada vez más a cero, mientras más grande sea la entrada que se le presenta a la red, esto ocasiona problemas cuando se usa un algoritmo de entrenamiento de pasos descendientes, porque el gradiente empieza a tomar valores muy pequeños y por lo tanto no

habrán cambios representativos en los pesos y las ganancias, así se encuentren bastante lejos de sus valores óptimos. El propósito del algoritmo Backpropagation Resileint (RPROP) es eliminar este efecto en la magnitud de las derivadas parciales. En este algoritmo solamente el signo de la derivada es utilizado para determinar la dirección de actualización de los parámetros, la magnitud de las derivadas no tiene efecto en la actualización. La magnitud en el cambio de cada peso es determinada por separado; el valor del incremento de pesos y ganancias es determinado por el factor *delt_inc*, así la derivada parcial del error con respecto a los pesos tenga el mismo signo durante dos iteraciones sucesivas; el valor de decremento está determinado por el factor *delt_dec* así la derivada del error con respecto a los pesos haya cambiado de signo con respecto a la anterior iteración; si la derivada es cero, entonces el valor actualizado se conserva; si los pesos continúan cambiando en la misma dirección durante varias iteraciones, la magnitud de cambios de los pesos se decrementa.

La sintaxis de este algoritmo se resume a continuación:

- *net.trainParam.epochs:* Máximo número de iteraciones del entrenamiento
- *net.trainParam.show:* Intervalo de visualización de los resultados
- *net.trainParam.goal:* Error deseado
- *net.trainParam.time=inf:* Máximo tiempo de entrenamiento en segundos
- *net.trainParam.min_grad:* Mínimo rendimiento del gradiente
- *net.trainParam.max_fail:* Máximo número de fallas
- *net.trainParam.lr*: Rata de aprendizaje
- *net.trainParam.delt_inc:* Incremento en la actualización de pesos

- *net.trainParam.delt_dec:* Decremento en la actualización de pesos
- *net.trainParam.delta0:* Incremento inicial en la actualización de pesos
- *net.trainParam.deltamax:* Máximo cambio en los pesos

3.4 Trainbfg*:* Algoritmo alternativo que emplea la técnica del gradiente conjugado, su expresión matemática se deriva del método de Newton, con la ventaja de que no es necesario computar las segundas derivadas; este algoritmo requiere mas capacidad de almacenamiento que el algoritmo tradicional, pero generalmente converge en menos iteraciones. Requiere de un cálculo aproximado de la matriz Hessiana, la cual es de dimensiones n^2 x n^2, donde n la cantidad de pesos y ganancias de la red; para redes que involucren una gran cantidad de parámetros es preferible emplear el algoritmo trainrp.

- *net.trainParam.epochs*: Máximo número de iteraciones del entrenamiento
- *net.trainParam.show:* Número de iteraciones entre las cuales se muestran resultados
- *net.trainParam.goal:* Error deseado
- *net.trainParam.time=inf:* Máximo tiempo de entrenamiento en segundos
- net.trainParam.min_grad: Mínimo rendimiento del gradiente
- net.trainParam.max_fail=5: Máximo número de fallas
- *net.trainParam.searchFcn* 'srchcha' Nombre de la rutina de búsqueda lineal a utilizar.
- *net.trainParam.scal_tol*: Se divide entre el valor de Delta para determinar la tolerancia para la búsqueda lineal.
- *net.trainParam.alpha*: Factor de escala que determina una reducción suficiente en el desempeño.

- *net.trainParam.beta*: Factor de escala que determina un tamaño de paso suficientemente grande.
- *net.trainParam.delta:* Tamaño de paso inicial en el intervalo de localización de paso.
- *net.trainParam.gama*: Parámetro para evitar pequeñas reducciones en el desempeño.
- *net.trainParam.low_lim*: Límite inferior en el cambio del tamaño del paso.
- *net.trainParam.up_lim*: Límite superior en el cambio del tamaño del paso.
- *net.trainParam.maxstep:* Máximo longitud de paso.
- *net.trainParam.minstep:* Mínima longitud de paso; por defecto es 1.0e-6
- *net.trainParam.bmax:* Máximo tamaño de paso.

3.5 TrainIm*:* Algoritmo que actualiza los pesos y las ganancias de acuerdo a la optimización de Levenberg-Marquardt. Es el algoritmo más rápido para redes Backpropagation; tiene la desventaja de requerir de un set de entrenamiento lo más estándar posible, pues de otra forma solo aproximará correctamente valores que se encuentren dentro de los patrones de aprendizaje. Si el set de entrenamiento es muy extenso, se recomienda reducir el Jacobiano.

La sintaxis de este algoritmo es la siguiente:

- *net.trainParam.epochs*: Máximo número de iteraciones del entrenamiento
- *net.trainParam.goal:* Error deseado
- *net.trainParam.lr:* Rata de aprendizaje
- *net.trainParam.max_fail:* Máximo número de veces que falla el valor de Mu

- *net.trainParam.mem_reduc*: Factor de fraccionamiento de Jacobiano para ahorrar memoria
- *net.trainParam.min_grad:* Mínimo rendimiento del gradiente
- *net.trainParam.show*: Intervalo de visualización de los resultados.
- *net.trainParam.time:* Máximo tiempo de entrenamiento en segundos
- *tr.mu:* Valor del Mu adaptivo