UNIVERSIDAD
D
CÓRDOBA

INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES

Práctica 1: Implementación de perceptrón multicapa

4º Curso-Grado en Ingeniería Informática

UCO-Escuela Politécnica Superior de Córdoba

Manuel Casas Castro - 31875931R

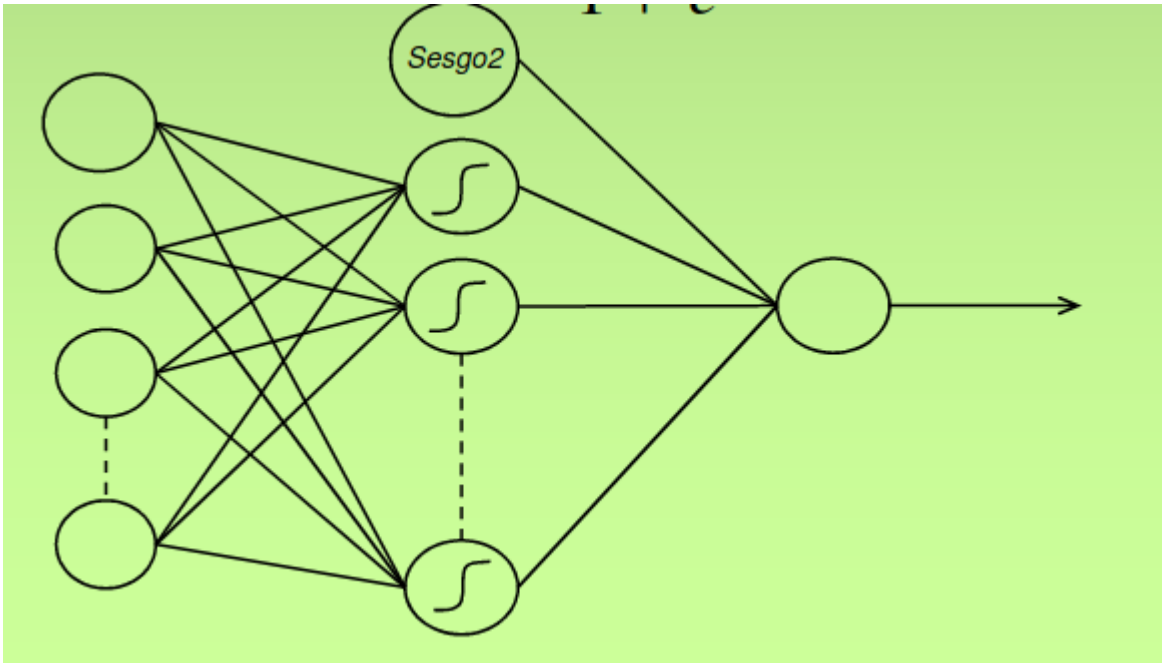
i72cascm@uco.es

Descripción de los modelos de redes neuronales utilizados

Durante la realización de los experimentos (más adelante), se llevará a cabo la implementación de un perceptrón multicapa, donde deberá existir de manera obligatoria en todos los modelos una capa de entrada y una capa de salida.

Respecto al número de capas ocultas utilizadas, estas podrán variar de una única capa oculta a N capas ocultas (siendo N un valor indicado en el apartado de experimentación y que puede variar según el experimento) y donde todas las capas ocultas siempre alojarán el mismo número de neuronas (al igual que el número de capas, este número de neuronas también se indicará en el apartado de experimentación).

Todas las neuronas de una capa deberán estar conectadas con cada una de las neuronas de la capa anterior y posterior donde las neuronas de capa oculta y de salida contarán con una función sigmoide, tal como se indica en la ilustración siguiente (modelo orientativo obtenido de moodle):



Descripción de pseudocódigo más relevante

A continuación se muestran una serie de pseudocódigos, los cuales son los algoritmos más importantes de la implementación:

- forwardPropagate():

```
forwardPropagate()
    for i desde 1 hasta nº de capas
        for j desde 0 hasta nº de neuronas de capa i
            net <-0
            for k desde 1 hasta nº de neuronas capa i-1
                net <- net+(peso de capa i)*(salida de capa i-1)
            net<- net + sesgo
            out <- sigmoide(-net)
```

- feedInputs(input):

```
feedInputs(input)
    for i desde 0 hasta nº neuronas en primera capa
        neurona(i)_out <-input(i)
```

- getOutputs(output):

```
getOutputs(output)
    for i desde 0 hasta nºneuronas de ultima capa
        output(i) <- neurona(i)_out
```

- backpropagateError(target):

```
backpropagateError(target)
    for i desde nº neuronas en ultima capa
        out <- neurona(i)_out
        delta <- -(target(i)-out)*out*(1-out)

    for i desde penultima capa hasta 1
        for j desde 0 hasta nº de neuronas de la capa i
            out <- neuron(j)_out
            aux <- 0
            for k desde 0 hasta nº neuronas de la capa i+1
                aux <- aux + delta(k) * neurona(k)_peso(j+1)
            neurona(j)_delta <- aux*(1-out)*out
```

-weigthAdjustment():

```
weigthAdjustment()
    for i desde 1 hasta nº de capas
        for j desde 0 hasta nº de neuronas de la capa i
            for k desde 1 hasta nº de neuronas de la capa i-1
                neurona(j)_peso(k) <- neurona(j)_peso(k)-eta*neurona(j)_deltaW(0)-mu*eta*neurona(j)_lastDeltaW(k)
```

Experimentos y evaluación de los mismos

Breve descripción de las bases de datos empleadas:

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

seno: Base de datos compuesta por 120 patrones de train y 41 patrones de test con cierto ruido.

quake: Base de datos compuesta por 1633 patrones de train y 546 patrones de test.

parkinsons: Base de datos compuesta por 4406 patrones de train y 1469 patrones de test.

Resultados obtenidos:

Realizaremos una serie de experimentos a cada uno de los datasets dados para la práctica. A cada uno de los datasets durante esta primera fase de experimentos no utilizaremos conjunto de validación ni tampoco emplearemos decremento de la tasa de aprendizaje.

Los argumentos eta (0.1) y mu (0.9) se mantendrán constantes durante estos experimentos, el número de iteraciones será 200 (para no obtener tiempos demasiados altos en el dataset parkinsons).

A continuación se mostrarán una serie de tablas de 12 arquitecturas por cada dataset con los valores de capas y neuronas indicadas en el guión:

quake:

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test
1–2	0.0301768	0.0272903	1.58694e-05	2.31044e-05
1–4	0.0301875	0.0273026	7.1035e-06	1.04521e-05
1–8	0.0301441	0.0272413	2.43228e-05	3.09655e-05
1–32	0.0300886	0.027171	3.38483e-05	4.21745e-05
1–64	0.0300854	0.0271668	2.43334e-05	3.06156e-05
1–100	0.0300252	0.0271107	2.48749e-05	3.0361e-05
2–2	0.0302573	0.0273893	4.61921e-05	4.25216e-05
2–4	0.0301916	0.0273105	5.17651e-06	1.94237e-05
2–8	0.0301813	0.0272887	6.01084e-06	1.62907e-05
2–32	0.0301388	0.0272176	1.40034e-05	1.48367e-05
2–64	0.0300763	0.0271676	2.41018e-05	2.03628e-05
2–100	0.0299244	0.0270968	5.44548e-05	3.54428e-05

Para la base de datos quake, podemos observar que la combinación que mejores resultados obtiene es la 2–100:

A pesar de ser este el mejor resultado, podemos observar que de manera general todas las pruebas han dado el mismo resultado aproximado, por lo que escogeremos como **mejor resultado la combinación 1-100** debido a que su costo computacional es mucho menor.

Mean_train: 0.0300252
Mean_test: 0.0271107

xor:

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test
1–2	0.245227	0.245227	0.00396363	0.00396363
1–4	0.24567	0.24567	0.00416914	0.00416914
1–8	0.234185	0.234185	0.0071204	0.0071204
1–32	0.185283	0.185283	0.00744212	0.00744212
1–64	0.134563	0.134563	0.0159566	0.0159566
1–100	0.237918	0.237918	0.148307	0.148307
2–2	0.250071	0.250071	0.000739219	0.000739219
2–4	0.250012	0.250012	0.000788267	0.000788267
2–8	0.248205	0.248205	0.00187955	0.00187955
2–32	0.168776	0.168776	0.0336397	0.0336397
2–64	0.0387228	0.0387228	0.00492702	0.00492702
2–100	0.0136371	0.0136371	0.00149016	0.00149016

Para esta base de datos si podemos notar diferencias significativas en los resultados obtenidos y podemos notar que el mejor resultado obtenido está en la combinación 1-64.

Mean_train: 0.134563
Mean_test: 0.134563

Aunque también podemos notar que para la combinación 2--100 obtenemos resultados muy parecidos.

sin:

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test
1–2	0.0297475	0.0361852	2.60998e-05	8.66092e-05
1–4	0.0298324	0.0360779	3.57385e-05	8.73945e-05
1–8	0.0299792	0.0360804	8.12236e-05	5.5548e-05
1–32	0.0302764	0.0363159	0.000138517	0.000220819
1–64	0.029456	0.0367986	0.000126038	0.000280903
1–100	0.0295984	0.0373869	0.000105322	0.000744355
2–2	0.0297282	0.0362495	3.80419e-05	0.000132888
2–4	0.0297779	0.0363758	6.092e-05	0.000348377
2–8	0.0299944	0.0362344	0.000189546	0.000396792
2–32	0.0299359	0.0364223	0.00018025	0.000129411
2–64	0.0290837	0.0378101	0.000257747	0.000654073
2–100	0.0298489	0.037735	0.000349502	0.00103882

En esta base de datos obtenemos que la mejor combinación es 2–64 para Mean_train, pero también obtenemos que para Mean_test la mejor combinación es 1–4.

2–64:
Mean_train: 0.0290837
Mean_test: 0.0378101

1–4
Mean_train: 0.0298324
Mean_test: 0.0360779

Para los experimentos siguientes nos quedaremos con la combinación 2–64.

parkinsons:

Capas – Neuronas	Mean_train	Mean_test	SD_train	SD_test
1–2	0.035132	0.0371566	1.26324e-05	4.49836e-05
1–4	0.0293764	0.0293594	0.00186775	0.00211423
1–8	0.0258653	0.0258336	0.000984954	0.0013266
1–32	0.0239208	0.0245381	0.00043354	0.00081656
1–64	0.0224367	0.0230772	0.00202744	0.00237111
1–100	0.0206833	0.0208642	0.000436119	0.000734268
2–2	0.0335309	0.0348675	0.000193332	0.000159954
2–4	0.0284922	0.0285893	0.000821659	0.000944588
2–8	0.0241165	0.0238979	0.00180535	0.00188631
2–32	0.0181132	0.0189998	0.00042134	0.000154103
2–64	0.0152882	0.0161845	0.00100715	0.00157197
2–100	0.0157086	0.0165572	0.00091813	0.000754105

Para esta base de datos obtenemos que la mejor combinación es 2–64 y podemos notar que las combinaciones si dan resultados significativos entre sí.

Debido al costo computacional de este experimento podríamos establecer como **mejor resultado la combinación 2–32** ya que pese a que obtiene peores resultados que la combinación 2–64, el costo computacional es mucho menor.

Mean_train:0.0181132
Mean_test: 0.0189998

A continuación se realizará la siguiente fase de experimentos donde se obtendrán las mejores combinaciones del apartado anterior y se cambiarán los parámetros “v” y “F”:

quake (-l 1 -h 100):

v – F	Mean_train	Mean_test	SD_train	SD_test
0.0–1	0.0300252	0.0271107	2.48749e-05	3.0361e-05
0.15–1	0.0304199	0.0272688	0.000483014	0.000156116
0.25–1	0.0301037	0.0273087	0.00052833	0.000176646
0.0–2	0.0300846	0.0271643	1.96957e-05	2.86213e-05
0.15–2	0.0304708	0.0273078	0.00048978	0.000137973
0.25–2	0.030151	0.0273381	0.000545388	0.000153545

xor (-l 1 -h 64)

v – F	Mean_train	Mean_test	SD_train	SD_test
0.0–1	0.134563	0.134563	0.0159566	0.0159566
0.0–2	0.172225	0.172225	0.0141452	0.0141452

sin (-l 2 -h 64):

v – F	Mean_train	Mean_test	SD_train	SD_test
0.0–1	0.0290837	0.0378101	0.000257747	0.000654073
0.15–1	0.031456	0.039131	0.00584267	0.00255539
0.25–1	0.0315356	0.0388039	0.00606169	0.00235747
0.0–2	0.0292695	0.037628	0.000191974	0.000679585
0.15–2	0.0313909	0.0385953	0.00523831	0.00188513
0.25–2	0.0315845	0.0385098	0.00568415	0.0020216

parkinsons (-l 2 -h 32):

v – F	Mean_train	Mean_test	SD_train	SD_test
0.0–1	0.0181132	0.0189998	0.00042134	0.000154103
0.15–1	0.0179854	0.0189318	0.00152947	0.00122721
0.25–1	0.0192564	0.0201273	0.00167562	0.0014923
0.0–2	0.0216352	0.0216845	0.000701579	0.000691846
0.15–2	0.0232723	0.023601	0.00150279	0.00128498
0.25–2	0.0245672	0.0249498	0.00182891	0.00174393

Podemos observar que a diferencia del anterior experimento, los resultados son muy similares entre sí, pero si podemos notar una gran diferencia de costo computacional respecto a las anteriores ejecuciones debido a la nueva condición de parada al utilizar valores en el argumento -v.

Cuando lleva un número de iteraciones donde “validation error” no mejora lo suficiente, la ejecución finaliza y se queda con el mejor resultado y es por ello que al introducir valores en el argumento -v obtenemos tiempos mucho más bajos.

De igual manera, podemos observar que en todos los casos en los que utilizamos un factor de decremento (F = 2), obtenemos resultados notablemente peores que cuando no utilizamos el factor de decremento en el coeficiente de aprendizaje (F = 1). Esto se debe a que la red neuronal no mantiene constante el factor de aprendizaje, sino que empieza siendo bajo en las primeras capas de la red neuronal e incrementa a medida que avanzamos por las capas.

Esto permite que la red neuronal actúe de una manera más natural para dar lugar a menos sobreaprendizaje y que de esta forma la red neuronal no memorice patrones.

Bibliografía

http://neuralnetworksanddeeplearning.com/chap2.html

https://es.wikipedia.org/wiki/Propagación_hacia_atrás

https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/

https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

https://www.jeremyjordan.me/nn-learning-rate/