Aprendizaje Automático

Beltrán Domínguez, Victoria
Ge, Xiangyang

Memoria Práctica 3: Redes neuronales multicapa

El objetivo de esta práctica es experimentar con redes neuronales multicapa dentro de octave, en concreto con redes Multilayer Perceptron (MLP). Para ello utilizamos la librería nnet para octave.
Así, la práctica nos va guiando sobre cómo cargar los datos y barajarlos para coger un 80% aleatorio de las muestras para entrenamiento, y un 20% restante para validación.
El primer ejercicio que nos plantea la práctica es cómo cambiar las etiquetas de clase a one-hot encoding. Para hacerlo, creamos una matriz de ceros que tendrá tantas filas como clases diferentes y tantas columnas como muestras. Entonces en un bucle, leemos la clase y ponemos a uno la fila que sea igual a la clase con la columna que sea el número de la muestra correspondiente. Esto lo hacemos tanto para entrenamiento como para validación. El siguiente código de octave realiza los cálculos:

```
%Cambiamos a one-hot encoding las etiquetas de clase
filas=rows(unique(trlabels));
columnas=nTr;
mTrainOutputResult=zeros(filas,columnas);
for i=1:1:columnas
  clase=mTrainOutput(i);
  mTrainOutputResult(clase,i)=1;
endfor
filas=rows(unique(tslabels));
columnas=nSamples-nTr;
mValidOutputResult=zeros(filas,columnas);
for i=1:1:columnas
  clase=mValiOutput(i);
  mValidOutputResult(clase,i)=1;
endfor
```

A continuación, normalizamos los datos, configuramos la red neuronal y entrenamos la red neuronal tal y como la práctica indica.

Al normalizar y clasificar las muestras de test para conocer el error, nos encontramos con que no obtenemos el porcentaje de error o precisión, sino que obtenemos los valores de salida de la red para cada clase (por filas) y para cada muestra (por columnas). Por lo tanto, el siguiente ejercicio pide crear una función que reciba la salida de la MLP para el conjunto de test y sus etiquetas y devuelva el porcentaje de error de clasificación. La siguiente imagen muestra la implementación deseada de la función para devolver el error:

```
function [error] = calcularError (simOut, labels_corres)
error=0;
nMuestras=columns(simOut);
for i=1:1:nMuestras
  [~,clase]=max(simOut(i));
  if(labels_corres(i)!=clase) error+=1;endif
endfor
error=error/nMuestras;
endfunction
```

La llamamos desde nuestro programa como:

```
%Clasificar los datos
simOut= sim(net,mTestInputN);
error=calcularError(simOut,mTestOutput)
```

La práctica nos avisa de que las tasas de error en la tarea hart (80 % entrenamiento, 20 % validación) utilizando 1, 2 y 5 neuronas en la capa oculta con función tansig, y en la capa de salida con función logsig son 14.90 %, 11.30 % y 2.40 %, respectivamente. Calculamos el error y como se muestra en la figura, nos da correctamente, por lo que las funciones implementadas son correctas.

```
Porcentaje de aciertos con una neuronas en la capa oculta: 14.900
Porcentaje de aciertos con dos neuronas en la capa oculta: 11.300
Porcentaje de aciertos con cinco neuronas en la capa oculta: 2.400
```

La última tarea que nos propone la práctica consiste en encontrar los parámetros adecuados para obtener el mínimo error sobre la clasificación de MNIST con redes neuronales. Cabe destacar que a causa de la gran dimensionalidad, es necesario reducirla con PCA. Así, tendremos que encontrar qué proyección es la idónea, además de otros parámetros a considerar como el número de capas ocultas y el número de neuronas en

cada capa. Otra opción comentada en la práctica es seleccionar un subconjunto del conjunto de entrenamiento de acuerdo a algún criterio (i.e. aleatorio) para entrenar una red más compleja. Esta estrategia posibilita la combinación de clasificadores entrenados en subconjuntos del conjunto de entrenamiento que es propia de las técnicas de Bagging.
Hemos elegido simplemente ir probando valores para ver la tendencia de estos.

PCA	# Neuronas por capa	# Capas ocultas	Error
10	2	1	0.456±0.010
10	4	1	0.277±0.009
10	8	1	0.164±0.007
10	16	1	0.106±0.006
10	32	1	0.076±0.005
10	2	2	0.532±0.010
10	4	2	0.254±0.009
10	8	2	0.126±0.006
10	16	2	0.082±0.005
10	2	3	0.611±0.010
10	4	3	0.245±0.008
10	8	3	0.106±0.006

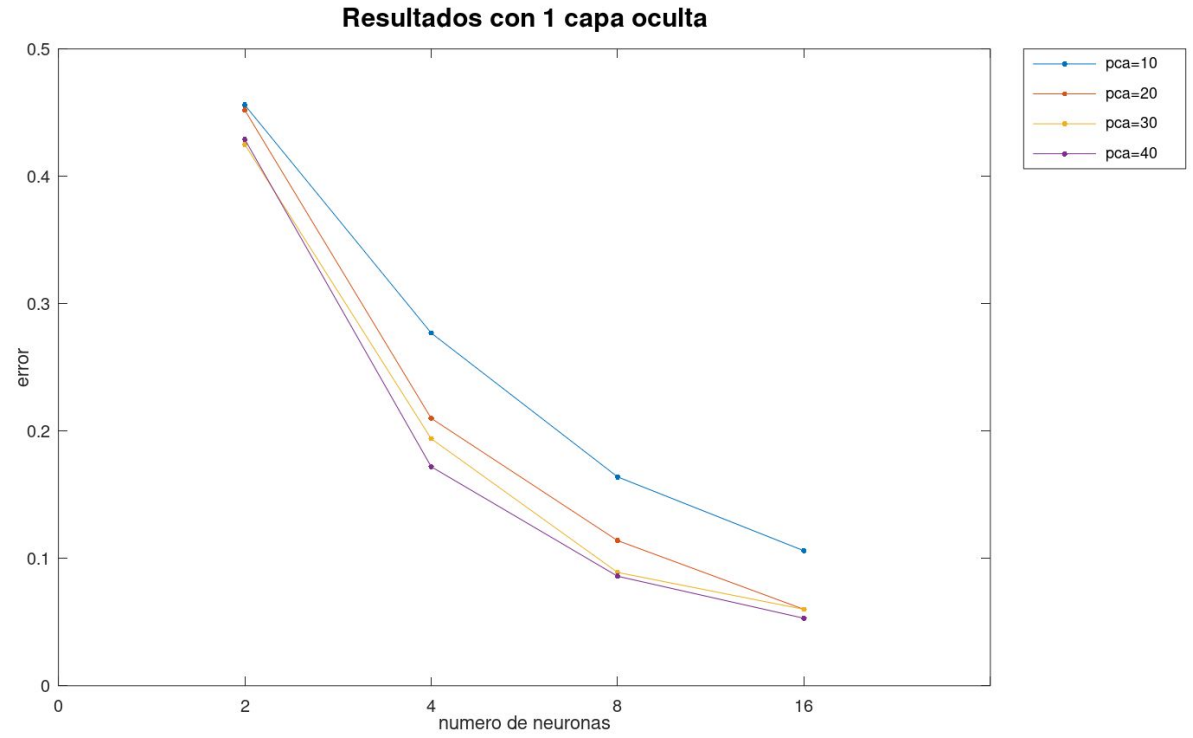
PCA	# Neuronas por capa	# Capas ocultas	Error
20	2	1	0.452±0.010
20	4	1	0.210±0.008

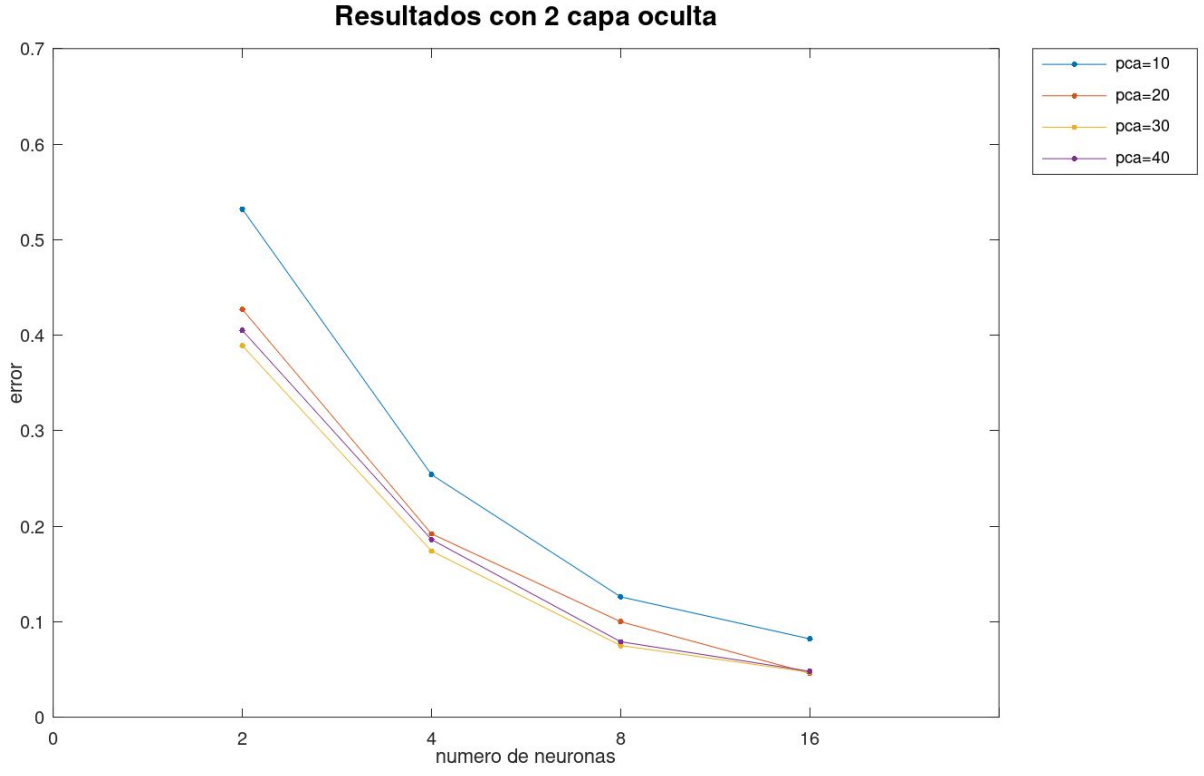
20	8	1	0.114±0.006
20	16	1	0.060±0.005
20	32	1	0.042±0.004
20	2	2	0.427±0.010
20	4	2	0.192±0.008
20	8	2	0.100±0.006
20	16	2	0.046±0.004
20	2	3	0.478±0.010
20	4	3	0.220±0.008
20	8	3	0.085±0.005

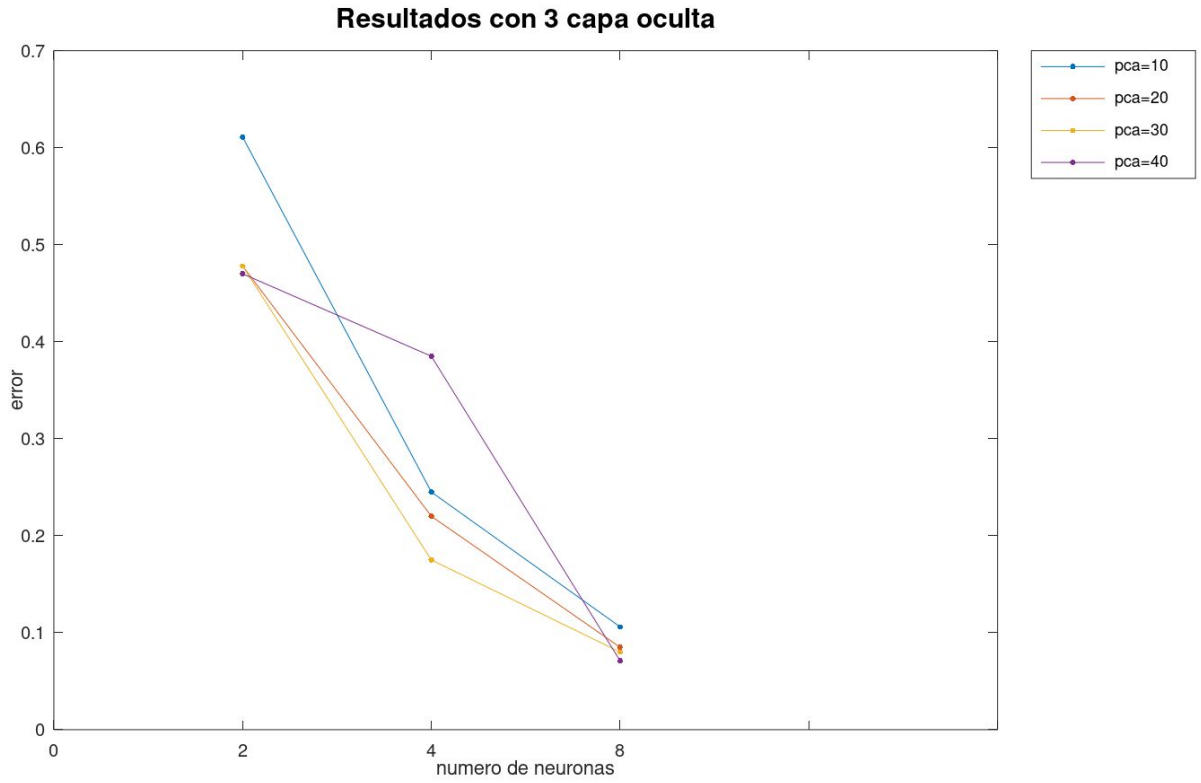
PCA	# Neuronas por capa	# Capas ocultas	Error
30	2	1	0.425±0.010
30	4	1	0.194±0.008
30	8	1	0.089±0.006
30	16	1	0.060±0.005
30	2	2	0.389±0.010
30	4	2	0.174±0.007
30	8	2	0.075±0.005
30	16	2	0.047±0.004
30	2	3	0.478±0.010
30	4	3	0.175±0.007
30	8	3	0.080±0.005
30	16	3	0.046±0.004

PCA	# Neuronas por capa	# Capas ocultas	Error
40	2	1	0.429±0.010
40	4	1	0.172±0.007
40	8	1	0.086±0.005
40	16	1	0.053±0.004
40	2	2	0.405±0.010
40	4	2	0.186±0.008
40	8	2	0.079±0.005
40	16	2	0.048±0.004
40	2	3	0.470±0.010
40	4	3	0.385±0.010
40	8	3	0.071±0.005

También adjuntamos algunas gráficas para una mejora visual y poder estudiar mejor los resultados obtenidos.


Resultados con 1 capa oculta
error
numero de neuronas
0
0.1
0.2
0.3
0.4
0.5
0
2
4
8
16
pca=10
pca=20
pca=30
pca=40


Resultados con 2 capa oculta
error
numero de neuronas
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2
4
8
16
pca=10
pca=20
pca=30
pca=40



Como podemos observar en los diferentes valores obtenidos mediante las gráficas anteriores, los errores tienden a bajar al aumentar el número de neuronas por capa. Sin embargo, en cuanto a la reducción PCA, los resultados parecen bastante similares entre las reducciones 20, 30 y 40. Esto significa que ya no es necesario aumentar la reducción PCA ya que su reducción de error empieza a empequeñecer.

Los errores mínimos de las ejecuciones realizadas se encuentran alrededor de 4-5% (i.e. pca 20, 32 neuronas y 1 capa). No hemos podido conseguir un error más bajo debido a la falta de poder computacional y/o memoria RAM.