[[1]](#footnote-2)

Predictor de velocidad para un motor DC con una RNA backpropagation (Enero 2010)

Cajamarca Villa Julio Eduardo, Idrovo Coronel Diego Rolando, Rocano Tenesaca Jose Luis, Valverde Jara Juan Andrés, *Estudiantes*, Universidad Politécnica Salesiana

*Abstract*—En este trabajo se diseña una red backpropagation, específicamente para una de sus aplicaciones, en nuestro caso esta red neuronal es un predictor de velocidad de un motor, tomando datos referenciales de voltaje, corriente y velocidad como parámetros de aprendizaje, para que después la red pueda generar datos de salida que nos harán referencia a la velocidad del motor.

*Index Terms*—Backpropagation, RNA, Motor DC, sensores.

# INTRODUCTION

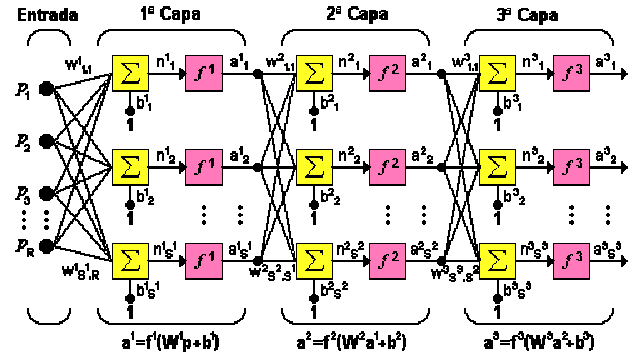
L

as redes de retro propagación o Backpropagation fueron diseñadas bajo la regla de aprendizaje de Widrow-Hoff, para que estas aprendan se da un vector de datos de entrada y uno de salida entrenando a la red de esta manera para sacar una función aproximada que pueda representar estos valores, la realización del aprendizaje esta dado por el algoritmo de gradiente descendente, donde los pesos son movidos a lo largo de toda el gradiente de la función.

Cuando una red Backpropagation está correctamente diseñada, puede darnos valores de salida diferentes para los cuales ha sido entrenada con una muy buena aproximación a su valor real.

# Síntesis teórica backpropagation

La Backpropagation es una red de aprendizaje supervisado, a la que se debe de dar un patrón de entrada y de salida, dado el estimulo de entrada a la red este se propaga desde la primera capa hacia las capas superiores, generando así una salida, esta señal de salida se compara con el valor de la señal de salida esperada y se calcula el error generado para cada salida.



**Figura 1.** Red backpropagation

El error calculado se lo toma en si a la inversa, pues se comienzan a variar los pesos desde la salida hacia el principio, siempre intentando tener el menor error posible, esto se da por el algoritmo de Widrow y Hoff, ya que se basa en el mismo principio de la gradiente decreciente para encontrar el mínimo de la superficie de error en base al MSE.

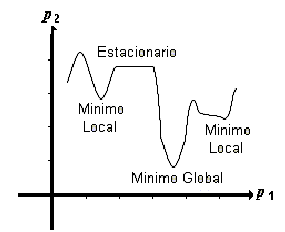
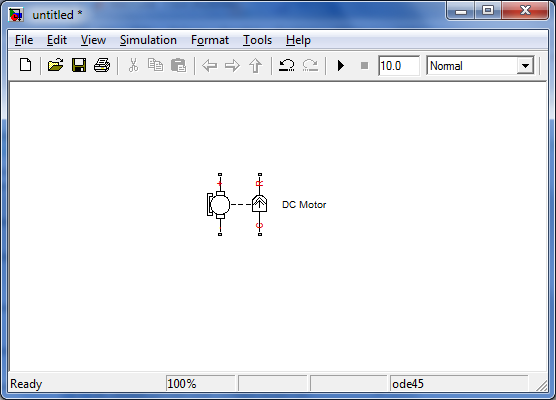


Figura 2. Mínimos locales y globales

# Modelo del sistema en Simulink™

## Modelado del motor DC en Simulink™

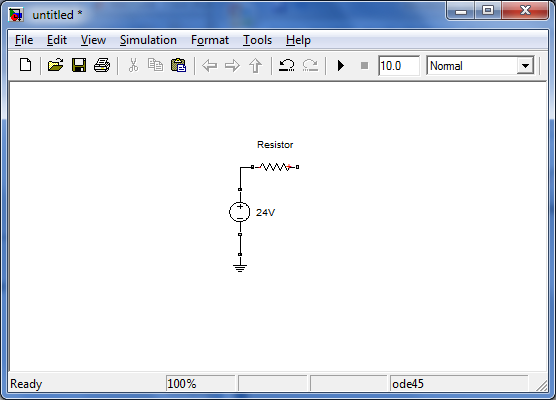
Modelar un motor DC en Simulink™ es sencillo, su bloque se encuentra en la librería Simscape/SimElectronics/Actuators & Drivers, este viene con parámetros preestablecidos, los que podemos variar de acuerdo al motor que deseemos simular. En nuestro caso ingresamos una resistencia de armadura de 4.8Ω, una inductancia de armadura de 1600e-6 H y una contante de torque de 0.0637 N\*m/A.



**Figura 3.** Bloque del motor DC

## Modelado de una fuente de voltaje real en Simulink™

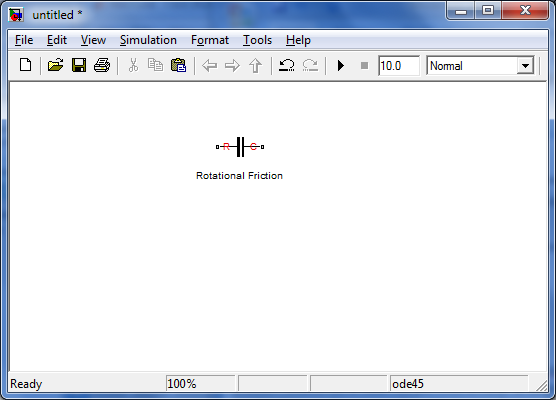
En Simulink™ la fuente de voltaje es ideal se encuentra en la librería Simscape/Foundation Library/, ella mantiene el voltaje en sus bornes independientemente de la corriente que entregue. Para simular una fuente de voltaje real se necesita una resistencia de valor pequeño en serie a la fuente real, para nuestra simulación su valor es de 3Ω.



**Figura 4.** Bloque de la resistencia.

## Modelado de fricción rotacional en Simulink™

En la librería Simscape/Foundation Library/Mechanical/Rotational Elements encontramos el bloque de fricción rotacional, el cual nos permite simular la carga a la que está expuesto el eje del motor DC. Los parámetros usados en nuestro sistema son: Torque de fricción breakaway y Torque de fricción de Coulomb variables desde 0.0001 a 0.2 N\*m. Coeficiente de fricción viscosa de 0 N\*m/(rad/s), coeficiente de aproximación para la transición de 10 s/rad y umbral de la región de velocidad lineal de 0.1 rad/s.

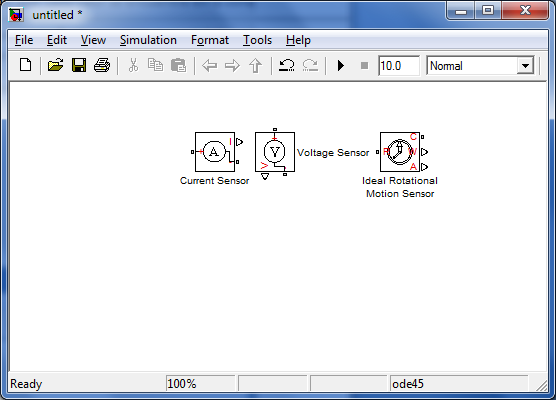


**Figura 5.** Bloque de la fricción rotacional

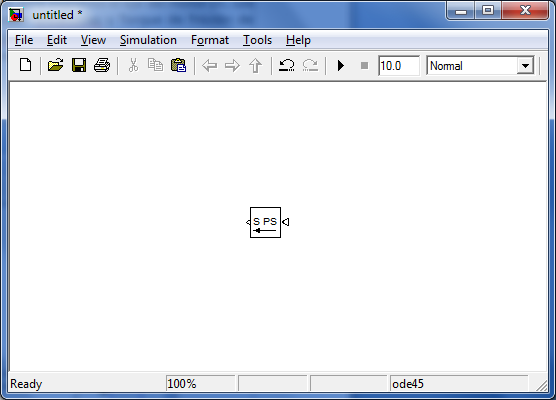
# Adquisición de Datos

Luego de realizar varias pruebas se opto por tomar 100 muestras del sistema, cada una de ellas tomada 3 segundos después de iniciar la simulación, esto con el fin de que cada valor se encuentre en la zona estable del sistema, ya que como sabemos el motor necesita un pequeño tiempo para que estabilice su velocidad a la nominal y por ende la corriente que circulara por el sea la nominal.

Para realizar todos estos objetivos primero debemos añadir sensores de voltaje, corriente (Simscape/Foundation Library/Mechanical/Mechanical Sensors) y de velocidad rotacional (Simscape/Foundation Library/Electrical Sensors) al sistema, estas señales físicas hay que pasarlas a datos con las que pueda trabajar Simulink™ o MATLAB™, el bloque PS-Simulink Converter (Simscape/Utilities) hace este trabajo.

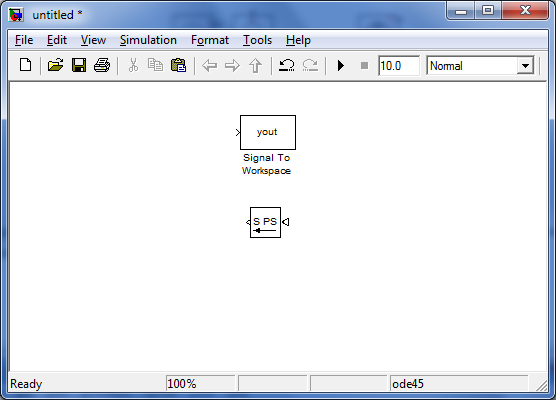


**Figura 6.** Sensores de voltaje, corriente y de velocidad rotacional.

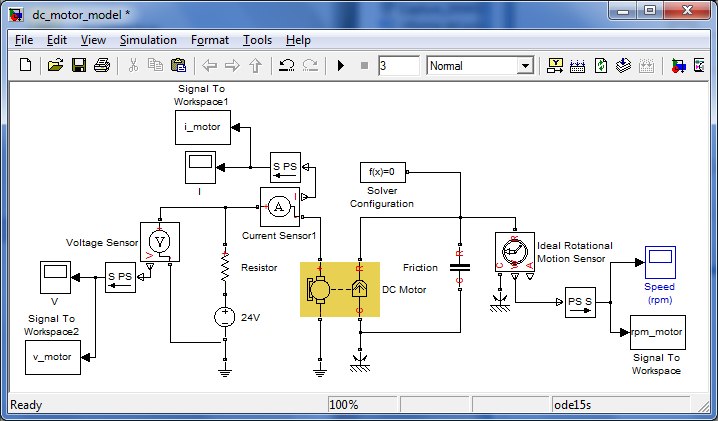


**Figura 7.** Bloque PS-Simulink Converter.

Para poder trabajar desde MATLAB™ es necesario pasar los datos del sistema al workspace a través del bloque Signal to Workspace en la librería Signal Processing Blockset/Signal Processing Sinks.



**Figura 8.** Bloque Signal to Workspace.



**Figura 9.** Modelo del sistema (dc\_motor\_model.mdl).

Una vez que tenemos el modelo creado hacemos un M-file que tomara los datos de nuestro sistema. Este programa llama al modelo del motor, coloca los parámetros de Torque de fricción breakaway y Torque de fricción de Coulomb, simula al sistema y recoge estos datos en vectores para posteriormente guardarlos.

open\_system('dc\_motor\_model');

h=waitbar(0,'1','Name','DC Motor Measurement...','CreateCancelBtn','setappdata(gcbf,''canceling'',1)');setappdata(h,'canceling',0)

i=0;

for trq\_motor=0.0001:0.2/100:0.2

set\_param('dc\_motor\_model/Friction','brkwy\_trq',num2str(trq\_motor,'%0.9f'));

set\_param('dc\_motor\_model/Friction','Col\_trq' , num2str(trq\_motor,'%0.9f'));

set\_param(gcs,'SimulationCommand','Start');

i=i+1;

pause(1);

if getappdata(h,'canceling'),break,end;waitbar(i/100,h,sprintf('%d',i));

v\_dc\_motor(i) = v\_motor;

i\_dc\_motor(i) = i\_motor;

rpm\_dc\_motor(i)=rpm\_motor;

trq\_dc\_motor(i)=trq\_motor;

end

set\_param('dc\_motor\_model/Friction','brkwy\_trq','3.0e-2');

set\_param('dc\_motor\_model/Friction','Col\_trq' , '3.0e-2');

save\_system('dc\_motor\_model');

close\_system('dc\_motor\_model');

if ~getappdata(h,'canceling')

save('motor\_model\_measurement','-regexp','v\_dc\_motor','i\_dc\_motor','rpm\_dc\_motor','trq\_dc\_motor');

end

delete(h);

clear all;

# Diseño de la red Backpropagation

En MATLAB™ usamos el comando new feed forward **newff** para crear una red con conexiones hacia adelante. Para ello especificamos los valores de entrada (MATLAB™ R2009b), el numero de capas, las neuronas de cada capa las funciones de activación (en cada una de las capas) y el nombre del algoritmo de entrenamiento.

net=newff(p,t,[3,1],{'tansig','purelin'},'trainlm');

Ingresamos la matriz (o vector) p con los valores de ingreso, la matriz (o vector) t con los valores esperados, [3,1] indica que existen 2 capas en la red, la primera con 3 neuronas y la segunda con una sola neurona, {'tansig','purelin'} son las funciones de activación de la primera y la segunda capa respectivamente y ' trainlm' es uno de los mejores algoritmos de entrenamiento llamado Levenberg-Marquart. Este comando ya inicia los pesos, sin embargo si deseamos inicializarlos nuevamente utilizamos el comando net=init(net). Esta red una el método del mínimo error cuadrado por defecto.

Los parámetros de entrenamiento para la red son mostrar cada 500 épocas los avances del entrenamiento, con una taza de aprendizaje de 0.05, un número de épocas de 2000 y un error de 1 e-5.

net.trainParam.show = 500;

net.trainParam.lr = 0.05;

net.trainParam.epochs = 2000;

net.trainParam.goal = 1e-5;

Para iniciar el entrenamiento de la red usamos la función net=train(net,p,t) cuyos parámetros son el nombre de la red, la matriz (o vector) p con los valores de ingreso y la matriz (o vector) t con los valores esperados.

En definitiva el archivo M-file para el aprendizaje es:

load motor\_model\_measurement;

p=vertcat(i\_dc\_motor,v\_dc\_motor);

t=rpm\_dc\_motor;

net=newff(p,t,[3,1],{'tansig','purelin'},'trainlm');

net=init(net);

net.trainParam.show = 500;

net.trainParam.lr = 0.05;

net.trainParam.epochs = 2000;

net.trainParam.goal = 1e-5;

[net,tr]=train(net,p,t);

# Pruebas

## RPM con los valores de aprendizaje p en función del voltaje (100pts).



## RPM con otros valores de v e i en función del voltaje(60pts).

## 

# Conclusión

La red backpropagation pueden ser muy útil en sistemas donde no se cuenta siempre con elementos físicos para realizar medidas de nuestro sistema, se puede decir que se lo utiliza como un sensor virtual, ya que en ocasiones no se cuenta con el dinero, la estructura física para la colocación de uno o la inexistencia del mismo.

La exactitud que este genere estará dependiendo de cómo se tome la regla de aprendizaje en nuestro entrenamiento, en nuestro caso al realizar el entrenamiento con el algoritmo de entrenamiento “trainlm”, es donde se genero un menor error, pues utilizando el comando “traingd” la convergencia del sistema no se generaba.

Los valores predeterminados por el software no siempre van a ser los que nos permitan obtener los resultados más idóneos o según las especificaciones que necesitemos, se debe de tomar en cuenta siempre los factores que intervienen cuales se pueden cambiar y cuales se deben de cambiar, pues en ocasiones al querer tener un menor error puede significarnos un mayor tiempo de procesamiento y consumo de más recursos innecesariamente.

Referencias

1. Introducción a las Redes Neuronales (2009), Arredondo Vidal Tomás, Depto. Electrónica UTFSM. Obtenido de: <http://profesores.elo.utfsm.cl/~tarredondo/info/soft-comp/Introduccion%20a%20las%20redes%20neuronales.pdf>
2. Krose Ben, van der Smagt Patrick (1996). An Introduction to Neural Networks. The University of Amsterdam.
3. Neuronal Network Modeling (1994), Neelakanta, P., DeGroff, D., CRC, Boca Raton.

1. [↑](#footnote-ref-2)