Universidad de Córdoba
Escuela Politécnica Superior de Córdoba

Ingeniería Informática
Especialidad: Computación
Cuarto curso. Primer cuatrimestre

Introducción a los modelos computacionales.

Práctica 3: Implementación de las redes neuronales RBF.

José Manuel Cuevas Muñoz
31013019A
i62cumuj@uco.es

Curso académico 2019-2020
Córdoba, 17 de noviembre de 2019

Índice

Índice de figuras II

Índice de tablas III

1. Introducción y pasos a seguir 1

2. Resultados prácticos 2

2.1. Pruebas modificando la arquitectura de la red neuronal 2
2.2. Variando η y algoritmo de regularización 5
2.3. Iniciación del algoritmo k-means: de aleatorio a k-means++ . 8
2.4. Usando regresión en clasificación 9
2.5. Resultados prácticos en nomnist 10

Índice de figuras

1. Matriz de confusión de nomnist 10
2. Letras falsamente clasificadas como A 11
3. Letras falsamente clasificadas como B 11
4. Letras falsamente clasificadas como C 11
5. Letras falsamente clasificadas como D 11
6. Letras falsamente clasificadas como E 11
7. Letras falsamente clasificadas como F 11

Índice de tablas

1. Base de datos seno con varias arquitecturas 3
2. Base de datos quake con varias arquitecturas 3
3. Base de datos parkinson con varias arquitecturas 3
4. Base de datos vote con varias arquitecturas 4
5. Base de datos nomnist con varias arquitecturas 4
6. Train de la base de datos vote con varios eta y L2 5
7. Test de la base de datos vote con varios eta y L2 6
8. Train de la base de datos nomnist con varios eta y L2 7
9. Test de la base de datos nomnist con varios eta y L2 8
10. Comparación entre iniciación aleatoria y kmeans++ en regresión 9
11. Comparación entre iniciación aleatoria y kmeans++ en clasificación 9
12. Comparación entre usar clasificación y regresión en un problema de clasificación 10

1. Introducción y pasos a seguir

Anteriormente, hemos desarrollado perceptrones multicapa, un modelo de red neuronal con una o varias capas ocultas que se basaban en funciones de proyección y con una fase de aprendizaje. Sin embargo las redes neuronales en base radial tienen una filosofía totalmente distinta. Estas redes neuronales tienen una parte de aprendizaje supervisado y otra parte no supervisado. La arquitectura de la RBF está formada por una capa de entrada, una capa oculta y una de salida que será sumatorio en caso de regresión y Softmax en caso de clasificación. Por lo general, este tipo de redes aprende sobre regiones del espacio de entrenamiento.

Este tipo de modelos separa los patrones mediante hiperesferas; una por cada neurona de la capa oculta en la red RBF; las cuales se definen mediante un centro y un radio. El procedimiento, por tanto, para realizar un entrenamiento de la RBF es el siguiente:

1. Si es clasificación, sacamos los centroides estratificados de las clases.
2. Realizamos el algoritmo K-Means con tantos clusters como neuronas RBF haya y sacamos de aquí los centros y las distancias entre neuronas.
3. Calculamos los radios de cada neurona como $\frac{1}{2*(n_1-1)} * \sum_{i \neq j} ||c_j - c_i||$
4. Calculamos la matriz R, que es el valor de activación de cada neurona para cada patrón.
5. Si es regresión:
   a) Calculamos la inversa (o pseudoinversa en caso de que no sea cuadrada) de la matriz R y la multiplicamos por las salidas de entrenamiento para conseguir la matriz de coeficientes β.
   b) Para predecir, multiplicamos MatrizR*β
6. Si es clasificación:
   a) Utilizamos el algoritmo de regresión logística con la matriz R.

2. Resultados prácticos

Para los resultados, tenemos cuatro bases de datos con las cuales haremos diversas pruebas y sacaremos conclusiones a partir del error y desviacion típica del error obtenido:

- **Función seno**: En esta base de datos conformada por 120 patrones de train y 41 patrones de test se intenta obtener la función seno a partir de unos datos con algo de ruido.
- **Base de datos quake**: esta base de datos está compuesta por 1633 patrones de train y 546 patrones de test.se corresponde con una base de datos en la que el objetivo es averiguar la fuerza de un terremoto (medida en escala sismológica de Richter). Como variables de entrada, utilizamos la profundidad focal, la latitud en la que se produce y la longitud.
- **Base de datos parkinsons**: esta base de datos está compuesta por 4406 patrones de train y 1469 patrones de test. Contiene, como entradas o variables independientes, una serie de datos clínicos de pacientes con la enfermedad de Parkinson y datos de medidas biométricas de la voz, y, como salidas o variables dependientes, el valor motor y total del UPDRS (de las siglas en inglés Unified Parkinson's Disease Rating Scale).
- **Base de datos vote**: vote contiene 326 patrones de entrenamiento y 109 patrones de test. La base de datos incluye los votos para cada uno de los para cada uno de los candidatos para el Congreso de los EEUU, identificados por la CQA. Todas las variables de entrada son categóricas.
- **Base de datos noMNIST**: Base de datos formada por 900 patrones de train y 300 de test. La base de datos incluye imágenes de letras de la A a la F identificada por sus valores en escala de grises.

2.1. Pruebas modificando la arquitectura de la red neuronal

Nuestra red neuronal RBF se compone de la capa de entrada, una capa oculta y la capa de salida. De estas capas la única de la cual se puede modificar a priori la arquitectura es de la capa oculta. En nuestro caso esta

capa oculta está formada por num_rbf, un argumento que se calcula multiplicando ratio_rbf por el numero de patrones en entrenamiento. Este valor puede ser modificado con el flag -r a la hora de ejecutar. Para comprobar cual es la mejor arquitectura para cada base de datos, vamos a probar con distintos ratio_rbf y a comparar. Los valores resaltados en color amarillo son los elegidos en cada una de las bases de datos.

ratio_rbf	mean_train_mse	std_train_mse	mean_test_mse	std_test_mse
0.05	0.013816	0.00018855	0.0221816	0.000245
0.15	0.011215	6.806153e-05	0.369389	0.08050
0.25	0.010369	2.458570e-05	1.87887	0.38689
0.5	0.010357	2.33336e-06	2.091692	0.39019

Tabla 1: Base de datos seno con varias arquitecturas

ratio_rbf	mean_train_mse	std_train_mse	mean_test_mse	std_test_mse
0.05	0.0283651	6.2540976e-5	0.0284754	0.0002226
0.15	0.0253314	0.0001182	0.0408557	0.0019817
0.25	0.0221973	8.9793824e-5	0.9496999	0.2907586
0.5	0.0183975	8.2725448e-5	167.94112	14.917458

Tabla 2: Base de datos quake con varias arquitecturas

Tanto en quake como en la base de datos seno, el error de test va aumentando de manera considerable con forme vamos aumentando el numero de neuronas en capa oculta, dado a que puede darse el caso de sobreentrenamiento

ratio_rbf	mean_train_mse	std_train_mse	mean_test_mse	std_test_mse
0.05	0.0216294	0.0002470	0.0247395	0.0003331
0.15	0.0139700	0.0002877	0.0198037	0.0004122
0.25	0.0103408	7.9820e-5	0.0223383	0.0015536
0.5	0.0052210	8.6626e-5	0.0546421	0.0048177

Tabla 3: Base de datos parkinson con varias arquitecturas

En el caso de la base de datos parkinsons, se necesita un ratio de neuronas mas alto que en las otras para que converja completamente.

ratio_rbf	mean_train_mse	std_train_mse	mean_train_ccr	std_train_ccr
0.05	0.0262427	0.0034409	96.134969	0.6886485
0.15	0.0035828	1.2924e-6	99.386503	0.0
0.25	0.0035805	7.5503e-7	99.386503	0.0
0.5	0.0035800	2.3141e-7	99.386503	0.0
	mean_test_mse	std_test_mse	mean_test_ccr	std_test_ccr
0.05	0.0351429	0.0018054	95.963302	0.4494476
0.15	0.0672803	0.0100205	92.293577	1.2444642
0.25	0.0580698	0.0056816	93.577981	0.5802344
0.5	0.0577294	0.0009391	94.128440	0.4494476

Tabla 4: Base de datos vote con varias arquitecturas

En el caso de las bases de datos de clasificación tenemos que tener en cuenta además del MSE el CCR.

ratio_rbf	mean_train_mse	std_train_mse	mean_train_ccr	std_train_ccr
0.05	0.0301910	0.0015274	87.933333	0.7844161
0.15	0.0005102	0.0001332	100.0	0.0
0.25	3.5951e-5	5.3695e-6	100.0	0.0
0.5	1.7112e-5	1.1550e-6	100.0	0.0
	mean_test_mse	std_test_mse	mean_test_ccr	std_test_ccr
0.05	0.0302849	0.0016821	88.533333	1.1469767
0.15	0.0423402	0.0026951	86.533333	1.0456258
0.25	0.0380750	0.0016513	87.333333	0.5577733
0.5	0.0362962	0.0024778	87.866666	1.0022197

Tabla 5: Base de datos nomnist con varias arquitecturas

2.2. Variando η y algoritmo de regularización

η	l2	mean_train_mse	std_train_mse	mean_train_ccr	std_train_ccr
1.0	False	0.0412458	0.0022807	94.539877	0.3005508
0.1	False	0.0290820	0.0027380	95.705521	0.5132883
0.01	False	0.0265450	0.0033579	96.134969	0.6316337
0.001	False	0.0262726	0.0034225	96.073619	0.6831612
0.0001	False	0.0262505	0.0034287	96.134969	0.6886485
1e-05	False	0.0262427	0.0034409	96.134969	0.6886485
1e-06	False	0.0262422	0.0034416	96.134969	0.6886485
1e-07	False	0.0262421	0.0034416	96.134969	0.6886485
1e-08	False	0.0262421	0.0034416	96.134969	0.6886485
1e-09	False	0.0262421	0.0034416	96.134969	0.6886485
1e-10	False	0.0262421	0.0034416	96.134969	0.6886485
1.0	True	0.0535575	0.0013524	93.251533	0.5820142
0.1	True	0.0355432	0.0019276	95.460122	0.4508263
0.01	True	0.0290875	0.0027713	95.582822	0.5689336
0.001	True	0.0267342	0.0032315	96.134969	0.6316337
0.0001	True	0.0262847	0.0034102	96.012269	0.6994941
1e-05	True	0.0262432	0.0034427	96.134969	0.6886485
1e-06	True	0.0262396	0.0034458	96.134969	0.6886485
1e-07	True	0.0262393	0.0034461	96.134969	0.6886485
1e-08	True	0.0262394	0.0034461	96.134969	0.6886485
1e-09	True	0.0262405	0.0034465	96.134969	0.6886485
1e-10	True	0.0262392	0.0034461	96.134969	0.6886485

Tabla 6: Train de la base de datos vote con varios eta y L2

η	l2	mean_test_mse	std_test_mse	mean_test_ccr	std_test_mse
1.0	False	0.0452072	0.0018897	95.045871	0.4494476
0.1	False	0.0339000	0.0016117	96.146788	0.3669724
0.01	False	0.0343424	0.0011596	95.963302	0.4494476
0.001	False	0.0350337	0.0016915	95.963302	0.4494476
0.0001	False	0.0351144	0.0017647	95.963302	0.4494476
1e-05	False	0.0351429	0.0018054	95.963302	0.4494476
1e-06	False	0.0351433	0.0018054	95.963302	0.4494476
1e-07	False	0.0351434	0.0018054	95.963302	0.4494476
1e-08	False	0.0351434	0.0018054	95.963302	0.4494476
1e-09	False	0.0351434	0.0018054	95.963302	0.4494476
1e-10	False	0.0351434	0.0018054	95.963302	0.4494476
1.0	True	0.0525457	0.0010583	93.211009	0.4494476
0.1	True	0.0392837	0.0021479	95.412844	0.8205754
0.01	True	0.0345976	0.0014121	95.330275	0.0
0.001	True	0.0344801	0.0011141	95.963302	0.4494476
0.0001	True	0.0350580	0.0017108	95.963302	0.4494476
1e-05	True	0.0351393	0.0017565	95.963302	0.4494476
1e-06	True	0.0351683	0.0017938	95.963302	0.4494476
1e-07	True	0.0351694	0.0017946	95.963302	0.4494476
1e-08	True	0.0351694	0.0017947	95.963302	0.4494476
1e-09	True	0.0351713	0.0017937	95.963302	0.4494476
1e-10	True	0.0351695	0.0017947	95.963302	0.4494476

Tabla 7: Test de la base de datos vote con varios eta y L2

Aquí se puede ver ver como cuando η es menor que 0.001, la diferencia a la hora de reducir o aumentar su valor en los resultados es mínima ya que apenas se tiene en cuenta la regularización, así que realmente no merece la pena. En cuanto a la hora de variar el algoritmo de regularización ,en este caso me ha dado mejores resultados L1 que L2.

η	l2	mean_train_mse	std_train_mse	mean_train_ccr	std_train_ccr
1.0	False	0.0435912	0.0004604	85.4	0.5904591
0.1	False	0.0177410	0.0005942	94.177777	0.3691832
0.01	False	0.0027614	0.0004314	99.755555	0.1911627
0.001	False	0.0001528	1.8117e-5	100.0	0.0
0.0001	False	4.3442e-5	5.7348e-6	100.0	0.0
1e-05	False	3.5951e-5	5.3695e-6	100.0	0.0
1e-06	False	3.5199e-5	5.4298e-6	100.0	0.0
1e-07	False	3.5161e-5	5.4359e-6	100.0	0.0
1e-08	False	3.5156e-5	5.4358e-6	100.0	0.0
1e-09	False	3.5155e-5	5.4358e-6	100.0	0.0
1e-10	False	3.5155e-5	5.4358e-6	100.0	0.0
1.0	True	0.0426650	0.0003429	86.4	0.2287917
0.1	True	0.0256708	0.0003047	91.222222	0.3220305
0.01	True	0.0144451	0.0006853	95.422222	0.5324066
0.001	True	0.0048460	0.0006440	99.266666	0.2948110
0.0001	True	0.0005165	8.7706e-5	100.0	0.0
1e-05	True	2.6635e-5	4.7445e-6	100.0	0.0
1e-06	True	1.5233e-6	4.6631e-7	100.0	0.0
1e-07	True	6.1940e-7	4.1669e-7	100.0	0.0
1e-08	True	5.0457e-7	1.7057e-7	100.0	0.0
1e-09	True	7.2517e-7	4.8604e-7	100.0	0.0
1e-10	True	6.8875e-7	3.5790e-7	100.0	0.0

Tabla 8: Train de la base de datos nomnist con varios eta y L2

η	l2	mean_test_mse	std_test_mse	mean_test_ccr	std_test_ccr
1.0	False	0.0406770	0.0002954	89.066666	0.4422166
0.1	False	0.0250406	0.0002691	90.6	0.5734883
0.01	False	0.0332088	0.0014161	87.666666	0.4714045
0.001	False	0.0364759	0.0013239	87.799999	0.6863753
0.0001	False	0.0379956	0.0015851	87.333333	0.6666666
1e-05	False	0.0380750	0.0016513	87.333333	0.5577733
1e-06	False	0.0381116	0.0017725	87.333333	0.5577733
1e-07	False	0.0380856	0.0017812	87.4	0.5734883
1e-08	False	0.0380857	0.0017813	87.4	0.5734883
1e-09	False	0.0380857	0.0017813	87.4	0.5734883
1e-10	False	0.0380857	0.0017813	87.4	0.5734883
1.0	True	0.0409051	0.0002977	89.666666	0.2981423
0.1	True	0.0269078	0.0002128	91.266666	0.3887301
0.01	True	0.0245010	0.0004526	90.733333	0.2494438
0.001	True	0.0290567	0.0011518	88.999999	0.7601169
0.0001	True	0.0353212	0.0015598	87.6	0.6110100
1e-05	True	0.0384943	0.0015718	86.666666	0.7601169
1e-06	True	0.0403672	0.0015675	86.6	0.6798692
1e-07	True	0.0410286	0.0016568	86.666666	0.5962847
1e-08	True	0.0405408	0.0016771	86.733333	0.7717224
1e-09	True	0.0409378	0.0015904	86.733333	0.5734883
1e-10	True	0.0416692	0.0010444	86.4	0.4422166

Tabla 9: Test de la base de datos nomnist con varios eta y L2

En este caso, sin embargo el resultado que mejor valor me ha dado ha sido con L2 y η=0.1, con el cual se aporta algo de regularización a la regresión logística usando L2. Aquí, usando valores mas pequeños de η si varía algo el resultado pero no significativamente, menos aún cuando sus valores son menores que 1e-06.

2.3. Iniciación del algoritmo k-means: de aleatorio a k-means++

Uno de los puntos principales del algoritmo k-means es la iniciación de los centroides. Actualmente usamos un algoritmo completamente aleatorio para regresión y un algoritmo aleatorio pero estratificado para clasificación. Sin embargo, ¿que pasa si utilizamos un algoritmo de iniciación heurístico como puede ser k-means++? Para ello probaremos como ejemplo los parámetros de los mejores resultados sacados anteriormente y compararemos.

	problema	mean_train	std_train	mean_test	std_test
Random	sin	0.0137999	0.0001082	0.0223010	0.0002921
	quake	0.0282150	5.0900e-5	0.0286885	0.0001244
	parkinson	0.0149830	9.3286e-5	0.0217083	0.0006192
K-means++	sin	0.0138166	0.0001885	0.0221816	0.0002453
	quake	0.0283651	6.2540e-5	0.0284754	0.0002226
	parkinson	0.0139700	0.0002877	0.0198037	0.0004122

Tabla 10: Comparación entre iniciación aleatoria y kmeans++ en regresión

Como se puede ver la diferencia entre kmeans++ y random no es muy notable. Eso es porque aunque inicialice los centroides de una "mejor manera", nada te asegura que los clusters encontrados sean más correctos. Ahora probaremos que tal con los problemas de clasificación.

	problema	mean_train_mse	std_train_mse	mean_train_ccr	std_train_ccr
Random	vote	0.0290820	0.0027380	95.705521	0.5132883
	nomnist	0.0256708	0.0003047	91.222222	0.3220305
Kmeans++	vote	0.0293643	0.0031207	95.521472	0.3680981
	nomnist	0.0238115	0.0002640	92.0	0.1859244
	problema	mean_test_mse	std_test_mse	mean_test_ccr	std_test_ccr
Random	vote	0.0339000	0.0016117	96.146788	0.3669724
	nomnist	0.0269078	0.0002128	91.266666	0.3887301
Kmeans++	vote	0.0348424	0.0054304	96.146788	0.8988953
	nomnist	0.0262591	0.0004338	91.533333	0.3399346

Tabla 11: Comparación entre iniciación aleatoria y kmeans++ en clasificación

De nuevo, la diferencia no es notable dado a que la iniciación del algoritmo aunque influya en el resultado de este, en estos casos ha convergido dando unos resultados parecidos. En este caso, además nuestra iniciación también se podría considerar heurística y más orientada al problema de base, mientras que el kmeans++ busca patrones lo más alejados posibles a la hora de calcular los centroides, obviamente sin tener en cuenta las clases de estos patrones y por tanto, no estratificado. Sin embargo no puedo concluir que uno es mejor que otro dado a que el factor aleatorio afecta mucho al resultado.

2.4. Usando regresión en clasificación

Como última prueba, vamos a probar a utilizar regresión en problemas de clasificación. Para ello, vamos a quitar el flag -c a la hora de ejecutar y

para calcular el CCR usaremos el redondeo al entero más cercano como clase predicha.

	problema	mean_train_mse	std_train_mse	mean_train_ccr	std_train_ccr
Clasificación	vote	0.0290820	0.0027380	95.705521	0.5132883
	nomnist	0.0256708	0.0003047	91.222222	0.3220305
Regresión	vote	0.0467571	0.0032335	95.153374	0.5948073
	nomnist	0.4948683	0.0262497	68.066666	1.5260697
	problema	mean_test_mse	std_test_mse	mean_test_ccr	std_test_ccr
Clasificación	vote	0.0339000	0.0016117	96.146788	0.3669724
	nomnist	0.0269078	0.0002128	91.266666	0.3887301
Regresión	vote	0.0523172	0.0040329	93.944954	1.7977906
	nomnist	0.7167571	0.0105622	60.733333	1.4514360

Tabla 12: Comparación entre usar clasificación y regresión en un problema de clasificación

Como podemos ver, el resultado es claramente inferior al problema en clasificación en nomnist y parecido en vote. En el caso de la base de datos vote es parecido porque tanto las entradas como las salidas son binarias por lo que es probable que los pesos tiendan a los valores 0 o 1 teniendo más en cuenta los valores que hagan 0 o 1 la salida. Sin embargo, en la base de datos nomnist la salida va de 0 a 6 por lo que presupone que una letra 'a' tiene menos valor que una letra 'f', lo cual a la hora de la clasificación no tiene mucho sentido.

2.5. Resultados prácticos en nomnist

Hemos sacado anteriormente como conclusión que la mejor arquitectura es la formada por un ratio_rbf de 0.25, una η de 0.1 y el algoritmo de regularización L2. Tras ver esto, podemos ver los errores cometidos en la matriz de confusión. 1

$$\begin{bmatrix} 47 & 0 & 0 & 2 & 0 & 1 \\ 0 & 47 & 0 & 2 & 1 & 0 \\ 1 & 0 & 46 & 0 & 1 & 2 \\ 1 & 3 & 0 & 46 & 0 & 0 \\ 0 & 3 & 1 & 0 & 43 & 3 \\ 2 & 1 & 0 & 1 & 1 & 45 \end{bmatrix}$$

Figura 1: Matriz de confusión de nomnist

Figura 2: Letras falsamente clasificadas como A

Figura 3: Letras falsamente clasificadas como B

Figura 4: Letras falsamente clasificadas como C

Figura 5: Letras falsamente clasificadas como D

Figura 6: Letras falsamente clasificadas como E

Figura 7: Letras falsamente clasificadas como F

Como podemos ver, la mayor parte de las letras que tienen fallos son letras que son más raras.º que tienen alguna característica que el problema no es capaz de detectar.

Por último vamos a comparar el perceptrón multicapa que hicimos en la práctica 2 y el algoritmo RBF. Para ello utilizaremos la mejor arquitectura

encontrada en ambos casos.En el caso del algoritmo de retropropagación del error con 16 neuronas, MSE y softmax que fue la mejor arquitectura en la anterior practica, con un CCR de test del 84 % y algoritmo ha durado 213.199 segundos, mientras que con RBF ha conseguido un 91.27 % en test y lo ha realizado en 14.288677 segundos.