Práctica Redes Neuronales

Sumit Kumar Jethani Jethani
Carlos Sánchez Velázquez
Grupo 39

Introducción

En esta práctica se nos requiere desarrollar una red neuronal convolutiva para la clasificación de un conjunto de imágenes dividido en diferentes categorías. Siendo recomendable escoger como mínimo el número de clases distintas entre 4 y 7. Cada clase deberá tener un mínimo de 20 imágenes (cuantas más, mejor) para el conjunto de entrenamiento y como mínimo 5 para el conjunto de validación.

Se requiere que la red sea desarrollada usando Python y empleando las librerías de keras, Tensor Flow y numpy para python.

Una vez creado el *dataset* en primer lugar, entrenaremos la red y visualizaremos gráficamente el progreso del *accuracy* tanto del conjunto de entrenamiento como del conjunto de validación. Posteriormente, probaremos con distintas configuraciones de hiper parámetros para escoger la que mejor resultados ofrezca. Finalmente, se procederá a explicar el funcionamiento de la función de pérdida *Categorical cross entropy*.

Nuestra red consiste en la clasificación de las distintas posiciones de fútbol teniendo en cuenta el mapa de calor de los diferentes jugadores durante la temporada. Esta clasificación se realizará en diez posiciones distintas: portero (POR), centrales (DFC), lateral izquierdo (LI), lateral derecho (LD), mediocentro (MC), mediocentro defensivo (MCD), mediocentro ofensivo (MCO), extremo izquierdo (EI), extremo derecho (ED) y delantero centro (DC).

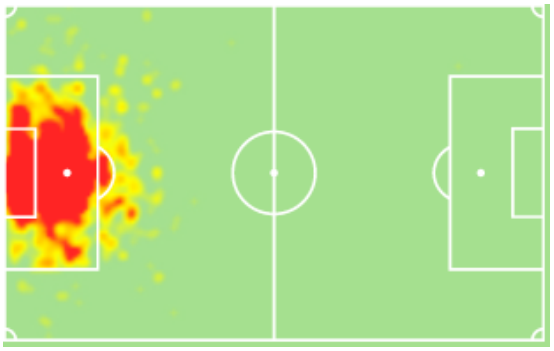
Dataset

Nuestro dataset, se encuentra dividido en dos sub datasets, uno de entrenamiento, que cuenta con 400 mapas de calor (40 por cada posición distinta), con los cuales entrenaremos la red neuronal, y, 200 mapas de calor (20 por cada una de las posiciones) pertenecientes a un conjunto de validación, los cuales nos servirán para corroborar que el modelo realmente es generalizable, y que no se está llevando a cabo un sobreajuste con el conjunto de entrenamiento.

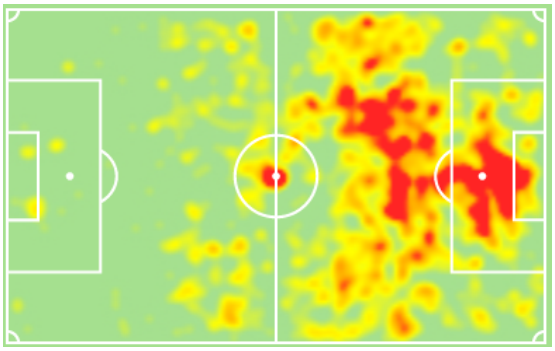
Para recopilar todos estos mapas de calor, nos apoyamos en la página web de Sofascore, la cual reúne una gran cantidad de información y estadísticas de una infinidad de deportes. En nuestro caso concreto, obtuvimos los mapas de calor de diferentes jugadores profesionales de fútbol, los cuales recopilan las diferentes áreas del campo de fútbol, por las cuales se ha movido más un jugador a lo largo de la temporada.

Como podemos ver en las imágenes de ejemplo, estos mapas muestran un campo de fútbol, en el que la portería izquierda es la del jugador, y la derecha es la del equipo contrario. Además, sobre el campo se muestran diferentes manchas que indican las zonas por las que ha jugado más un jugador, siendo el color amarillo para aquellas zonas en las

que el jugador se ha movido, pero en menor cantidad, y aumentando esto a medida que el color de las manchas se acerca al rojo, indicando para este último color las zonas por las que un jugador suele moverse mayoritariamente a lo largo de la temporada.



Thibaut Courtois (POR)



Robert Lewandowski (DC)

Estos mapas, permiten a un humano con conocimientos básicos de fútbol, poder determinar a qué posición corresponde el mapa de calor, solamente en base a la colocación de las manchas a lo largo del mapa. Por lo tanto, el objetivo, es que la red consiga simular esa capacidad humana, y poder determinar la posición en la que ha jugado el futbolista al que corresponde dicho mapa.

Las imágenes de estos mapas de calor, que nos servirán como entradas en nuestra red neuronal, serán imágenes de 150x150 y de x3, ya que al ser imágenes a color, tienen de tres componentes, para la codificación RGB de los colores. Además, destacar que los valores de las componentes RGB se han reescalado, dividiéndolas entre 255, para que sus valores se encuentren entre 0 y 1, lo cual facilita el entrenamiento de la red.

No se ha procedido a realizar la técnica de Data Augmentation por dos motivos principales, en primer lugar, en el dataset es relativamente importante la posición horizontal de las manchas sobre el campo, ya que un lado corresponde con la portería del equipo y el otro la del equipo rival. Por ejemplo, si invertimos el mapa de calor de un lateral izquierdo este pasaría a ser considerado como un extremo izquierdo. Y, por otro lado, si hiciéramos zoom a la imagen, se podrían perder partes del campo de fútbol que pueden ser imprescindibles para la determinación de la posición del futbolista.

Por último. también cabe mencionar que este dataset no existe de manera predefinida en la red, sino que ha sido elaborado “a mano” por los autores de la práctica a base de sacar capturas de la página web mencionada anteriormente.

Proceso de selección del modelo

Modelo	Capas Conv 2D	Capas Densas	dropouts	accuracy	val_accuracy	épocas	lote
Modelo 1	1 de 32	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.8350	0.955	20	5
Modelo 2	1 de 32	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.07	0.135	20	10
Modelo 3	1 de 32	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.9175	0.96	20	2
Modelo 4	1 de 32	1 de 128 1 de 10	2 de 0.25	0.97	0.96	20	5
Modelo 5	1 de 64	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.925	0.965	20	5
Modelo 6	1 de 64	1 de 128 1 de 10	2 de 0.25	0.9425	0.955	20	5
Modelo 7	2 de 32	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.8975	0.955	20	5
Modelo 8	2 de 32	1 de 128 1 de 10	2 de 0.25	0.9075	0.93	20	5
Modelo 9	1 de 32 1 de 64	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.8975	0.945	20	5
Modelo 10	1 de 32 1 de 64	1 de 128 1 de 10	2 de 0.25	0.85	0.935	20	5
Modelo 11	1 de 32 1 de 64	1 de 128 1 de 10	1 de 0.25	0.97	0.95	20	5
Modelo 12	1 de 64 1 de 32	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.8625	0.94	20	5
Modelo 13	2 de 64	1 de 128 1 de 10	1 de 0.25 1 de 0.5	0.9525	0.945	20	5

Como se visualiza en la tabla anterior, hemos partido de modelos más sencillos con una capa convolutiva 2D, ya sea de 32 o 64 neuronas, a las que hemos ido cambiando los hiperparametros como el *dropout* y el tamaño del *lote.* Como se ve en la tabla (filas en rojo), al escoger un tamaño de lote de 10, el *val_accuracy* se queda en un 13.5% lo cual es demasiado bajo para nuestro modelo y podemos ver que el *accuracy* no llega ni al 10% que es el que debería ser estadísticamente ya que tenemos 10 posibles clases en las que clasificar. Por otra parte, al disminuir el tamaño del lote a 2, es cierto que aumenta el *val_accuracy* hasta el 96% pero el tiempo de entrenamiento consumido por este modelo es relativamente mayor respecto al resto de modelos, por lo que directamente descrtamos estos tamaños de lote, y escogimos como 5 el tamaño que usaremos.

Por último, tras haber probado distintas configuraciones de los hiperparametros en un modelo de una sola capa convolutiva, pasamos a probar ahora sobre los modelos que tienen dos capas convolutivas pero cambiando solo los hiperparametros de la *capa convolutiva*, los *dropouts* y dejando el número de *lotes* constante por la razón que mencionamos anteriormente. Tras haber probado varios modelos de 1 capa convolutiva y de dos de ellas, llegamos a la conclusión de que se obtienen mejores resultados con una

capa convolucional (quizás son demasiado complejos, y se ajustan demasiado al entrenamiento), por lo que decidimos que si dos capas convolucionales ya empeoraban los resultados, no seguir añadiendo más, y quedarnos con nuestro mejor modelo hasta el momento.

Modelo seleccionado

El modelo que se ha procedido a escoger para la clasificación de las posiciones de jugadores de fútbol profesional es el siguiente:

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 148, 148, 64)	1792
max_pooling2d_11 (MaxPooling	(None, 74, 74, 64)	0
dropout_12 (Dropout)	(None, 74, 74, 64)	0
flatten_8 (Flatten)	(None, 350464)	0
dense_15 (Dense)	(None, 128)	44859520
dropout_13 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 10)	1290

Vemos que la primera capa del modelo se trata de una capa convolutiva 2D cuya entrada son imágenes de la dimensión 150 x 150 x 3, como mencionamos anteriormente. Esta capa posee 9 pesos de entrada por cada canal: rojo, verde y azul, lo que dan lugar a un total de 27 pesos más un peso que se corresponde con el bias del modelo, lo que dan en total 28 pesos de entrada a la capa. Aparte, en esta capa convolutiva se dispone de 64 filtros, lo que nos devolverá un tensor de 64 x 148 x 148. Por lo tanto, el número total de parámetros de la actual capa es: 28*64 = 1792.

Al tensor de salida de la capa anterior se le aplica la técnica de maxpooling con la que podemos eliminar gran cantidad de datos sin socavar una parte significativa de esta información. Es decir, el tensor de la dimensión 64 x 148 x 148 se consigue reducir en su cuarta parte en otro de 64 x 74 x 74, sin perder información significativa.

Para evitar que la red adolezca de problemas de sobreajuste (overfitting) se procede a emplear la técnica de dropout, que consiste en deshabilitar temporalmente un porcentaje de las salidas tras el maxpooling, en nuestro caso del 25%, para conseguir que todas las neuronas de la siguiente capa deban emplearse a fondo para lograr un buen nivel de *accuracy*.

Posteriormente, se procede a linealizar el último tensor obtenido en un único vector de tamaño 350464, que constituirán las entradas a la capa densa de 128 neuronas generando 44859520 parametros (128 bias + 128*350464) y una salida de un vector de 128. Esta última salida procederá a pasar por otra capa densa de 10 neuronas, debido a las 10 categorías existentes, dando un vector de tamaño 10.

Para las distintas capas hemos empleado como función de activación la llamada “RELU” (*Rectified Linear Unit*) cuya definición es:

$$ReLU(x) = max(0,x)$$

Lo cual significa que si la entrada es un valor mayor que 0, la salida será ese mismo valor. Pero si la entrada es negativa la salida será 0. Esta es una función predilecta en la composición de redes convolutivas ya que ofrece muy buenos resultados empíricos.

Sin embargo, para la última capa se ha empleado como función de activación “*Softmax*” ya que es esta capa la que realiza la clasificación en las distintas categorías. La función softmax es parecida a la sigmoide, ésta también convierte cualquier valor a un valor entre cero y uno. La diferencia está en que la función softmax no trabaja sobre un valor sino sobre un vector. De esta forma, convierte todas las componentes de un vector en valores entre cero y uno, pero, además, garantiza que la suma de todos estos valores sea 1. La expresión de la función softmax es:

$$Softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_j}}$$

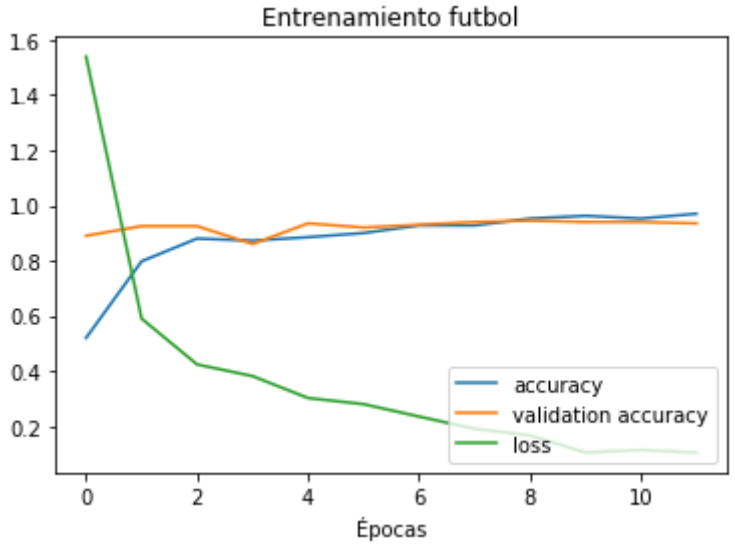
Para el entrenamiento de la red neuronal, hemos probado diferentes tamaños de lote, siendo el que mejor resultados nos ha proporcionado de 5 imágenes por cada lote. Además, hemos establecido en 20 el número de épocas que la red repetirá para entrenarse. Sin embargo, en la mayoría de las ocasiones no se alcanzan las 20 épocas, pues hacemos uso de la técnica de early stopping, de esta forma, desde que durante 3 épocas consecutivas el validation accuracy disminuya, se finalizará el entrenamiento, y se recuperará el modelo que dió los mejores resultados. De esta forma, evitamos sobreentrenar la red.

Finalmente, una vez entrenado el modelo, se guarda este en un fichero, para poder utilizarlo posteriormente para realizar predicciones.

Resultados

Epoch 1/20
80/80 [==============================] - 87s 1s/step - loss: 3.2971 - accuracy: 0.5450 - val_loss: 0.1849 - val_accuracy: 0.9200
Epoch 2/20
80/80 [==============================] - 97s 1s/step - loss: 0.6162 - accuracy: 0.7925 - val_loss: 0.2279 - val_accuracy: 0.9400
Epoch 3/20

80/80 [==============================] - 95s 1s/step - loss: 0.4430 - accuracy: 0.8525 - val_loss: 0.1577 - val_accuracy: 0.9350
Epoch 4/20
80/80 [==============================] - 97s 1s/step - loss: 0.2992 - accuracy: 0.8800 - val_loss: 0.0034 - val_accuracy: 0.9500
Epoch 5/20
80/80 [==============================] - 99s 1s/step - loss: 0.2438 - accuracy: 0.9250 - val_loss: 0.2062 - val_accuracy: 0.9650
Epoch 6/20
80/80 [==============================] - 99s 1s/step - loss: 0.1625 - accuracy: 0.9525 - val_loss: 0.7812 - val_accuracy: 0.9500
Epoch 7/20
80/80 [==============================] - 96s 1s/step - loss: 0.1287 - accuracy: 0.9625 - val_loss: 0.4954 - val_accuracy: 0.9550
Epoch 8/20
80/80 [==============================] - 99s 1s/step - loss: 0.1092 - accuracy: 0.9600 - val_loss: 0.0024 - val_accuracy: 0.9400
Restoring model weights from the end of the best epoch
Epoch 00008: early stopping



Como vemos el modelo ha logrado un *training accuracy* bastante bueno, llegando en ciertas épocas al valor 0.9625. Además, vemos que el modelo es generalizable a cualquier dato distinto a aquellos con los que ha sido entrenado ya que consigue obtener un *validation accuracy* del 96,5%, lo cual nos indica que no existe el problema del sobreajuste ya que no solo se logra un buen resultado en el conjunto de entrenamiento sino también en el de validación. Además, destacar que para guiarnos de que el modelo ha realizado el entrenamiento de forma correcta, podemos observar un continuo descenso en el valor de nuestra función de pérdida, aunque el algoritmo para por el early stopping y recupera los valores con los que obtuvo la mejor validation accuracy. Por último, destacar, que nos faltaría, tras haber elegido definitivamente nuestro modelo, probarlos con un conjunto de test, para conocer el rendimiento real de nuestro modelo, (pues puede que la elección de hiperparámetros realizada sea aquello que mejor se ajuste a nuestro conjunto de validación, por lo que podría estar sesgado en cierta parte ese valor de accuracy con el conjunto de validación.

Confusion Matrix

```
[[20  0  0  0  0  0  0  0  0  0]
 [ 0 20  0  0  0  0  0  0  0  0]
 [ 0  0 19  0  1  0  0  0  0  0]
 [ 0  0  0 20  0  0  0  0  0  0]
 [ 0  0  0  0 19  0  1  0  0  0]
 [ 0  0  0  0  1 19  0  0  0  0]
 [ 0  0  0  0  0  0 19  1  0  0]
 [ 0  0  0  0  0  0  2 18  0  0]
 [ 0  0  0  0  0  0  1  0 19  0]
 [ 0  0  0  0  0  0  0  0  0 20]]
```

Tras el proceso de entrenamiento, es importante visualizar de forma esquemática, y clara los resultados obtenidos. En nuestro caso, hemos usado las librerías de Keras que nos permiten visualizar la matriz de confusión de nuestro modelo para el conjunto de validación.

En esta matriz, se representan en las filas, los jugadores de las 10 distintas posiciones, y en las columnas se representan para cada posición definida por una fila, cuantos jugadores se han clasificado en cada posición. De esta forma, si todos se clasificaran correctamente, todos los valores de la matriz serían 0 salvo la diagonal principal, y en este diagonal habría para cada posición, el número total de jugadores de dicha posición en el conjunto de validación, en nuestro caso 20.

Esta matriz nos permite analizar de forma más profunda los resultados que nos ofrece un modelo, más allá de un simple valor de accuracy sobre el total de casos. De esta forma, podemos determinar, si el modelo presenta problemas de clasificación en alguna categoría en concreto, y, si fuera necesario, tomar medidas correctoras para solventar ese problema, o determinar que son errores de clasificación de no tanta gravedad.

Podemos apreciar que en nuestro problema en concreto, el modelo presenta mayor dificultad sobre todo, a la hora de clasificar jugadores del centrocampo, por ejemplo, de los mediocentros defensivos, a 1 lo clasifica como mediocentro ofensivo, y de los mediocentros uno lo considera mediocentro defensivo, al igual que confunde a 1 mediocentro ofensivo con extremo, errores medianamente razonables por su posición en el campo. Por otro lado, con los extremos también presenta algún fallo de clasificación, por ejemplo, clasificando varios de ellos como mediocentros ofensivos, siendo igualmente errores de clasificación comprensibles, al ser ambas posiciones que juegan en zonas adelantadas del campo.

Classification Report

	precision	recall	f1-score	support
Por	1.00	1.00	1.00	20
Ld	1.00	1.00	1.00	20
Dfc	1.00	0.95	0.97	20
Li	1.00	1.00	1.00	20
Mcd	0.90	0.95	0.93	20
Mc	1.00	0.95	0.97	20
Mco	0.83	0.95	0.88	20
Ed	0.95	0.90	0.92	20
Ei	1.00	0.95	0.97	20
Dc	1.00	1.00	1.00	20
accuracy			0.96	200
macro avg	0.97	0.97	0.97	200
weighted avg	0.97	0.96	0.97	200

Por último, esta librería también nos permite mostrar la matriz de análisis de precisión y recall para nuestro modelo con el conjunto de validación. Entendiendo la precisión como el porcentaje de acierto en nuestras clasificaciones, es decir, a mayor precisión menos falsos positivos. Y el recall entendido como la capacidad de nuestro modelo para identificar todos los casos pertenecientes a una clase, por lo que a mayor recall, menos falsos negativos.

Con esta matriz podemos visualizar la precisión y recall para cada una de las posiciones, y de esta forma identificar aquellas en las que tenemos más problemas con falsos positivos y falsos negativos.

Podemos observar que el modelo tiene unos niveles medios de precisión y recall similares al accuracy obtenido. Sin embargo, como podemos predecir en función de la matriz anterior, las posiciones del centro campo son las que tienen mayores problemas en cuanto a falsos positivos y negativos. De hecho, por ejemplo, para la posición de mediocentro ofensivo detectamos que el modelo solo acierta un 83% de los jugadores que clasifica para dicha posición, siendo la menor precisión de todas las posiciones, por lo que se podría decir que tiene una tendencia mayor a clasificar como mediocentro ofensivo que en el resto de posiciones. Por otro lado, el modelo solo detecta el 90% de los casos reales de extremos derechos, siendo este el menor recall de todos.

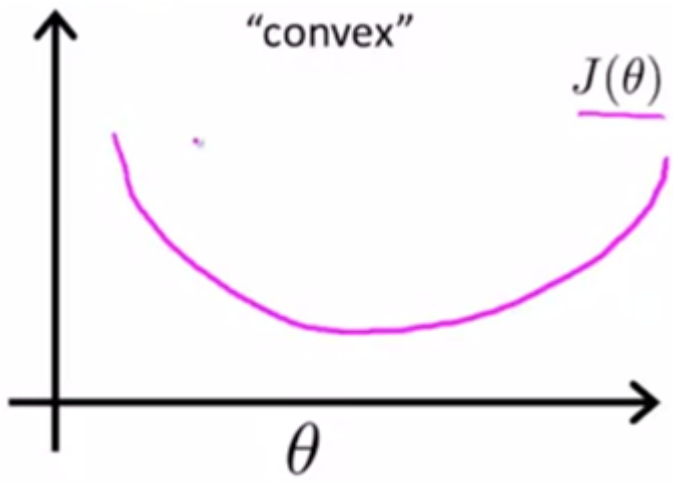
Por último, destacar que esta librería también nos permite analizar el f1-score para cada categoría, el cual es una valoración conjunta de precisión y recall.

Cathegorical Cross Entropy Loss

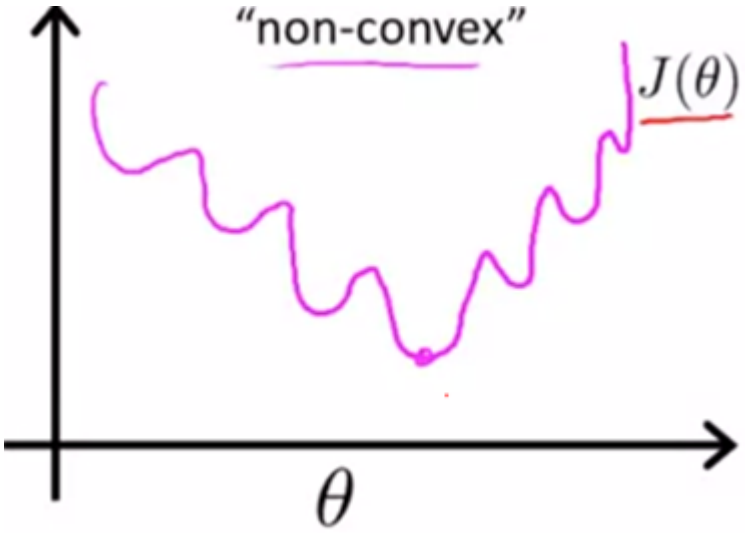
Normalmente la función de error que hemos usado en clase es la media de los errores al cuadrado, tal que así:

$$J(\theta) = \frac{1}{m}\sum_{i=0}^{m}(h(x_i) - y_i)^2$$

Esta fórmula funciona bien para hacer descenso por el gradiente en el caso de regresión lineal, porque al ser una función lineal, la función de coste sería una función convexa, es decir sin mínimos locales, sino un mínimo global, tal que así:

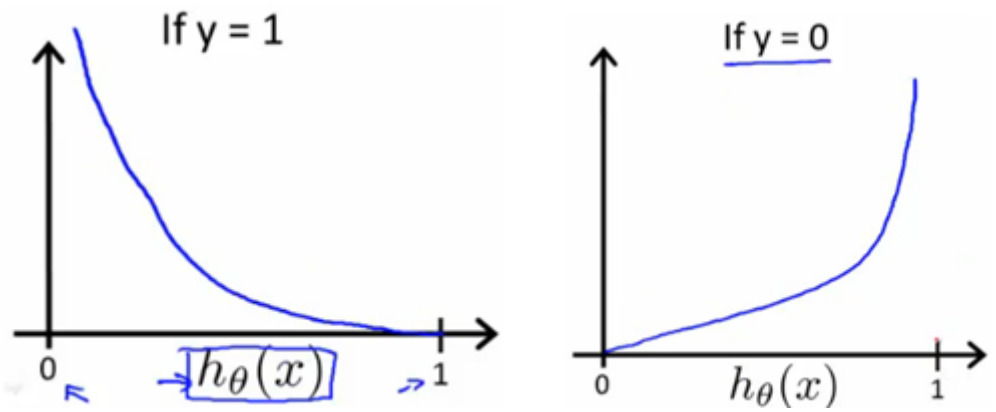


Sin embargo, a la hora de realizar clasificaciones, y usar la función sigmoide, o la softmax (que parte de la sigmoide), al ser esta no lineal, la función de coste tendrá mínimos locales, tal que así:



Es por ello, que se utiliza otra función de coste para los algoritmos de clasificación. Esta fórmula del coste es la siguiente:

$$J(\theta) = \begin{cases} -\log(h(x)) \ si \ \ y = 1 \\ -\log(1 - h(x)) \ si \ y = 0 \end{cases}$$



Es decir, como vemos, cuando la etiqueta es 1, el coste de aquellos resultados a 0 es muy alto, y el de aquellos cercanos a 1 es muy bajo. Y al contrario cuando el verdadero valor es 0. Y vemos que en ningún momento esta fórmula tiene mínimos locales.

Por último, esto se puede simplificar juntando ambos casos en una sola expresión de error:

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m} y_i * \log(h(x_i)) + (1 - y_i) * \log(1 - h(x_i))$$