Centro de Investigación en Matemáticas A.C.
Programación y Algoritmos 1 (2019)

Tarea 13 Red Neuronal Multicapa

Marco Antonio Esquivel Basaldua

Las redes neuronales artificiales son modelos computacionales inspirados en las redes neuronales biológicas. Las unidades básicas de las estas redes son las neuronas, mismas que se conectan entre sí para enviar señales hasta alcanzar las neuronas de salida de las que se obtiene la información final.

Cada una de las neuronas tiene una función de activación en la que los valores de salida de la o las neuronas anteriores son multiplicados por un valor de peso. Estos pesos pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función limitadora o umbral, que modifica el valor resultado o impone un límite que se debe sobrepasar antes de propagarse a otra neurona.

Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Para realizar este aprendizaje automático normalmente, se intenta minimizar una función de pérdida que evalúa la red en su total. Los valores de los pesos de las neuronas se van actualizando buscando reducir el valor de la función de pérdida. Este proceso se realiza mediante la propagación hacia atrás.

El objetivo de la red neuronal es resolver los problemas de la misma manera que el cerebro humano.

Implementación

En la implementación de la Tarea 13 se cuenta con los siguientes 100 datos, que serán separados en 85 datos para entrenamiento y 15 datos de validación de la red programada.

X1	X2	X3	X4	Y
0	30	150	100	0
0	32	100	90	0
0	24	120	80	0
0	28	135	75	0
0	31	110	95	0
0	28	360	78	0
1	27	390	140	0
0	33	155	102	0
0	35	130	98	0
0	31	220	80	0
1	24	310	105	0
0	29	130	162	0
0	27	220	104	0
0	29	356	155	0
0	35	300	101	0
0	40	210	79	0
0	20	200	160	0
0	15	312	99	0

0 33 201 140 0
1 38 249 83 0
0 40 320 145 0
0 25 158 159 0
0 29 198 92 0
0 31 150 83 0
0 23 155 146 0
0 30 209 78 0
0 41 296 81 0
0 38 204 114 0
0 25 269 130 0
0 36 295 96 0
0 31 150 128 0
0 27 200 133 0
0 24 240 82 0
0 29 190 148 0
0 48 600 200 1
0 52 580 220 1
0 52 598 215 1
0 53 521 170 1
0 59 478 190 1
0 47 400 175 1
0 49 428 182 1
0 53 486 215 1
0 61 507 222 1
0 47 420 185 1
0 54 460 226 1
0 48 420 175 1
0 60 600 176 1
0 47 585 188 1
1 25 300 160 0
1 34 180 84 0
1 23 190 120 0
1 26 148 141 0
1 30 167 92 0
1 31 254 94 0
1 23 304 80 0
1 41 287 82 0
1 22 268 140 0
1 41 260 154 0
1 22 195 115 0
1 25 308 92 0
1 31 170 81 0
0 38 300 148 0
1 15 152 162 0
1 40 197 88 0
1 56 480 190 1
1 60 520 170 1
0 50 540 170 1
1 54 400 210 1
1 49 410 180 1
1 60 535 175 1

1 60 485 190 1
1 52 475 210 1
0 49 600 212 1
0 49 563 220 1
1 51 419 170 1
1 48 501 200 1
1 60 585 230 1
1 59 600 170 1
1 51 590 180 1
1 58 560 200 1
1 56 432 185 1
1 49 400 210 1
0 60 586 225 1
1 50 545 171 1
1 48 590 190 1
1 49 430 201 1
1 58 445 180 1
1 55 570 220 1
1 58 550 204 1
1 57 496 236 1
0 55 400 180 1
1 53 480 174 1
1 48 525 196 1
1 48 600 182 1
1 45 500 231 1
1 60 547 210 1
1 60 460 212 1
1 57 420 184 1
0 54 600 170 1
1 58 570 213 1

Una vez separados, se procede a utilizar los datos de entrenamiento generando valores de pesos y el valor del bias de manera aleatoria con valores en el intervalo $[-1, 1]$. Es importante mencionar que para esta implementación no se hace uso de la propagación hacia atrás, para generar los valores de pesos apropiados para la red neuronal se genera por 10000 iteraciones valores de pesos y de bias aleatorios para cada una de las neuronas presentes en la red de acuerdo a la arquitectura de la misma. Se conserva la colección de pesos que haya generado el menor error sobre los datos de entrenamiento. un criterio adicional para encontrar los pesos adecuados es detener las iteraciones cuando se haya alcanzado un error del 0%.

Clases

La implementación de este programa se realiza implementando la programación orientada a objetos, por lo que se generan clases. Las clases generadas se presentan a continuación.

Clase	Descripción	Entradas	Salidas
sigmoidNeuron	Genera una neurona que es la unidad base de la red neuronal.	Dos vectores de valores dobles. El primer vector corresponde a las entradas de la neurona (x), el segundo corresponde a los pesos de cada una de las entradas (w). El vector de pesos tiene un elemento más que el vector de entradas ya que se trata del valor del bias (b).	Se obtiene una sola salida de esta clase que se obtiene al aplicar la función sigmoidal partir de los datos de entradas (x), sus pesos (w) y el bias (b) $outpus = \frac{1}{1+exp(-\sum_j w_j x_j - b)}$
Layer	La clase *Layer* consta de un vector de neuronas tipo *sigmoidNeuron*. Cada objeto instanciado con esta clase constituye una capa en la red neuronal.	Un valor entero($numNeurons$) indicando el número de neuronas presentes en esa capa. Un vector de dobles (x) con las entradas para cada una de las neuronas (cada neurona recibe las mismas entradas). Un vector de vectores de dobles (w) que corresponde a los pesos para cada una de las neuronas, el tamaño de este vector coincide con el número de neuronas en la capa.	Al generar cada una de las neuronas solicitadas y cada una de ellas calcular su valor de salida con los paràmetros (x y w) ingresados se genera un vector ($outputs$) con cada una de las salidas de cada neurona.

Clase	Descripción	Entradas	Salidas
Net	La clase *Net* consta de un vector de capas tipo *Layer*. Cada objeto instanciado con esta clase constituye una red neuronal.	Un vector de enteros ($architecture$) indicando la arquitectura de la red neuronal, cada valor en este vector corresponde a la cantidad de neuronas por capa, por ejemplo para el vector {4,5,1} se genera una red de tres capas con 4, 5 y 1 neurona por capa respectivamente. **Es importante que el ultimo valor se 1 ya que es la capa de salida** que para este caso solo se arroja un dato de salida. Un vector de vectores de dobles ($trainData$) con los datos usados para el entrenamiento de la red neuronal. Un vector de vectores de dobles ($valiData$) con los datos usados para la validación de los resultados.	Esta clase despliega y guarda en un archivo de texto los pesos usados en la validación de los resultados así como el porcentaje de error en la etapa de entrenamiento.

Resultados

Para una ejecución del código se obtienen los siguientes resultados para una red neuronal con arquitectura {4,5,1}, es decir, de tres capas con 4 neuronas en la primera de ellas, 5 en la segunda y 1 en la capa de salida.

Antes de separar los datos en datos de entrenamiento y datos de validación se realiza un *shuffle* entre las filas de la matriz dada de datos, esto se hace para que en cada ejecución del código se trabaje con datos distintos y no exista un patrón constante entre los datos de entrenamiento y los de validación. Una vez aplicado el *shuffle* se obtiene lo siguiente:

Datos de entrenamiento

X1 X2 X3 X4 Y
0 36 295 96 0
1 22 195 115 0
1 58 550 204 1
0 48 420 175 1

1 23 304 80 0
1 57 420 184 1
1 52 475 210 1
1 27 390 140 0
0 29 198 92 0
1 58 570 213 1
0 40 210 79 0
0 30 150 100 0
1 60 520 170 1
1 41 260 154 0
0 53 486 215 1
1 59 600 170 1
1 49 400 210 1
1 41 287 82 0
1 58 560 200 1
0 29 356 155 0
0 31 220 80 0
0 33 155 102 0
0 31 150 83 0
1 40 197 88 0
1 51 590 180 1
0 27 220 104 0
1 53 480 174 1
1 23 190 120 0
0 60 600 176 1
0 47 585 188 1
0 38 204 114 0
0 61 507 222 1
0 29 190 148 0
0 27 200 133 0
0 35 130 98 0
0 54 460 226 1
0 38 300 148 0
0 41 296 81 0
1 48 590 190 1
0 48 600 200 1
1 60 485 190 1
0 55 400 180 1
0 33 201 140 0
0 53 521 170 1
1 60 460 212 1
1 25 300 160 0
1 48 501 200 1
0 49 600 212 1
0 49 563 220 1
1 49 410 180 1
0 47 420 185 1
0 52 598 215 1
0 24 240 82 0
0 28 360 78 0
0 60 586 225 1
1 60 535 175 1

0 31 150 128 0
0 59 478 190 1
1 48 525 196 1
1 26 148 141 0
1 34 180 84 0
0 32 100 90 0
0 23 155 146 0
1 57 496 236 1
0 28 135 75 0
1 45 500 231 1
0 35 300 101 0
0 40 320 145 0
0 30 209 78 0
0 15 312 99 0
1 54 400 210 1
0 24 120 80 0
0 54 600 170 1
1 55 570 220 1
0 20 200 160 0
0 29 130 162 0
1 56 432 185 1
1 15 152 162 0
1 25 308 92 0
1 48 600 182 1
0 50 540 170 1
1 56 480 190 1
1 24 310 105 0
0 25 269 130 0
0 52 580 220 1

Datos de validacion

X1 X2 X3 X4 Y
1 51 419 170 1
0 31 110 95 0
1 58 445 180 1
1 49 430 201 1
0 49 428 182 1
1 31 170 81 0
1 22 268 140 0
0 25 158 159 0
1 31 254 94 0
1 60 547 210 1
1 60 585 230 1
1 50 545 171 1
1 38 249 83 0
0 47 400 175 1
1 30 167 92 0

Una vez aplicada la búsqueda de los pesos apropiados para la red se obtiene lo siguiente, donde cada fila representa los valores de los pesos para cada neurona recordando que el último de estos datos es el valor del bias de la neurona. Al trabajar con una arquitectura {4,5,1} se tienen en total 10 neuronas.

-0.603 -0.762 0.422 -0.073 0.937
-0.985 0.298 -0.996 0.726 0.577
0.816 0.773 0.874 0.462 -0.855
-0.82 -0.494 0.425 0.399 0.563
-0.366 0.601 0.486 0.999 0.171
-0.496 0.826 0.802 -0.68 -0.75
0.508 -0.726 -0.512 -0.07 0.202
0.982 -0.055 -0.944 0.986 -0.773
0.191 0.358 0.557 -0.379 -0.624
0.702 -0.199 -0.117 -0.317 0.757 -0.555

Estos pesos son generados cuando se obtiene un error del 0% en la etapa de entrenamiento.

Al aplicar los pesos generados en el set de datos de validación los resultados son los siguientes:

X1	X2	X3	X4	Y(calculada)	Y(real)
1	51	419	170	1	1
0	31	110	95	0	0
1	58	445	180	1	1
1	49	430	201	1	1
0	49	428	182	1	1
1	31	170	81	0	0
1	22	268	140	0	0
0	25	158	159	0	0
1	31	254	94	0	0
1	60	547	210	1	1
1	60	585	230	1	1
1	50	545	171	1	1
1	38	249	83	0	0
0	47	400	175	1	1
1	30	167	92	0	0

Se logra un error del 0% en los datos de validación.

Observaciones

Debido a la cantidad limitada de datos con los que se cuenta, se decidió repartirlos solo en dos grupos: el de entrenamiento y el de validación. Esto se consulto previamente con uno de los ayudantes.

Aunque en el ejemplo mostrado en este reporte se logra un error del 0%, lo cual quiere decir que se clasifican correctamente todos los datos, se debe tener en cuenta que el proceso para buscar los pesos no es el más recomendado, ya que al trabajar con datos aleatorios estos no garantizan que siempre se obtendrán los pesos indicados y en caso de trabajar con un set de entrenamiento muy grande el tiempo de búsqueda crecerá de la misma forma. Lo ideal es utilizar la propagación hacia atrás.

Para lograr mejores resultados los valores de X1,...,X4 son divididos entre 1000 antes de ser introducidos a la red, lo que logra que todos ellos tengan un valor en el intervalo $[0, 1]$. Se observó que al cargar los datos en la red tal como fueron leídos del archivo de texto de entrada *data.dat*, las salidas de las neuronas de la primera capa de la red son todas 1 o cercanas a 1 lo cual resta variabilidad a las siguientes capas y por consiguiente a la salida.