Back Propagation Neural Network

Sistemas Inteligentes 1

Grado en Ingeniería Informática

Christian García Viguera
Pablo Ortigosa Quevedo
04/11/2021

Índice

Introducción

Esta práctica parte del desarrollo de una red neuronal *back propagation* sin librerías y utilizando *Keras*, y concluye con una comparativa entre ambas.

Para este caso, se ha utilizado un *dataset* de pruebas automovilísticas proporcionado por el profesor. Cada muestra de dicho *set* contiene información sobre la meteorología, las características de la carretera y el vehículo, si hubo accidente o no, etc…

Procesamiento de datos

La información se carga en el programa utilizando la librería Pandas con la función *read_excel()*, que crea un objeto *dataframe* de Pandas.

Cabe destacar que *read_excel()* utiliza un motor obsoleto (el cual dio varios problemas) para leer *.xls*, el formato del dataset. Por ello, el formato del excel fue cambiado a *.xlsx* para poder leerlo con el motor *openpyxl*.

Dado que algunas columnas del *dataset* contienen valores categóricos (no numéricos), debemos formatearlos para que la red neuronal pueda utilizarlos. Esto se consigue también utilizando el método *factorize()* de Pandas.

Tras ello, eliminamos la primera columna, correspondiente a las fechas y horas de las mediciones, dado que no proporcionarían ninguna información útil a la red. Además, algunas filas tenían valores vacíos. Dichos valores fueron convertidos a *nan* y las filas que los contenían fueron suprimidas utilizando la función *dropna()*, también de Pandas.

Finalmente, con el *dataset* listo para introducir en la red, creamos dos *"subdatasets"*: un *dataset* de entrenamiento y otro de validación.

Red Neuronal

A la hora de crear entrenar las redes, tanto la manual como la hecha con *Keras*, se han utilizado dos capas ocultas, con tantas neuronas en cada una como *inputs*.

Las pérdidas son recogidas en cada iteración, tanto en la red con *Keras* como en la manual.

La red de *Keras* se realiza creando un modelo con *Sequential()* y añadiendo capas *Dense.* Todas ellas con una sigmoide como función de activación. Finalmente compilamos el modelo utilizando el optimizador *Adam,* y utilizando error absoluto medio como función de pérdida.

A continuación, se explica el desarrollo de la red manual:

Red manual

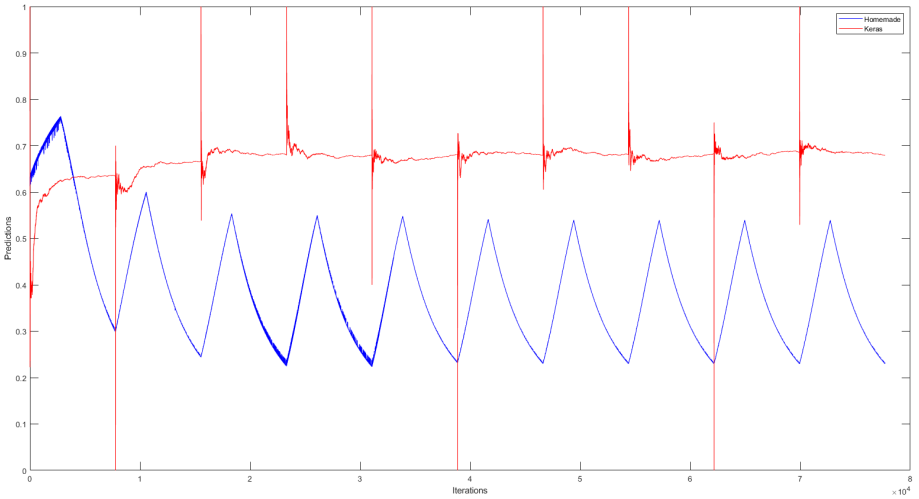
El desarrollo de la red se ha distribuido en cuatro clases: la clase *bpnn*, la clase abstracta *layer*, y las clases *hiddenlayer* y *outputlayer*, que heredan de *layer*.

Para la construcción de dichas clases se ha seguido la descripción del *pdf* "Algoritmo_Aprendizaje_BPNN_v0" (ver código).

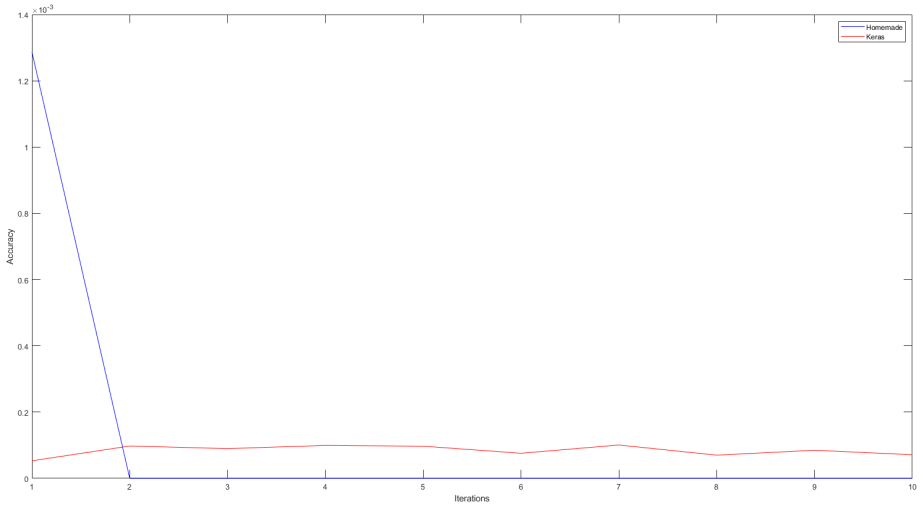
Comparativa

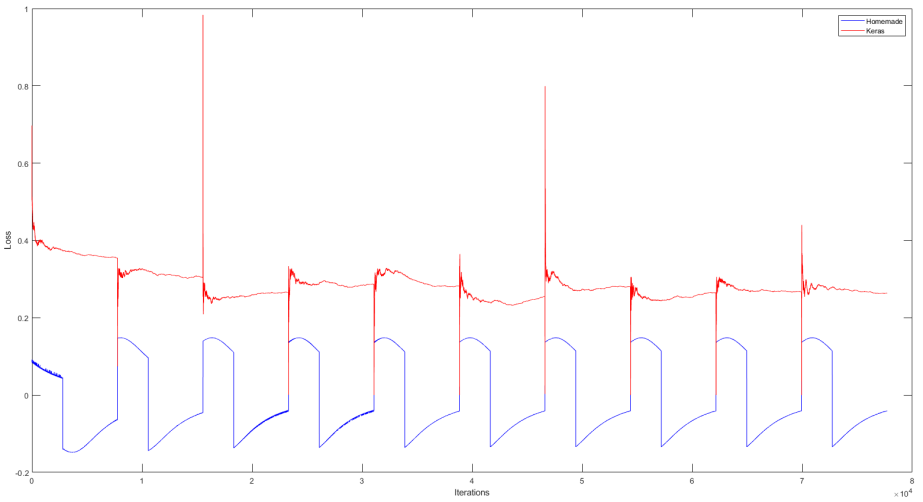
Como se comentará más adelante tenemos "2 arquitecturas" la primera usa la capa output para ajustar los pesos de las capas y la segunda usa la capa siguiente. Hablaremos de estas en el orden que se ha nombrado.

Como podemos observar en las gráficas las predicciones de las red a mano a medida que aumentan las iteraciones son cíclicas, y rondan el mismo valor, por lo que la red aprende a repetir datos, sin embargo la de keras tiene más picos, por lo que esta intenta al menos no repetirse.



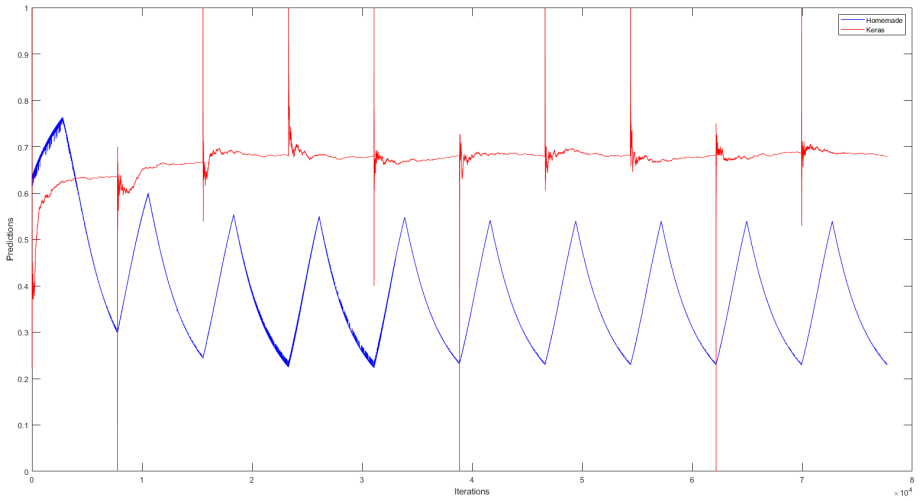
La gráfica de *accuracy* no tiene sentido honestamente, no puede empezar en 100%, al igual que sabemos que no falla absolutamente todo al final porque eso implicaría que simplemente las labels están mal, y sabemos que la red no es muy eficiente, y aunque así fuese es imposible lograr estos resultados en una epoch nada más creemos haber cometido algún error a la hora de calcular estos datos.

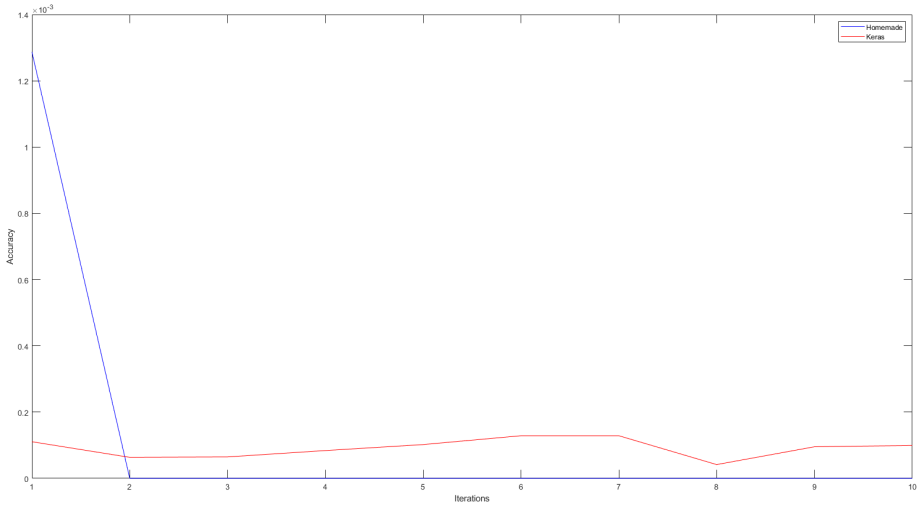


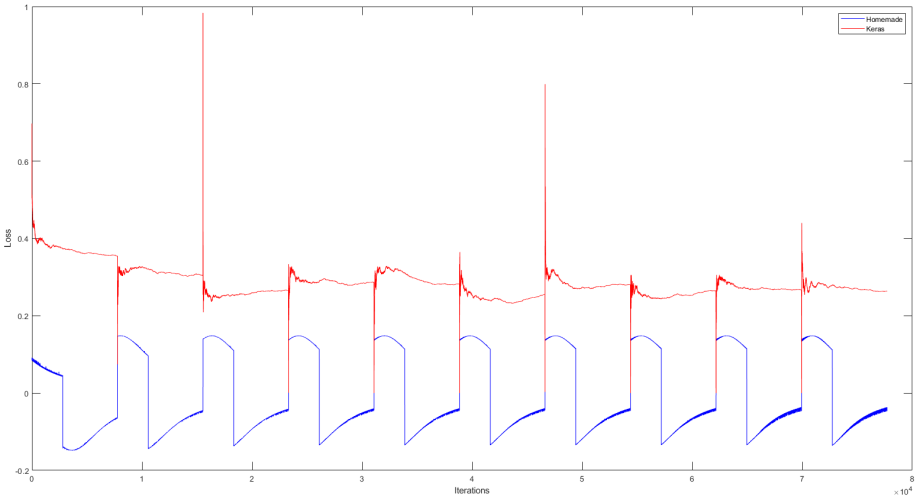


En las siguientes imágenes se muestra el resultado asumiendo que la capa U del pdf se refiere a la capa siguiente. Como se puede ver son bastante parecidos a los resultados anteriores, solo que en este caso las gráficas son más “peludas”. Como hemos dicho anteriormente, la gráfica de *accuracy* no tiene mucho sentido, dado que esta no puede dar cero.

La pérdida sube y baja en la red manual (azul) con cada iteración, como en el caso anterior.


Homemade
Keras
Predictions
Iterations
×10^4


×10^-3
Homemade
Keras
Accuracy
Iterations


Homemade
Keras
Loss
Iterations
×10^4

Conclusión

Creemos haber implementado correctamente la red a mano por lo que creemos que el mayor problema ha sido el preprocesado de datos, así como la recogida de información de la red. Por lo que hacemos incapié en la relevancia del tratamiento de datos, pues hay muchas variables que o bien pueden ser interpretadas incorrectamente, o que no son tan relevantes, pues por ejemplo en condiciones adversas los conductores son consciente de ello y solo los más confiados ya sea con o sin razón se atreverán a conducir, disminuyendo los datos de unas condiciones que claramente dificultan la conducción.

Problemas

El primer problema surge a raíz de la confusión a la que se presta el PDF que proporciona la arquitectura de la red. Al discutirlo con otro grupo no llegamos a ninguna conclusión, así que queríamos preguntarle a usted pero no coincidimos.

Habla de la capa U como si tuviese que ser expresamente la output layer de la red, sin embargo suena más coherente usar la capa siguiente para backpropagation por lo que decidimos aplicar ambas: en la línea *27* de *bpnn.py* hay un valor booleano para indicar la configuración primera hay que poner True y para la otra un False.

Al principio entendimos de lo que se habló en clase que el tratamiento de datos no era tan relevante, sin embargo, ha quedado claro que no es así.