UNIVERSIDAD
D
CÓRDOBA

INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES

Práctica 2: Perceptrón multicapa para problemas de clasificación

4º Curso-Grado en Ingeniería Informática

UCO-Escuela Politécnica Superior de Córdoba

Manuel Casas Castro - 31875931R

i72cascm@uco.es

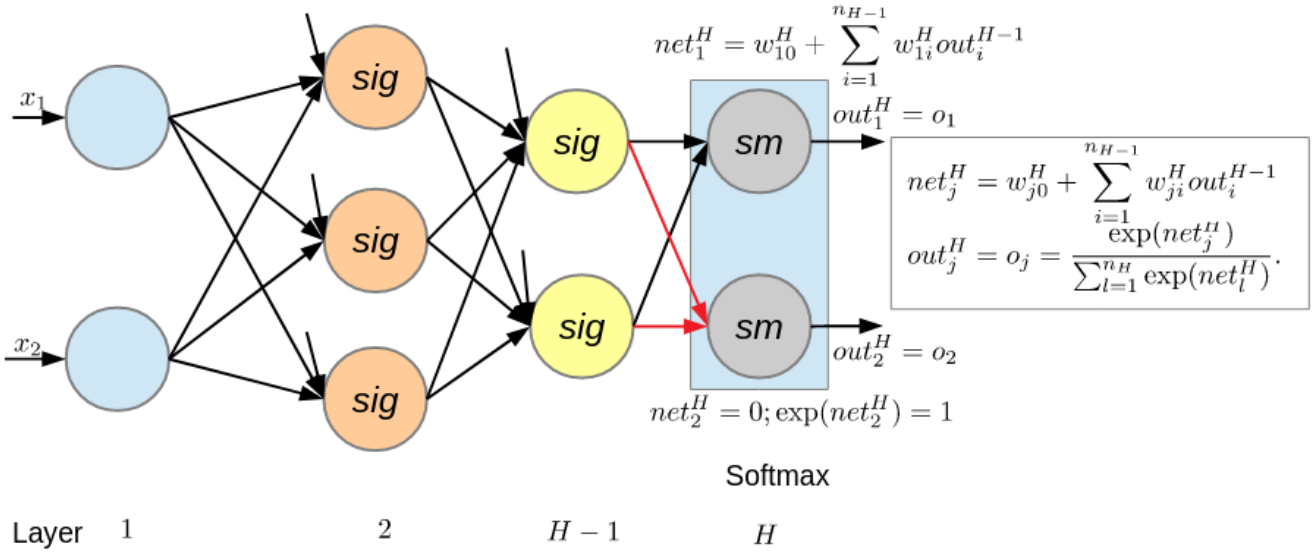
Adaptaciones aplicadas a los modelos de red

Para la realización de esta práctica, partimos del programa empleado para la práctica anterior, donde utilizamos una red neuronal con una versión on-line, donde cada vez que se realiza una época (cada patrón de entrenamiento), se ajustan los pesos de la red neuronal. En esta nueva práctica, añadiremos además la versión off-line, donde los pesos se ajustarán solo cuando se han realizado todas las épocas (todos los patrones de entrenamiento).

De esta manera, con la versión off-line evitamos el problema de “olvidar” los patrones más antiguos que procesó la red neuronal a costa de un mayor coste computacional en caso de que los datasets sean de gran tamaño.

De igual manera, aplicaremos un nuevo tipo de función de salida para la red neuronal llamada función softmax. Este tipo de función nos proporciona resultados consistentes si queremos obtener resultados desde un punto de vista probabilístico, ya que se encarga de normalizar los resultados de salida para que los valores estén comprendidos entre 0 y 1.

Por último, en contraposición al error cuadrático medio (MSE), se ha implementado una nueva manera de calcular el error, llamada entropía cruzada. Esta nueva forma de cálculo se adapta de mejor manera a problemas de clasificación al igual que la función softmax y por tanto, es más óptima para problemas desde puntos de vista más probabilísticos.



Descripción de pseudocódigos de aquellas funciones que han sido modificadas

A continuación se encuentran aquellos pseudocódigos de las funciones más relevantes que han sido modificadas junto a una breve descripción de los mismos:

```
forwardPropagate()
para i desde 1 hasta NumeroCapas
    denominador <- 0
    si i = NumeroCapas-1 entonces
        para j desde 0 hasta capa[i].NumeroNeuronas
            sumatorio <- 0
            sumatorio <- capa[NumeroCapas-1].neurona[j].peso[0]
            para k desde 1 hasta capa[NumeroCapas-2].NumeroNeuronas+1
                sumatorio <- sumatorio + capa[NumeroCapas-1].neurona[j].peso[k] * capa[NumeroCapas-2].neurona[k-1].salida
            fin para
            denominador <- denominador + e^sumatorio;
        fin para
    fin si
    para j desde 0 hasta capa[i].NumeroNeuronas
        net <- 0
        para k desde 1 hasta capa[i-1].NumeroNeuronas+1
            net <- net + capa[i].neurona[j].peso[k]*capa[i-1].neurona[k-1].salida
        fin para
        net <- net + capa[i].neurona[j].peso[0]
        si i = NumeroCapas-1 entonces
            capa[i].neurona[j].salida = e^(net/(1+denominador))
        sino
            capa[i].neurona[j].salida = 1/(1+e^(-net))
        fin si
    fin para
fin para
```

En la siguiente imagen podemos apreciar la función forwardPropagate (sólo la nueva modificación para la función Softmax), donde vamos recorriendo todas las capas realizando las modificaciones de las salidas en cada una de ellas.

Una vez nos encontramos en la última capa, calculamos los denominadores a partir de los valores de la penúltima capa y de esta forma, obtendremos una salida comprendida entre los valores 0 y 1.

```
backpropagateError()
sum <- 0
aux <- 0
para i desde 0 hasta capa[NumeroCapas-1].NumeroNeuronas
    para j desde 0 hasta capa[NumeroCapas-1].NumeroNeuronas
        si i != j entonces
            aux <- 0
        sino
            aux <- 1
        si entropiaCruazada = true entonces
            sum <- sum + (target[j] / capa[NumeroCapas-1].neurona[j].salida) * capa[NumeroCapas-1].neurona[i].salida * (aux-capa[NumeroCapas-1].neurona[j].salida)
        sino
            sum <- sum + (target[j] - capa[NumeroCapas-1].neurona[j].salida) * capa[NumeroCapas-1].neurona[i].salida * (aux-capa[NumeroCapas-1].neurona[j].salida)
        fin si
    fin para
    capa[NumeroCapas-1].neurona[i].delta = -sum
fin para
```

Al igual que en la función anterior, solo mostraremos la parte utilizada en la función softmax en la última capa, ya que en el resto de capas es similar a la práctica 1.

En esta función calculamos los valores de delta en la última capa de la red neuronal teniendo en cuenta si se está utilizando la entropía cruzada o el error cuadrático medio.

Experimentos y análisis de los resultados

Breve descripción de las bases de datos utilizadas:

XOR: Base de datos que representa el problema de clasificación no lineal XOR.

divorce: Contiene 127 patrones de entrenamiento y 43 patrones de test, todas las variables de entrada son numéricas con valores comprendidos entre 0 y 4.

noMNIST: Originalmente compuesta de 200.000 patrones de entrenamiento y 10.000 de test con un total de 10 clases, aunque la utilizada en esta práctica solo tiene 900 patrones de entrenamiento y 300 de test para obtener tiempos de ejecución razonables.

Fase de experimentos y análisis:

A continuación y siguiendo la dinámica de la memoria de la práctica 1, se seleccionarán una serie de arquitecturas para cada una de las bases de datos descritas en el apartado anterior para realizar una serie de ejecuciones variando los distintos parámetros de entrada que recibirá el programa.

Los parámetros de entrada serán indicados en cada una de las tablas de experimentación de las diferentes arquitecturas:

Primera fase de experimentación:

En estas **tres primeras experimentaciones**, consideramos v = 0.0, F = 1, eta = 0.7, mu = 1 y número de iteraciones = 1000.

Comenzaremos probando la mejor arquitectura obtenida para la XOR de la práctica anterior para de esta forma comparar los resultados con la nueva función implementada softmax y el nuevo cálculo del error a través de la entropía cruzada.

XOR:

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
1–64	0.00145999	0.00145999	7.59596e-05	7.59596e-05	100	100

Podemos observar mediante la ejecución con una capa oculta y 64 neuronas (mejor arquitectura con respecto a la práctica anterior), que obtenemos un CCR del 100% tanto para entrenamiento como para test. Esto puede significar que al utilizar tantas neuronas en la capa oculta, la red neuronal esté sobreentrenada y por tanto memoriza patrones.

A modo de prueba, realizaremos un segundo experimento con los mismos parámetros pero reduciendo considerablemente el número de neuronas en la capa oculta:

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
1–2	0.0772413	0.0772413	0.081419	0.081419	80	80

En este segundo experimento y con solo dos neuronas en la capa oculta, obtenemos un CCR del 80%, por lo que se podría confirmar la teoría de que con más neuronas se sobreentrena.

divorce:

A continuación llevaremos a cabo una serie de experimentos con el dataset divorce. Crearemos una tabla con los diferentes valores de número de capas y neuronas por capa, manteniendo los parámetros indicados con anterioridad de manera constante y utilizando la función softmax y error de entropía cruzada.

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
1–4	0.00197699	0.0562971	0.000947623	0.0104677	100	97.6744
1-8	0.00110505	0.0593532	0.0001796	0.0134407	100	97.6744
1-16	0.000853633	0.0570158	7.14096e-05	0.00521804	100	97.6744
1-64	0.000462284	0.0707286	2.74059e-05	0.0108586	100	97.6744
2-4	0.00126133	0.0589791	0.000170181	0.00843529	100	97.6744
2-8	0.000803625	0.0698679	3.3174e-05	0.00791022	100	97.6744
2-16	0.000592622	0.0691495	5.9487e-05	0.0133711	100	97.6744
2-64	0.000320454	0.0846076	1.28841e-05	0.00443166	100	97.6744

Como podemos observar, todos los CCR de entrenamiento y test son idénticos entre sí, aunque los errores medios varían. Podemos observar que aunque a mayor número de capas y neuronas colocamos en la red neuronal, menor error medio obtenemos, pero esto no quiere decir que sea lo mejor, ya que por ejemplo, la arquitectura de dos capas y 64 neuronas es la que menor error medio tiene en entrenamiento pero al mismo tiempo es la que mayor error medio tiene para test.

Podemos concluir entonces que las dos **mejores arquitecturas serían las de 4 neuronas tanto en una como en dos capas**, ya que aumentar el número de neuronas provocaría que la red neuronal se sobreentrenarse y comience a memorizar resultados, resultando así en un mayor error medio en test.

noMNIST:

Realizaremos una tabla idéntica a la del dataset divorce, pero en este caso reduciremos el **número de iteraciones del bucle externo a 500**, ya que al poseer tantos patrones tanto de entrenamiento como de test, podríamos obtener tiempos de ejecución muy elevados.

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
1–4	0.0835899	0.119527	0.0104554	0.0100459	84.8	78.5333
1-8	0.0574654	0.0954254	0.00443211	0.00467814	90.3556	83.5333
1-16	0.0381722	0.0967855	0.00155927	0.0061971	93.9333	84.4667
1-64	0.0092679	0.107377	0.00102218	0.00853923	99.5111	83.4667
2-4	0.136811	0.156991	0.019728	0.0170922	71.3556	66.5333
2-8	0.0697743	0.111703	0.00798964	0.0101454	88.1778	79.6
2-16	0.0370859	0.0986875	0.000936463	0.0118776	94.5111	81.8
2-64	0.00762651	0.106999	0.000610095	0.0129421	99.7333	83.2667

A diferencia de la tabla de experimentaciones anterior, los CCR de esta tabla son diferentes entre sí y podemos concluir que a mayor número de neuronas, obtenemos mejores CCR y errores medios.

Nos quedaremos en este caso con la **arquitectura 1-16** como la que mejores resultados nos ha otorgado, ya que es la arquitectura que mejor puntuación ha obtenido en el CCR test, es decir, ha clasificado de manera correcta más patrones de test.

Segunda fase de experimentación:

Procederemos ahora a buscar la **mejor combinación de función de error y función de activación** para cada una de las mejores arquitecturas de los anteriores experimentos, todos los parámetros de entrada se mantendrán constantes a excepción de aquellos que activan y desactivan dichas funciones. No se realizará la combinación

XOR:

Utilizaremos la arquitectura de una capa y 64 neuronas utilizada en la práctica anterior:

F.Error / F.Activación	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
MSE / Sigmoide	0.000980546	0.000980546	6.42452e-05	6.42452e-05	100	100
MSE / Softmax	0.00136904	0.00136904	5.4799e-05	5.4799e-05	100	100
Entropía / Softmax	0.00145999	0.00145999	7.59596e-05	7.59596e-05	100	100

Para este conjunto de combinaciones, podemos observar que la que mejores resultados nos proporciona es la **combinación MSE/Sigmoide**, ya que aunque todas las funciones tienen un 100% en el CCR, es esta la combinación que menor error medio obtiene en la parte test.

divorce:

Arquitectura de 2 capas y 4 neuronas.

F.Error / F.Activación	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
MSE / Sigmoide	0.0011644	0.0208064	0.000566053	0.0019042	100	97.6744
MSE / Softmax	0.00647804	0.0212897	0.00536799	0.00115201	99.5276	97.6744
Entropía / Softmax	0.00126133	0.0589791	0.000170181	0.00843529	100	97.6744

Al igual que la tabla anterior, podemos observar que los CCR son prácticamente iguales para todas las ejecuciones, aunque cabe destacar que la **combinación MSE/Sigmoide** es la que mejor resultados obtiene, por lo que seleccionaremos esta para próximos experimentos.

noMNIST:

Arquitectura de una capa y 16 neuronas.

F.Error / F.Activación	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
MSE / Sigmoide	0.0349598	0.0492508	0.00101109	0.00203344	88.4	81.9333
MSE / Softmax	0.0443635	0.0546953	0.000914099	0.00271206	84.1111	78.3333
Entropía / Softmax	0.0381722	0.0967855	0.00155927	0.0061971	93.9333	84.4667

Para esta serie de ejecuciones, nos quedaremos con la combinación **Entropía cruzada/Softmax**, ya que es la que notablemente nos ofrece mejores resultados con respecto al CCR.

Tercera fase de experimentación:

Para la mejor combinación encontrada para la fase anterior de experimentación, probaremos una ejecución con todos los parámetros de entrada de manera idéntica tanto para off-line como para on-line.

XOR:

Una capa, 64 neuronas, MSE/Sigmoide.

	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
Off-line	0.000980546	0.000980546	6.42452e-05	6.42452e-05	100	100
On-line	0.250068	0.250068	5.90335e-06	5.90335e-06	55	55

divorce:

Arquitectura de 2 capas, 4 neuronas, MSE/Sigmoide.

	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
Off-line	0.0011644	0.0208064	0.000566053	0.0019042	100	97.6744
On-line	3.643e-06	0.023111	3.9814e-07	6.27753e-05	100	97.6744

noMNIST:

Arquitectura de una capa y 16 neuronas, entropía cruzada/Softmax.

	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
Off-line	0.0381722	0.0967855	0.00155927	0.0061971	93.9333	84.4667
On-line	0.607866	0.640849	0.0387734	0.0449375	33.3556	32

Podemos observar a través de las tablas que solo para el dataset divorce hemos obtenido mejores resultados utilizando una versión on-line.

Podemos observar que para los otros casos, más concretamente el dataset noMNIST, obtiene resultados mucho peores que en una versión off-line y esto se debe a que ajusta los pesos de las neuronas en cada patrón y como consecuencia, a medida que pasan los patrones se van “olvidando” aquellos patrones más antiguos, por lo que la red neuronal está perdiendo mucha información útil a la hora de clasificar.

Cabe destacar que con la versión on-line se ha reducido mucho el costo computacional debido a que las ejecuciones terminaban antes que en la versión off-line debido a la condición de parada de un número de iteraciones sin mejora.

Cuarta fase de experimentación:

En esta última fase, escogeremos la arquitectura mejor obtenida para la fase anterior y modificaremos únicamente los parámetros de factor de decremento y patrones de validación.

A continuación se mostrará una tabla para cada dataset con los resultados obtenidos:

XOR:

Una capa, 64 neuronas, MSE/Sigmoide y versión off-line.

v / F	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
0.0 / 1	0.000980546	0.000980546	6.42452e-05	6.42452e-05	100	100
0.0 / 2	0.00157278	0.00157278	0.000106914	0.000106914	100	100

divorce:

Arquitectura de 2 capas, 4 neuronas, MSE/Sigmoide y versión on-line

v / F	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
0.0 / 1	3.643e-06	0.023111	3.9814e-07	6.27753e-05	100	97.6744
0.15 / 1	4.04374e-06	0.0231105	4.71069e-07	5.79142e-05	99.8425	97.6744
0.25 / 1	3.94224e-06	0.0230742	4.75236e-07	9.04872e-05	99.8425	97.6744
0.0 / 2	4.1185e-06	0.0229381	5.39145e-07	0.000253853	100	97.6744
0.15 / 2	4.40735e-06	0.0229082	6.86479e-07	0.000290798	99.8425	97.6744
0.25 / 2	4.49091e-06	0.0228626	6.00097e-07	0.000386754	99.8425	97.6744

noMNIST:

Arquitectura de una capa y 16 neuronas, entropía cruzada/Softmax y versión off-line.

v / F	Mean_train	Mean_test	SD_train	SD_test	CCR_train	CCR_test
0.0 / 1	0.0381722	0.0967855	0.00155927	0.0061971	93.9333	84.4667
0.15 / 1	0.0379747	0.0984355	0.00241812	0.00589154	93.0667	83.6667
0.25 / 1	0.0385679	0.0978726	0.00266863	0.00717266	92.8	83.8
0.0 / 2	0.0525705	0.102204	0.00197603	0.00335294	91.2889	82
0.15 / 2	0.0526535	0.10472	0.00302841	0.00327725	90.4222	81.8
0.25 / 2	0.0535262	0.105346	0.00278112	0.00489009	89.8444	81.0667

Mediante estas tablas, podemos observar que al aplicar un factor de decremento obtenemos peores resultados de manera general que cuando no se aplica, esto se debe a que el coeficiente de aprendizaje no se mantiene constante en todas las capas y va creciendo en la últimas, por lo que la red neuronal aprende de una manera más natural y tiende menos al sobreentrenamiento.

Bibliografía

https://en.wikipedia.org/wiki/Artificial_neural_network

https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s

https://deepai.org/machine-learning-glossary-and-terms/softmax-layer