Sistemas Inteligentes 1

Práctica 4

Redes Neuronales.

Perceptrón Multicapa.

Backpropagation.

Héctor Garbisu Arocha
Curso 2015-2016
Grado en Ingeniería Informática
Universidad de las Palmas de Gran Canaria

Índice

1. Presentación del problema

Una red neuronal sencilla debe aprender a reconocer dígitos de la lista MNIST de caracteres manuscritos. El tipo de red neuronal es un perceptrón multicapa aprendiendo por gradient descent y backpropagation.

La lista de caracteres es una matriz de 28*28x60000 para el entrenamiento y 28*28x10000. Por tanto la cantidad de entradas es 784.

2. Esquema general del algoritmo

Los alumnos tenemos libertad para implementar el numero de capas y neuronas a nuestro criterio. Para optimizar los tiempos de entrenamiento y tasa de acierto me he terminado decantando por una arquitectura de 10 salidas y una sola capa oculta de 600 neuronas.

La secuencia de operaciones que se siguen, independientemente de la forma en que estén implementadas, es la siguiente:

- Inicialización aleatoria de pesos
- Cálculo de las salidas de cada capa, pasándolas como entrada a la siguientes
- Cálculo de las diferencias parciales y gradiente de la función de coste
- Actualización de los pesos en función de la pendiente del coste y la diferencia con el resultado esperado

3. Implementación general

Con el esquema anterior, ya tenemos algo con lo que empezar. Como es improbable calcular la arquitectura óptima para la red de antemano, la primera implementación es muy sencilla y luego se modifica para añadir mejoras.
Algunas de las modificaciones que se realizan ayudan al aprendizaje y otras no. Otras propuestas simplemente podrían no haber salido bien porque no he sabido llevarlas a cabo.

```
X = loadMNISTImages('train-images.idx3-ubyte');
la = loadMNISTLabels('train-labels.idx1-ubyte');
la = la/10; % etiquetas entre 0 y 1;
ninputs = size(X,1);% numero de pixeles
nhidden = 200;% numero de neuronas 1ª capa
numimagenes = size(X,2);% numero de imagenes
noutput = 1;% neuronas de salida
alph = 0.1;% factor de aprendizaje

% Pesos y Bias iniciales
% INPUT LAYER 785 neuron
Wih = rand(ninputs,nhidden);
% HIDDEN LAYER (?)
Who = rand(nhidden,noutput);
Bh = rand(1,nhidden);
% OUTPUT LAYER
Bo = rand(1,noutput);

% Variables auxiliares
incaciertos = zeros(numimagenes,1);
incmse = zeros(numimagenes,1);
aciertos = 0;
dif = 0;
%% Entrenamiento
for i=1:1:numimagenes
    %% Cálculos hacia delante
    % Valores de la capa oculta
    H = X(:,i)'*Wih+Bh;
    H = ajusta(H,52.7,170);
%     Valores de la capa de salida
    O = H*Who+Bo;
    O = ajusta(O,196,207);

    % Cálculos hacia atrás
%     Deltas
    dO = O*(1-O)*(la(i)-O);
    dH = H*(1-H')*dO*Who;

%     Reevaluación de Pesos y Bias
    Who = Who + alph.*H'.*dO;
    Bo = Bo + alph.*dO;
    Wih = Wih + alph.*X(:,i)*dH';
    Bh = Bh + alph.*dH';

end
```

El código anterior se corresponde con una red de 200 neuronas en una sola capa oculta. Hace una sola época y termina dando una tasa de acierto con el conjunto de prueba de en torno al 20%.

La función de ajuste hace que la distribución de los pesos y las salidas formen una sigmoide. A la función de ajuste se le pasa la media y el valor máximo de la distribución sin ajustar, resta la media y divide por el máximo para tener una distribución entre 0 y 1 (se asume que no hay valores negativos porque rand da valores entre 0 y 1).

Los valores del factor de aprendizaje (alph) los actualicé tras cada prueba. El valor final de alfa es el más alto posible que garantiza que los errores medios cuadráticos convergen, minimizando el mse final.

Para comprobar los resultados utilicé un código similar al del entrenamiento de la red pero omitiendo la retroprograpación, y utilizando el conjunto de validación (10k imágenes).

4. Cambios al modelo incial

4.1 Capa oculta adicional

Unas cuantas pruebas con más de una capa oculta no produjeron mejoras sustanciales.

Los cambios para añadir una capa nueva son los siguientes:

- Inicialización

```
nhidde2 = 20;
```

```
Whh = rand(nhidden,nhidde2);
Bh2 = rand(1,nhidde2);
```

- Cómputo

```
H2 = H1*Whh+Bh2;
H2 = ajusta(H2,190,402);
```

- Deltas (los valores de la primera capa ahora se actualizan en función de la diferencia con la nueva)

```
dO  = O*(1-O)*(la(i)-O);
dH2 = H2*(1-H2')*dO*Who;
dH1 = H1*(1-H1')*dH2'*Whh';
```

- Actualización de Pesos y Bias

```
Whh = Whh + alph.*H1'*dH2';
Bh2 = Bh2 + alph.*dH2';
```

La red aprende, pero aproximadamente al mismo ritmo y termina, comprobando

4.2 Distribución normal para los valores iniciales

```
% INPUT LAYER 785 neuron
Wih = normrnd(0,1,ninputs,nhidden);
% HIDDEN LAYER (?)
Who = normrnd(0,1,nhidden,noutput);
Bh = normrnd(0,1,1,nhidden);
% OUTPUT LAYER
Bo = normrnd(0,1,1,noutput);
```

El problema de esta implementación es que ajustar al máximo, puede dividir cada valor de salida de la capa intermedia, por un número muy grande.
No he sabido encontrar la función de activación apropiada o cómo utilizar las posibles ventajas de una distribución normal.

4.3 Más salidas

Añadir más neuronas a la capa de salida es la forma más fácil de mejorar la precisión de la red. Ahora en vez de tener una neurona responsable de distinguir valores que difieren en 0.1, habrá 10, cuyas salidas deben ser 0, 1 o un valor muy cercano a éstos.

En primer lugar, el valor objetivo deja de ser un número. Ahora es un vector con un 1 en la posición indicada por la etiqueta.

```
lavec = zeros(size(la,1),10);
for i=1:size(lavec,1)
    lavec(i,la(i)+1) = 1;
end
```

El cálculo y descenso por gradiente cambia un poco

```
    distancia = lavec(i,:)-O;
    dO  = O*(1-O')*distancia;
```

El número que se obtiene como salida es el índice-1, del mayor valor que haya en el vector de salidas.

```
    [x,ind] = max(abs(O));
    aciertos = aciertos + lavec(i,ind);
```

En este caso, probando con 600 neuronas se obtiene aproximadamente un 90% de tasa de acierto tras entrenar con cada imagen una vez.

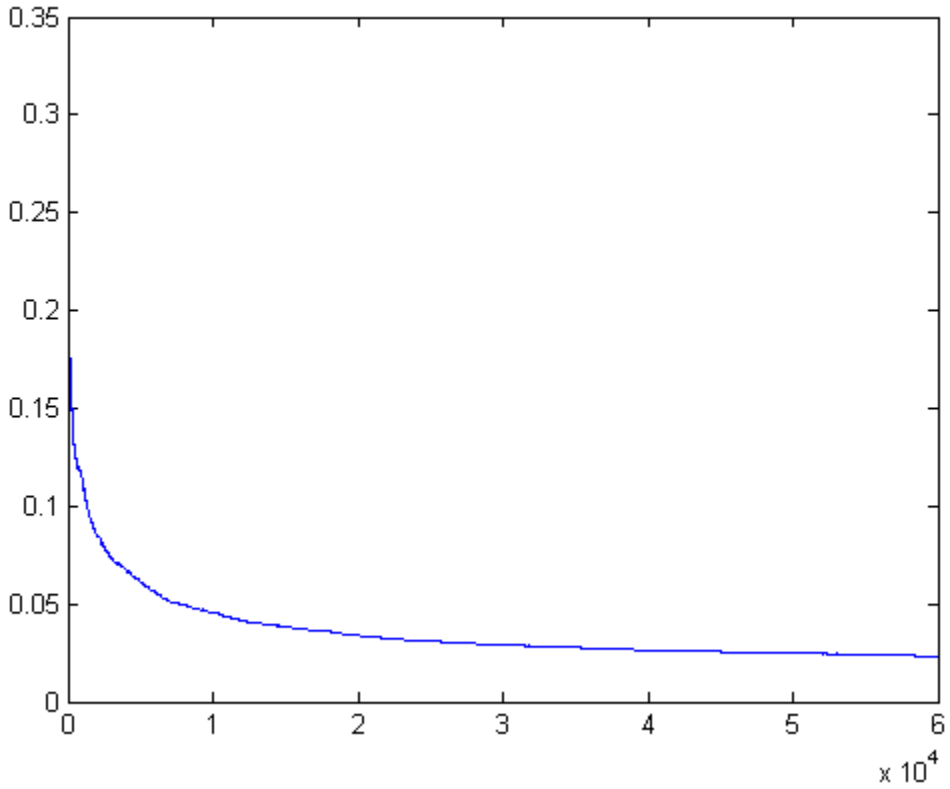
4.4 Más épocas

La útlima mejora consiste en utilizar varias veces el conjunto completo de imágenes en varias épocas. Ésto se implementa simplemente con otro bucle for.
La mejora sobre la red anterior para 10 épocas pasa de 90% de tasa de acierto a 92 o 93%. El error mínimo cuadrático se mantiene muy similar.

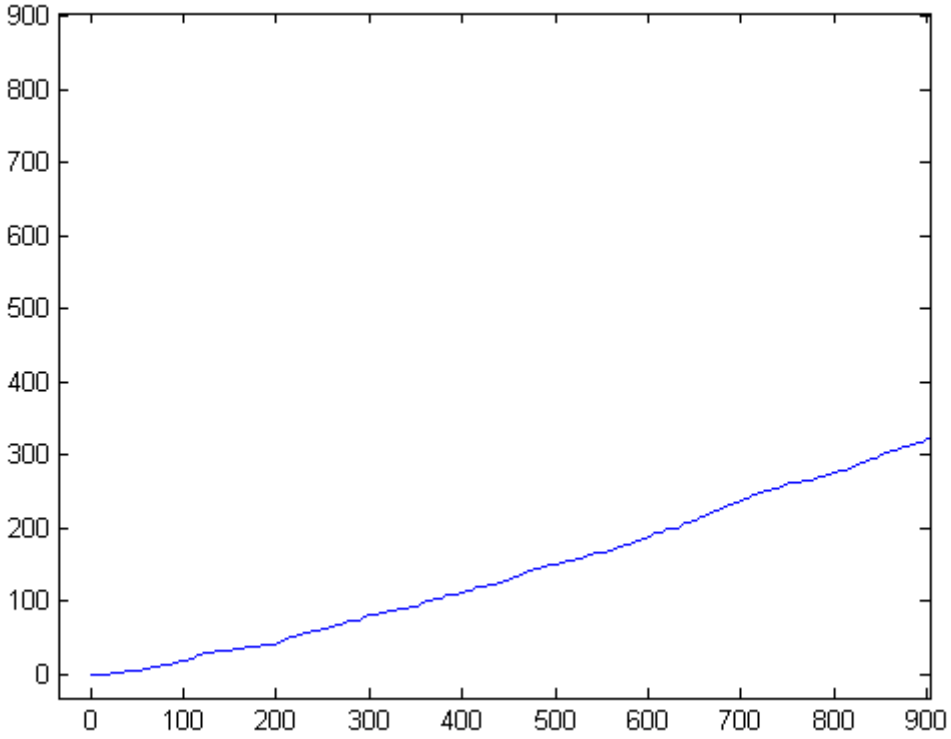
5. Resultados y conclusión

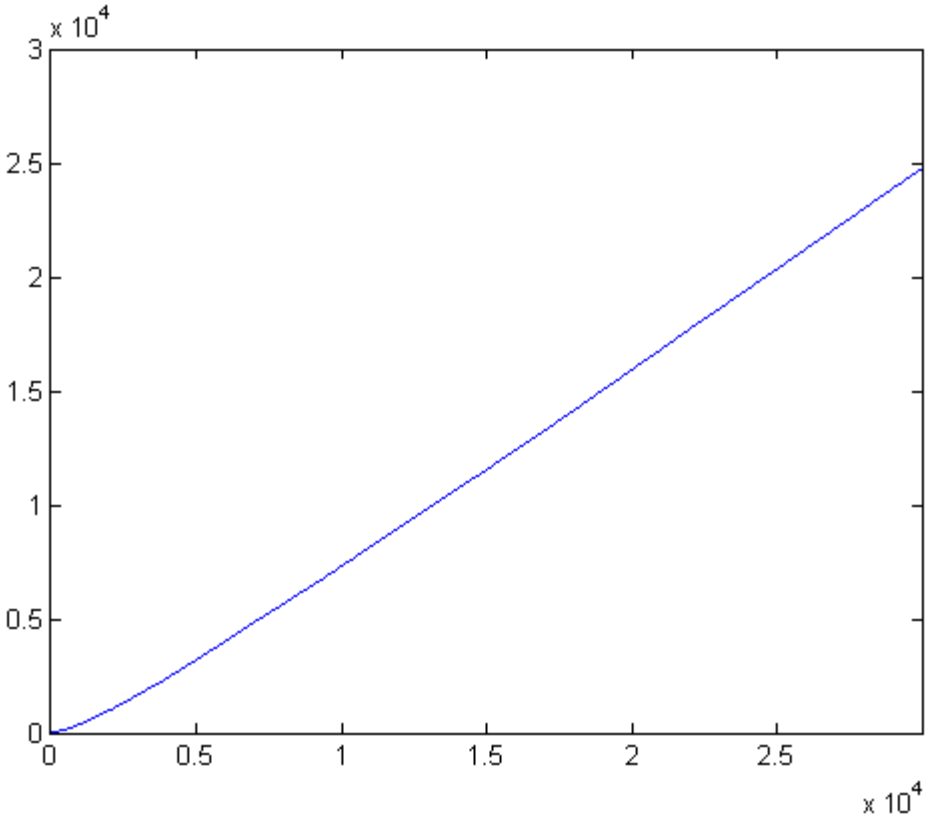
El factor limitante ha sido el tiempo. Es probable que una capa de varias salidas, varias capas ocultas y un número mayor de iteraciones hubiera sido una combinación mejor que las que he probado.
También me habría gustado multiplicar la variedad de las entradas mediante ruido o deformación del conjunto de entrada.

Progreso del MSE para la prueba de 60k imágenes:



La tasa de acierto para las primeras mil imágenes está cerca del 30% luego se acerca muy rápido al valor final en torno al 90%. Las dos imágenes siguientes son la misma gráfica de la tasa de acierto en función del tiempo durante el entrenamiento. Cambia la escala.





6. Anexo. Código

red_entrena_10o.m

```
X = loadMNISTImages('train-images.idx3-ubyte');
la = loadMNISTLabels('train-labels.idx1-ubyte');
% matriz de hot-zeros
% La salida esperada es un 1 en la
% neurona correspondiente
lavec = zeros(size(la,1),10);
for i=1:size(lavec,1)
    lavec(i,la(i)+1) = 1;
end
% numero de pixeles
ninputs = size(X,1);
% numero de neuronas 1ª capa
nhidden = 600;
% numero de imagenes
numimagenes = size(X,2);
% neuronas de salida
noutput = 10;
% factor de aprendizaje
alph = 0.1;

%% Pesos y Bias iniciales
% HIDDEN LAYER (?)
Wih = rand(ninputs,nhidden);
Bh = rand(1,nhidden);
% OUTPUT LAYER
Who = rand(nhidden,noutput);
Bo = rand(1,noutput);
% Variables auxiliares
aciertos = 0;
dif = 0;
incaciertos = zeros(numimagenes,1);
incmse = zeros(numimagenes,1);
Hes = zeros(numimagenes,nhidden);
Oes = zeros(numimagenes,noutput);
%% Entrenamiento
tic
niteraciones = 10;
for j=1:1:niteraciones
aciertos = 0;
dif = 0;
incaciertos = zeros(numimagenes,1);
incmse = zeros(numimagenes,1);
Hes = zeros(numimagenes,nhidden);
Oes = zeros(numimagenes,noutput);
disp(strcat('conjunto :',num2str(j),'/',num2str(niteraciones)));
for i=1:1:numimagenes
    %% Cálculos hacia delante
    %% Valores de la 1ª capa oculta
    H1 = X(:,i)'*Wih+Bh;
    H1 = ajusta(H1,52.7,174);
    Hes(i,:) = H1;

    %%  Valores de la capa de salida
    O = H1*Who+Bo;
    O = ajusta(O,49,103);
    Oes(i,:) = O;

    %% Calculo del coste
```

```
    distancia = lavec(i,:)-O;

    %% Cálculos hacia atrás APRENDIZAJE
%     Deltas
    dO  = O*(1-O')*distancia;
    dH1 = H1*(1-H1')*dO*Who';

% %     Reevaluación de Pesos y Bias
    Who = Who + alph.*H1'*dO;
    Bo  = Bo  + alph.*dO;

    Wih = Wih + alph.*X(:,i)*dH1;
    Bh = Bh + alph.*dH1;

    %% Comprobacion
    [x,ind] = max(abs(O));
    aciertos = aciertos + lavec(i,ind);
    incaciertos(i) = aciertos;
    dif = dif + (((ind-1)-la(i,:))/10).^2;
    mse = dif/i;
    incmse(i) = mse;

    if(mod(i,6000)==0)
        disp(strcat('training: ',num2str(i),'/ ',num2str(numimagenes)));
        disp(strcat('aciertos :',num2str(aciertos),'
( ',num2str(aciertos/i),'/1)'));
        disp(strcat('mse :',num2str(mse)));
    end
end
end
[alph nhidden noutput mse(end)]
plot(incmse)
toc
```

red_comprueba_10o.m

```
X = loadMNISTImages('train-images.idx3-ubyte');
la = loadMNISTLabels('train-labels.idx1-ubyte');
% matriz de hot-zeros
% La salida esperada es un 1 en la
% neurona correspondiente
lavec = zeros(size(la,1),10);
for i=1:size(lavec,1)
    lavec(i,la(i)+1) = 1;
end
% numero de pixeles
ninputs = size(X,1);
% numero de neuronas 1ª capa
nhidden = 600;
% numero de imagenes
numimagenes = size(X,2);
% neuronas de salida
noutput = 10;
% factor de aprendizaje


aciertos = 0;
dif = 0;
incaciertos = zeros(numimagenes,1);
incmse = zeros(numimagenes,1);
Hes = zeros(numimagenes,nhidden);
Oes = zeros(numimagenes,noutput);
```

```
%% Entrenamiento
tic
for i=1:1:numimagenes
    %% Cálculos hacia delante
    %% Valores de la 1ª capa oculta
    H1 = X(:,i)'*Wih+Bh;
    H1 = ajusta(H1,52.7,174);
    Hes(i,:) = H1;

    %%  Valores de la capa de salida
    O = H1*Who+Bo;
    O = ajusta(O,49,103);
    Oes(i,:) = O;


    %% Comprobacion
    [x,ind] = max(abs(O));
    aciertos = aciertos + lavec(i,ind);
    incaciertos(i) = aciertos;
    dif = dif + (((ind-1)-la(i,:))/10).^2;
    mse = dif/i;
    incmse(i) = mse;

    if(mod(i,6000)==0)
        disp(strcat('training: ',num2str(i),'/ ',num2str(numimagenes)));
        disp(strcat('aciertos :',num2str(aciertos),'
( ',num2str(aciertos/i),'/1)'));
        disp(strcat('mse :',num2str(mse)));
    end
end
[alph nhidden noutput mse(end)]
% plot(incmse)
toc
```