IABD – PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PRÁCTICA EVALUABLE – 5.2 – Iniciación a Keras y R.N.A.s

Vamos a seguir trabajando con los perceptrones multicapa. Esta vez, vamos a entrenar el conjunto iris a través de Tensorflow y Keras. El objetivo es que te familiarices con las sintaxis de las librerías TensorFlow y Keras y juegues con los cambios de los hiperparámetros en una red neuronal artificial. Experimentarás con el formato de los datos para obtener mejores convergencias. También añadirás y borrarás neuronas y capas de neuronas para observar los resultados.

Realiza los siguientes ejercicios en un cuaderno de Jupyter, copiando el enunciado en cajas de texto y a continuación el código que lo resuelve:

1. Carga el conjunto de datos IRIS. Puedes copiar el código de la carga del dataset de una práctica anterior. En la variable X carga la longitud de sépalo y de pétalo de las 100 primeras muestras (setosa y versicolor) y en la variable Y carga las etiquetas de clase verdaderas, siendo un 0 para setosa y un 1 para versicolor. Mezcla el conjunto de datos y normaliza sus características.

2. Basándote en el ejemplo de la práctica 5.1, construye con keras, una RNA secuencial con las siguientes capas:
   - Una capa oculta de 2 neuronas [con capa de entrada, input_shape=(2,)]
   - Una capa de salida de 1 neurona con activación sigmoide activation='sigmoid':

   Piensa en la implicación de todo ello. ¿Qué forma deben tener los datos de entrada a la RNA? ¿Qué forma debe tener la salida de la RNA?

3. Compila el modelo con el optimizador del descenso de gradiente estocástico y función de coste el error cuadrático medio. Muestra el resumen de sus parámetros.

4. Entrena el conjunto de datos con el método fit de tu RNA keras y observa el resultado ¿Crees que el modelo converge? Prueba varias veces con 10, 100 y 1000 épocas. Examina los resultados.

   Muestra los límites de la decisión con la función plot_decision_regions. Ten en cuenta que la función predict devuelve un número entre 0 y 1. Observa la variable Z mediante un punto de ruptura: ¿Qué tienes que cambiar en la función plot_decision_regions para que funcione?

5. Repite los apartados 1-4 esta vez para las variedades versicolor y virgínica. Verás que aunque consiguimos buenos resultados, el modelo está limitado por su linealidad.

6. Para introducir algo de no linealidad podemos utilizar una función de activación en la primera capa llamada “relu”. Añade el parámetro activation=’relu’ en la primera capa y examina los resultados con la función plot_decision_regions. ¿Han mejorado?

7. Modifica el modelo de keras añadiendo más capas, y variando las activaciones y número de neuronas experimentando para ver cuál es el mejor resultado que consigues.

8. Ahora vamos a entrenar el modelo para detectar las tres clases de flores (setosa, versicolor y virgínica) con el mismo modelo. Utiliza una RNA con 4 capas de entrada, 4 neuronas en una capa oculta y 3 capas de salida con

las 150 muestras. Esta vez, en la compilación del modelo, cambia el optimizador a Adam() que te permitirá jugar con el rango de aprendizaje (lr) y la función de coste a categorical_crossentropy. Por ejemplo:

```
model.compile(tf.keras.optimizers.Adam(lr=0.04),'categorical_crossentropy',metrics=['accuracy'])
```

Nota: Usando sólo longitud de sépalo y pétalo tu modelo debería converger con sgd y mean_square_error (eso si, a veces con muchas épocas). Con 4 neuronas de entrada, optimizador sgd y función de pérdida mean_square_error, verás que estos hiperparámetros hace que la RNA NO CONVERJA. Sin embargo, con Adam y categorical_crossentropy sí que converge en 4 variables. Experimenta también con esta situación.

9. Añade una capa densa de 16 neuronas con activación “relu” entre la capa de entrada y la capa de salida y repite el proceso de entrenamiento. ¿Observas alguna mejora en la convergencia del modelo?

10. Realiza alguna predicción y comprueba que, efectivamente, la RNA predice correctamente algunas muestras

11. Normalmente, cuando entrenamos una RNA solemos dividir el conjunto de muestras en dos: Uno para entrenamiento (train) y otro para evaluación (test). Esto es una buena práctica porque evaluamos el modelo con datos que la RNA no ha visto en el entrenamiento. Con la librería scikit-learn, divide el dataset en dos conjuntos: El conjunto de entrenamiento y el conjunto de test. Puedes quedarte con un 20 por ciento de los datos para el conjunto de test. Puedes usar el paquete train_test_split de la siguiente forma:

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train,y_test = train_test_split(X,Y,test_size=0.2,random_state=0)
```

12. Utiliza el método evaluate del modelo de keras para ver cuánto ha aprendido el conjunto. ¿Ves diferencias entre la exactitud (accuracy) del entrenamiento con la evaluación?

13. Haz alguna predicción para ver cómo se genera la salida.

14. ¿Sería posible representar gráficamente los límites de decisión con 4 variables de entrada?

    ¿Y si utilizaramos 3 variables únicamente?