Construyendo una red neuronal para estimar la calidad del vino

Sandra Gómez Gálvez: sandra.gomez.galvez@alumnos.upm.es
Alicia Germán Bellod: alicia.german@alumnos.upm.es

Redes Neuronales y Deep Learning, MUIA, 2020/21

13 de diciembre de 2020

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Descripción del problema

Esta practica tiene como objetivo construir una red neuronal artificial profunda para un problema de clasificación usando el dataset *Wine Quality* para tratar de estimar la calidad del vino en base a pruebas fisicoquímicas como tarea de clasificación. El dataset comprende datos sobre la variante de vino blanco del *Vinho Verde* Portugués. El dataset se puede descargar en https://archive.ics.uci.edu/ml/datasets/wine+quality .

Este trabajo se divide como sigue. En primer lugar, en las siguientes secciones de este capítulo se mostrarán cómo está compuesto el dataset y como se han preprocesado los datos antes de dar paso al proceso de construcción de la red. Seguidamente, en el capítulo 2 se detallará el proceso para buscar el modelo neuronal que mejor clasifique los datos. Finalmente, en el capítulo 3 se mostrarán los resultados del modelo de red neuronal profunda que se ha elegido como mejor clasificador.

1.2. Preprocesamiento de los datos: Dataset inicial

El dataset inicialmente está compuesto por 4.898 instancias con 11 atributos: *acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulfates*, y *alcohol.* Las etiquetas de calidad (salida) son 11 clases que se basan en la calificación de calidad del vino por un experto, estas están entre 0 (horrible) y 10 (excelente). Esta clasificación de salida no es muy buena ya que hay muchos vinos intermedios, por tanto, se agruparán los vinos en menos de 11 clases a continuación.

1.3. Preprocesamiento de los datos: Limpieza y preparación del dataset

Para la realización de este apartado se han utilizado los archivos:

- *WineQualityRawDataset.csv*: Contiene los datos en bruto.
- *PreparingWineQualityDataSet.ipynb*: Archivo de código para limpiar y preparar los datos.

PreparingWineQualityDataSet.ipynb recibe como datos de entrada los datos en bruto del archivo *WineQualityRawDataset.csv*, esos datos los limpia y prepara para devolver como salida los datos de entrada para la red neuronal en un archivo llamado *WineQualityPreparedCleanAttributes.csv* y los datos de salida deseada codificados con una codificación *One-Hot* en un archivo de nombre *WineQualityOneHotEncodedClasses.csv*.

Para ello, primero, se lee el csv de los datos en bruto. Después, se comprueba que no falten valores, y en este caso no faltan. Seguidamente se buscan valores atípicos y se modifican si se encuentran. El proceso continúa verificando cuántas instancias hay por etiqueta y las agrupa si es necesario. En este caso, se han agrupado en tres grupos de calidad: *poor* (pobre), *regular* (regular), y *excellent* (excelente). Más tarde, los datos se han mezclado tres veces para que haya aleatoriedad y entrenar mejor a la red neuronal. A continuación el conjunto de datos se ha dividido verticalmente en los atributos x y en la etiqueta t (target o salida deseada). Así, habrá 11 atributos y una etiqueta de calidad con 3 diferentes valores o clases. Después se ha realizado una codificación one-hot de las clases para el problema de clasificación. En penúltimo lugar, se han escalado los atributos del dataset de entrada con una escala min-max dentro de un rango [-1,1] para cada característica independientemente. Y, finalmente, se han guardado ambos csv.

Tanto el cvs *WineQualityPreparedCleanAttributes.csv* como el *WineQualityOneHotEncodedClasses.csv* serán los que utilicemos para la red neuronal.

Capítulo 2

Proceso de diseño: en búsqueda del mejor modelo neuronal

2.1. Modelo I

Una vez ya con los datos de entrada limpios y extraídos, y con la salida deseada codificada con una codificación *One-Hot*, se ha comenzado a buscar el mejor modelo neuronal que clasifique los datos de la calidad del vino. Para ello se ha utilizado como plantilla el archivo *5_MedianHouseValue_Deep_ANN_Keras.ipynb*.

En primer lugar, se han importado las librerías (*tensorflow, keras, numpy, pandas, matplotlib, tqdm*) y se ha montado el sistema de ficheros para *Google Drive*.

Después, se han cargado los datos de entrada en una variable ATT_FILE y los de la salida deseada o target en una variable LABEL_FILE. Seguidamente se han repartido los datos de entrada y de salida. Un 80 % han sido asignados como datos de entrenamiento, un 10 % de desarrollo, y el 10 % restante para el test final no se asigna aún hasta que se obtenga el modelo deseado, se dejará para más tarde. En este caso el conjunto de entrada de entrenamiento será la variable *x_train* compuesta de 11 atributos y 3918 valores (ejemplos de entrenamiento), y la variable de salida deseada de entrenamiento *t_train* está compuesta de 3 atributos (3 tipos de clasificación) y 3918. Los conjuntos de desarrollo tendrán un tamaño de 11 atributos y 490 valores en las entradas, y 3 atributos (clasificadores) y 490 valores para la salida deseada.

En un tercer apartado inicializamos variables con el número de inputs para el entrenamiento (11) y de outputs (3). También con el número de ejemplos de entrenamiento (3918) y de desarrollo (490). Además comprobamos si los datos son correctos.

Estos datos anteriores no son modificados en ningún momento a la hora de buscar el mejor modelo neuronal, a partir de ahora están fijados. Podrán variar datos que se muestran a continuación.

Los hiperparámetros (número de épocas, tasa de aprendizaje, tamaño del batch, nº de neuronas por capa) son indicados en este cuarto apartado. Para comenzar se han dejado sin tocar los datos de la plantilla con un número de épocas 500, tasa de aprendizaje 0.1, tamaño del batch 500, y nº de neuronas por capa [500, 250, 75, 25].

En el quinto apartado construimos el modelo con una arquitectura de red neuronal completamente conectada 11-500-250-75-25-3 y alimentada hacia delante (*Sequential API*). A continuación se añade la capa de entrada con el número de neuronas de entrada y sin ningún tamaño de batch, se añaden las capas ocultas densas (totalmente conectadas) con cada una de las capas

con su número de neuronas correspondientes y con una función de activación ReLU, y finalmente se añade la capa de salida densa (totalmente conectada) con el número de neuronas de salida y una función de activación Softmax. El código de esta adición de capas se muestra en la figura 2.1. Y un resumen de la arquitectura se puede ver en la figura 2.2. Comprobamos que están todas las capas que hemos añadido y accedemos a los pesos (*weights)* y bias de una de las capas. Inicialmente los bias son un array del tamaño de neuronas de la capa inicializados a 0 y los pesos están inicializados de manera aleatoria.

```
model.add(keras.layers.InputLayer(input_shape=(INPUTS,), batch_size=None))

for neurons in n_neurons_per_hlayer:
  model.add(keras.layers.Dense(neurons, activation="relu"))

model.add(keras.layers.Dense(OUTPUTS, activation="softmax"))
model.summary()
```

Figura 2.1: Código de la adición de capas - Modelo I

```
Model: "DeepFeedforward"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 500)               6000
_________________________________________________________________
dense_1 (Dense)              (None, 250)               125250
_________________________________________________________________
dense_2 (Dense)              (None, 75)                18825
_________________________________________________________________
dense_3 (Dense)              (None, 25)                1900
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 78
=================================================================
Total params: 152,053
Trainable params: 152,053
Non-trainable params: 0
_________________________________________________________________
```

Figura 2.2: Resumen de la arquitectura - Modelo I

En un sexto apartado compilamos el modelo. Con la función *model.compile* especificamos el algoritmo de entrenamiento, en este caso especificamos la función de pérdida, el optimizador a usar y métricas adicionales. Todos los elementos que se han seleccionado aquí son los mismos que la plantilla, se cambiarán para siguientes modelos si es necesario. Como función de perdida (*loss function*) se ha seleccionado la entropía cruzada (*categorical_crossentropy*), como optimizador el descenso del gradiente (*SGD*) y como métrica la precisión (*accuracy*). El código de esta especificación se puede ver en la figura 2.3.

```
model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.SGD(lr=learning_rate),
              metrics=["categorical_accuracy"])

```

Figura 2.3: Algoritmo de entrenamiento - Modelo I

Ya con el algoritmo de entrenamiento creado, entrenamos el modelo en un séptimo apartado. Para ello le introducimos los datos de entrada de entrenamiento *x_train*, los de la salida deseada de entrenamiento *t_train*, el tamaño del batch, el número de épocas (una época es una iteración sobre todo el dataset de entrenamiento) y los datos de validación tanto de entrada como de salida deseada. En este caso ha tardado un total de 17.911332 segundos en entrenar el modelo.

En el octavo apartado obtendremos los resultados. La gráfica de los resultados se muestra en la figura 2.4. Observamos que se produce overfitting (sobreentrenamiento) a partir de la época 120 aproximadamente. En cuanto a los datos de entrenamiento se da una perdida del 62 % y un *accuracy* del 71 %, y en cuanto a los datos de validación una pérdida del 84 % y un *accuracy* del 64 %. Esto se puede ver en la figura 2.5.

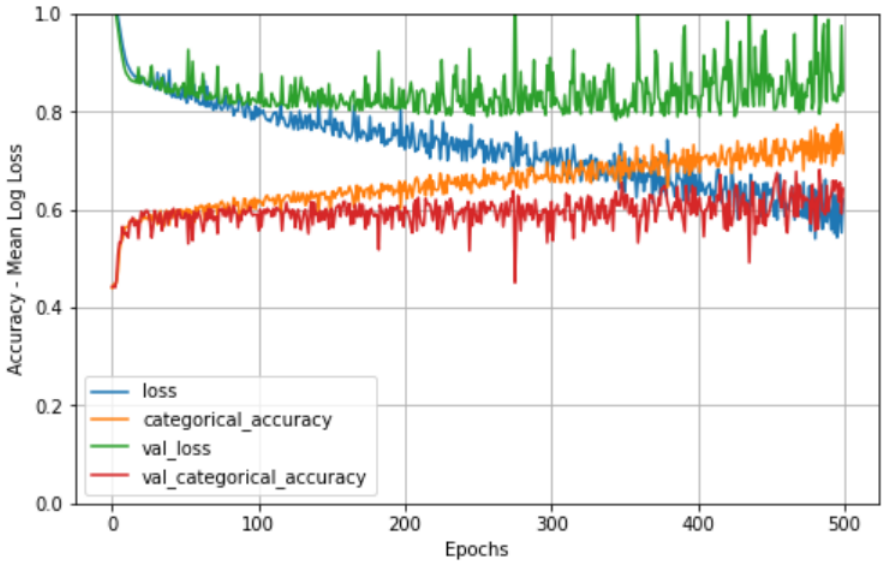

Figura 2.4: Gráfica Resultados - Modelo I

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
499	0.620971	0.715161	0.84249	0.642857

Figura 2.5: Resultados - Modelo I

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.1
Nº de épocas	500
m_b (tamaño del batch)	500
función de activación	ReLU
función de inicialización	No
función de salida	softmax
dropout	No
optimizador	SGD
regularizador	No
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.1: Parámetros - Modelo I

Por tanto, estos resultados nos proporcionan un modelo que no es muy bueno y hay que intentar mejorarlo. Pese a que no sea muy bueno, como es el mejor que tenemos, lo guardamos en un apartado que hemos añadido a la plantilla para tal fin. Como tampoco sabemos si es el mejor modelo no realizaremos por ahora con él el test final. La configuración final de este modelo se puede ver en 2.1.

2.2. Selección de estrategias

Para mejorar el desempeño del modelo nos guiaremos del gráfico que el profesor Martin Molina muestra en la asignatura. Este gráfico que nos ayudará a seleccionar estrategias de mejora se muestra en la figura 2.6.

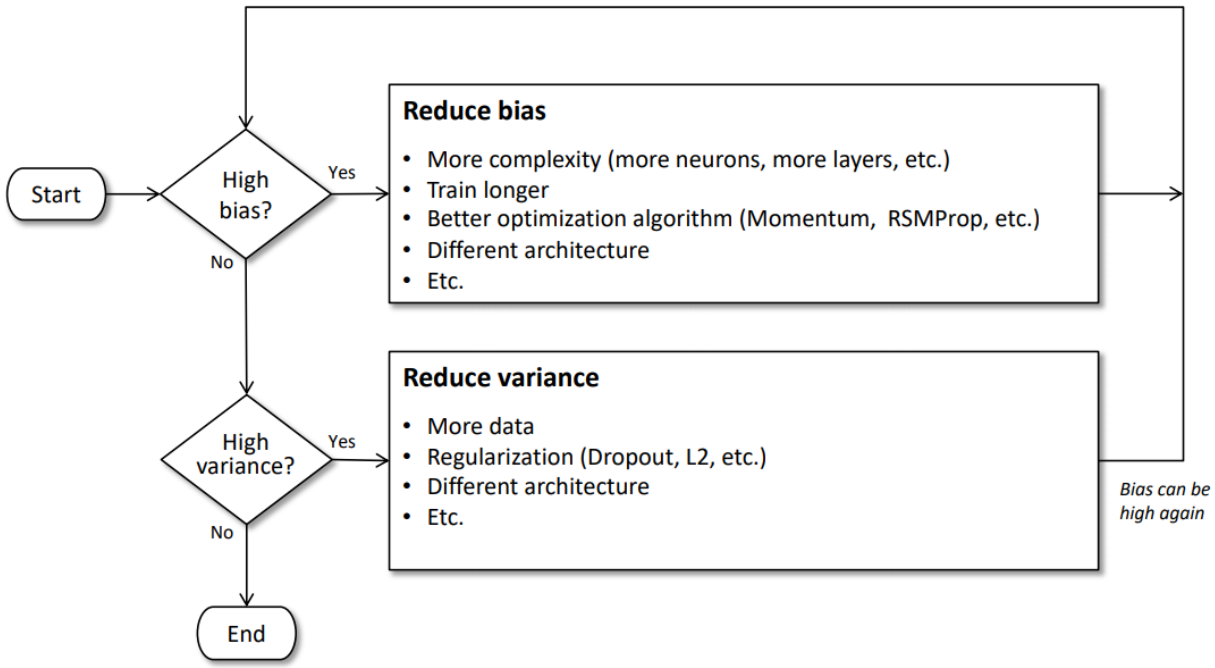

Figura 2.6: Cómo seleccionar estrategias de mejora

Para comprender este gráfico tenemos que tener en cuenta dos conceptos:

- **Bias:** No nos referimos a los bias de cada neurona. En este caso nos referimos a un estimador estadístico que mide el rendimiento. El valor de este bias se estimará mediante: $Bias = E_{entrenamiento} - E_{humano}$. Para este problema asumiremos que el error humano es 0.
- **Varianza:** El valor de la varianza se estimará con $Varianza = E_{desarrollo} - E_{entrenamiento}$.

La situación ideal que se perseguirá para obtener un buen modelo será la de un bias y una varianza bajas.

2.3. Modelo II

En el modelo I se ha observado (figuras 2.4 y 2.5) que hay un alto bias (0.620971) y una alta varianza ($0{,}84249 - 0{,}620971 = 0{,}221519$). Por tanto, guiándonos por el gráfico de selección de estrategias de la figura 2.6, se intentará mejorar en primer lugar el bias alto.
Para reducirlo, se han hecho dos modificaciones secuenciales:

- En cuanto a la arquitectura, se ha incrementado el tiempo de entrenamiento a 2000 épocas, porque se ha observado en la gráfica 2.4 que la precisión (*accuracy*) tiene una tendencia a aumentar si hay un mayor número de épocas y además se observa que el bias tiene una tendencia a descender a mayor número de épocas. También, se ha reducido la tasa de aprendizaje a un $\alpha = 0{,}001$ para que no haya oscilaciones aunque somos conscientes de que el proceso irá más despacio y, por tanto, tendrá una convergencia más lenta. Y,

finalmente, se ha reducido el tamaño del batch a la mitad, $m_b = 250$, para que los pesos sean actualizados más veces y haya una mejora.

- Se ha empleado el algoritmo de optimización *Adam*, en vez del *SGD* que había en un principio. El optimizador *Adam* viene incorporado en la librería keras. Los valores usados para los parámetros son los recomendados en las diapositivas (salvo el *learning rate*). Como sabemos, se incorpora por defecto el parámetro $\epsilon = 10^{-7}$ para asegurar la estabilidad numérica. Estos valores son: $\beta_1 = 0{,}9, \beta_2 = 0{,}999$ y como learning_rate el $0{,}001$ instanciado al cambiar la arquitectura.
  Se ha decidido emplear este algoritmo de optimización debido a que combina los optimizadores *momentum* y *RSMProp*, tomando lo mejor de ambos ya que recoge ambos efectos.

La evolución de la precisión (*accuracy*) en los conjuntos de entrenamiento y desarrollo (*val*), así como de las pérdidas (*loss*) en los mismos, se muestra en la gráfica 2.7.

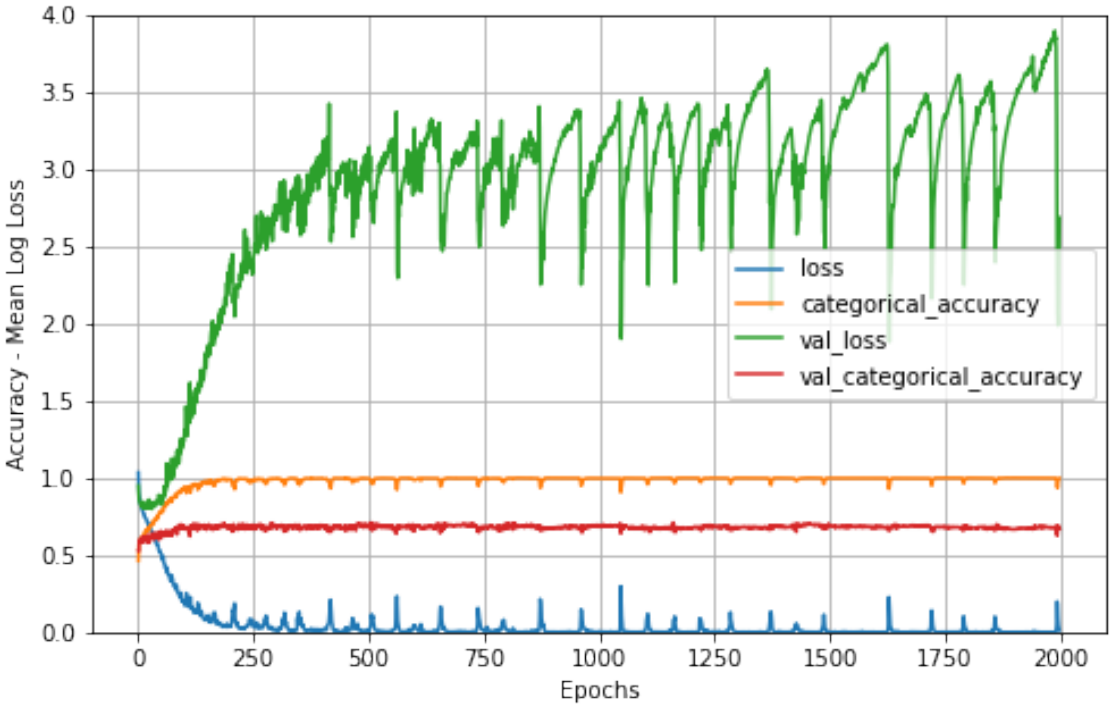

Figura 2.7: Gráfica Resultados - Modelo II

Como podemos observar en la gráfica, se produce un sobreajuste (*overfitting*) de los datos desde etapas muy tempranas (época 20 aproximadamente). Los valores obtenidos de precisión (*accuracy*) y pérdida (*loss*) en los conjuntos de entrenamiento y validación se pueden ver en la figura 2.8.

loss	categorical_accuracy	val_loss	val_categorical_accuracy
0.011683	0.997448	2.686069	0.671429

Figura 2.8: Modelo II: accuracies y pérdidas

Como vemos, con este modelo hemos conseguido bajar mucho el bias, pero la varianza ha aumentado considerablemente. En la tabla 2.2 se muestra un cuadro-resumen de los parámetros e hiperparámetros empleados que han dado lugar a los resultados mostrados.

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	2000
m_b (tamaño del batch)	250
función de activación	ReLU
función de inicialización	No
función de salida	softmax
dropout	No
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	No
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.2: Parámetros - Modelo II

Por tanto, como ya hemos reducido el bias y la varianza ahora se encuentra considerablemente alta, siguiendo el gráfico 2.6 para seleccionar estrategias, en el siguiente modelo intentaremos reducir la varianza.

2.3.1. Pruebas intermedias sin mejoras

En nuestra búsqueda de una red que prediga con la mayor precisión posible las calidades de los vinos en el conjunto de testeo, hemos hecho las siguientes pruebas de configuraciones intermedias, las cuales han dado resultados no satisfacientes.

En nuestro camino de encontrar un modelo bueno que reduzca el bias primero se han realizado los siguientes cambios antes de dar el Modelo II como el bueno.

Para la configuración de neuronas empleada en el modelo II, el resto de optimizadores incorporados en Keras (RMSprop, Nadam, Adamax, Ftrl, etc) han dado resultados similares o peores. En especial, los resultados arrojados por FTrl da unos resultados penosos en cuanto a la precisión y la pérdida en ambos sets de datos (entrenamiento y validación).
También se han probado distintas configuraciones estructurales en cuanto al número de capas y número de neuronas por capa. Los resultados obtenidos han sido, en todo momento, infructíferos, ya que la precisión y la pérdida han sido iguales o peores a los obtenidos hasta ahora.
Por otro lado, se han probado las funciones de activación sigmoid, elu y selu, obteniéndose, otra vez, peores resultados.

Con una función de activación tanh se han obtenido unos resultados similares e interesantes que hemos decidido preservar por si acaso:

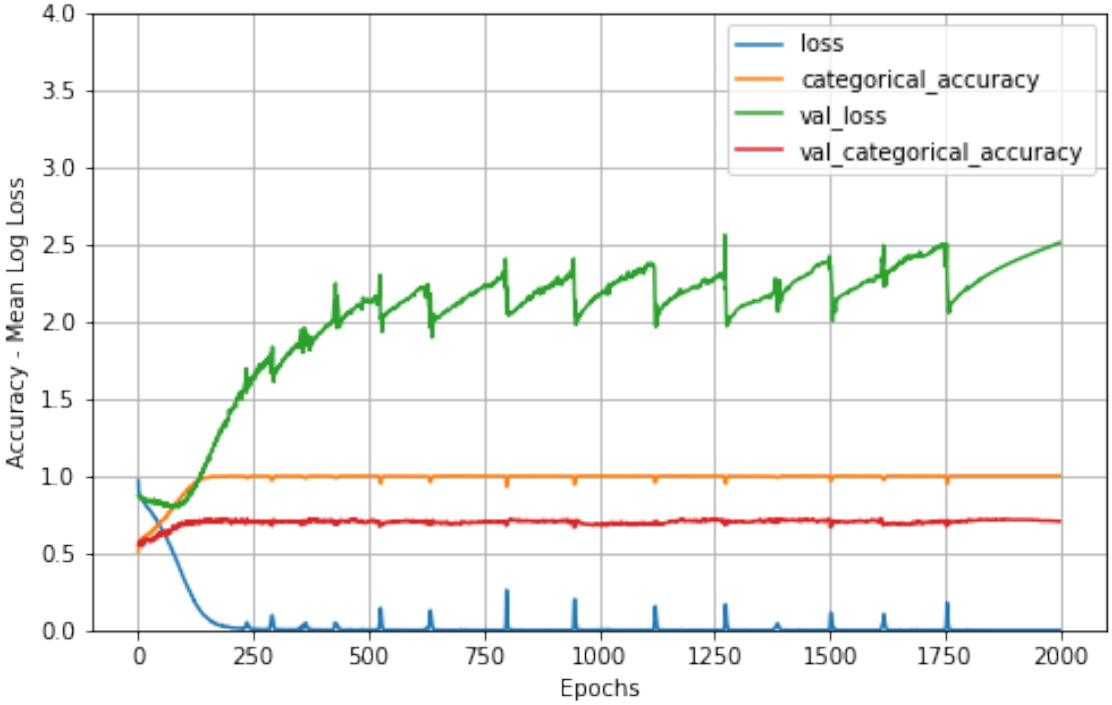

Figura 2.9: Gráfica Resultados - Modelo II modificado con tanh

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
1999	0.000049	1.0	2.510657	0.706122

Figura 2.10: Modelo II modificado: accuracies y pérdidas

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	2000
m_b (tamaño del batch)	250
función de activación	tanh
función de inicialización	No
función de salida	softmax
dropout	No
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	No
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.3: Parámetros - Modelo II con f. activación tanh

Tomando el modelo II como el modelo II modificado con tanh, se ha observado que en

ambos hay una varianza muy alta y como se explica al final del apartado 2.3 se intentará reducir y buscar así el mejor siguiente modelo.

Para ello, en primer lugar se ha intentando aplicar un regularizador para reducir la complejidad de la red y así lidiar con el sobreentrenamiento (*overfitting*).

Los regularizadores para mantener los pesos pequeños l1, l2 y l1-l2 han sido aplicados tanto al modelo II como al modelo II modificado con la función de activación tanh, dando la mayoría resultados nefastos. El único que ha dado un buen resultado, ha sido el l2 aplicado al modelo II sin modificar, empleando un parámetro de regularización $\lambda = 0{,}001$, ya que reduce algo el error en el conjunto de validación a pesar de aumentarlo considerablemente en el conjunto de entrenamiento, como se puede ver en las figuras 2.11 y 2.12.

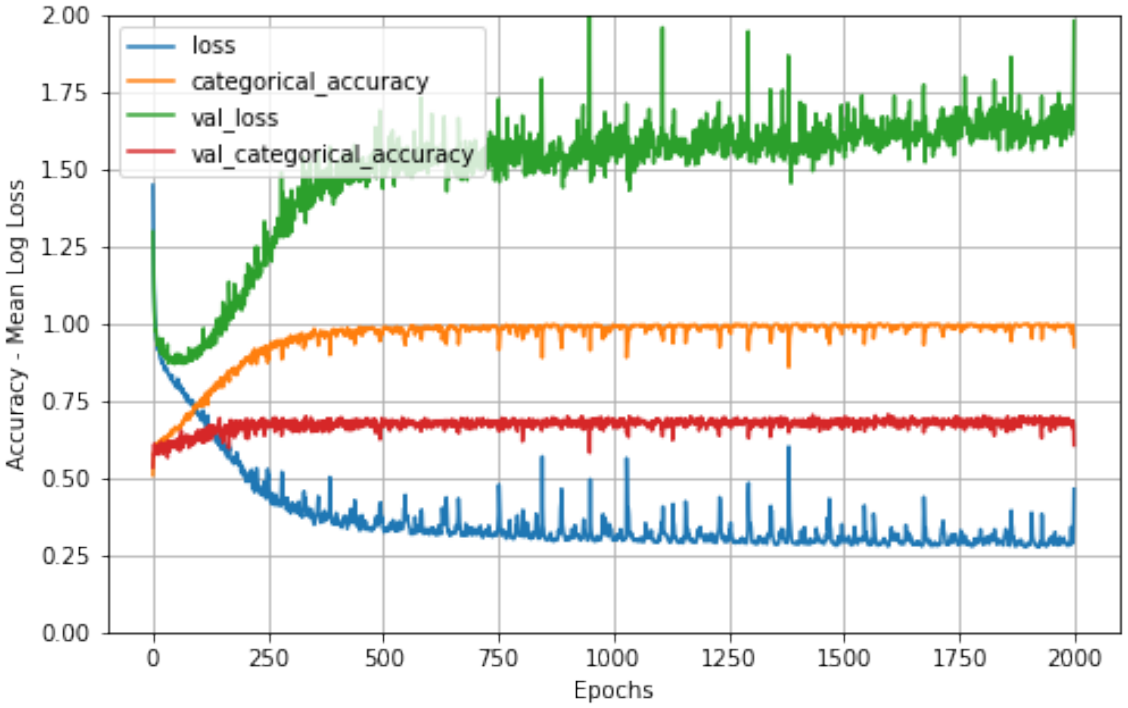

Figura 2.11: Gráfica Resultados - Modelo II con regularización L2

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
1999	0.463713	0.925472	1.980794	0.606122

Figura 2.12: Modelo II con regularización L2: precisión y pérdidas

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	2000
m_b (tamaño del batch)	250
función de activación	ReLU
función de inicialización	No
función de salida	softmax
dropout	No
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	L2
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.4: Parámetros - Modelo II con regularizador L2

2.4. Modelo III

Pese a haber obtenido unos resultados aceptables al aplicar el regularizador L2, se ha decidido aplicar sobre el Modelo II un regularizador de Dropout que ha reducido muchísimo más la varianza en comparación al L2, cual era nuestro objetivo. Por tanto, se ha continuado trabajando con este tercer modelo que mezcla las configuraciones del Modelo II con el dropout. Con el dropout se ha conseguido reducir la complejidad de la red, y con ello la varianza y el sobreentrenamiento (*overfitting*). El objetivo es reducir el número de neuronas, por ello, se ha instanciado una configuración de Dropout $[0{,}8, 0{,}4, 70{,}2, 0{,}1]$. Los valores más altos de dropout se han instanciado en las capas con mayor número de neuronas para conseguir que estas capas reduzcan su número, y así reducir la complejidad de la red. Como se puede observar en la figura 2.13 la probabilidad de eliminar neuronas de las capas ocultas disminuye con el número de neuronas de la capa.

Además, para mejorar el modelo, se han aplicado inicializadores de pesos de tipo *he-normal* para que se inicialicen los pesos de manera aleatoria haciendo que la varianza de los pesos sea $\frac{2}{n}$. Así se evitará que el sumatorio de los pesos por las entradas sea demasiado grande. También se ha probado el inicializador *he-uniform*, obteniendo resultados similares. Por tanto, se ha preferido el inicializador *he-normal*.

Los resultados obtenidos de este modelo se pueden observar en las figuras 2.13 y 2.14.

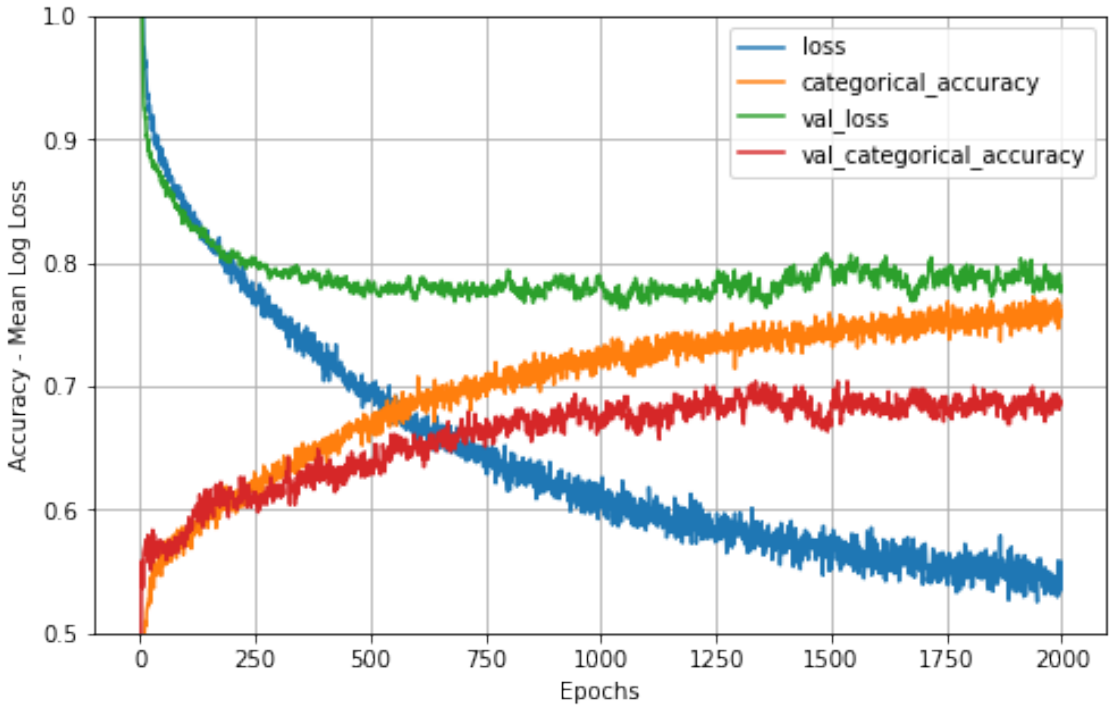

Figura 2.13: Gráfica Resultados - Modelo III

Las precisiones (*accuracy*) y pérdidas (*loss*) en el conjunto de entrenamiento y validación se pueden ver en la figura 2.14.

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
1999	0.535599	0.760592	0.780456	0.685714

Figura 2.14: Modelo III: accuracies y pérdidas

En la tabla 2.5 se resumen los parámetros empleados en este modelo.

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	2000
m_b (tamaño del batch)	250
función de activación	ReLU
función de inicialización	he_normal
función de salida	softmax
dropout	$[0{,}8, 0{,}4, 70{,}2, 0{,}1]$
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	Sí, dropout
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.5: Parámetros - Modelo III

La implementación de las capas es tan sencilla como se indica en la figura 2.15.

```
model = keras.models.Sequential([
keras.layers.InputLayer(input_shape=(INPUTS,), batch_size=None),
keras.layers.Dense(500, activation="relu", kernel_initializer="he_normal"),
keras.layers.Dropout(rate=0.8),
keras.layers.Dense(250, activation="relu", kernel_initializer="he_normal"),
keras.layers.Dropout(rate=0.4),
keras.layers.Dense(75, activation="relu", kernel_initializer="he_normal"),
keras.layers.Dropout(rate=0.2),
keras.layers.Dense(25, activation="relu", kernel_initializer="he_normal"),
keras.layers.Dropout(rate=0.1),
keras.layers.Dense(3, activation="softmax")])
```

Figura 2.15: Modelo III: construcción del modelo

Si analizamos los resultados, se ha conseguido reducir la varianza aunque sigue siendo alta y no se observa a primera vista sobreentrenamiento (*overfitting*). Además el bias a aumentado considerablemente desde el Modelo II.

Por tanto, el nuevo objetivo será reducir el bias como se indica en el gráfico 2.6 de selección de estrategias.

2.4.1. Pruebas intermedias sin mejoras

Se ha probado lo mismo que en el Modelo III pero con función de activación *tanh* y con inicializador *Xavier* (ya que suelen emplearse conjuntamente), pero, como se puede observar en la gráfica 2.16, los resultados arrojados por esta combinación son algo peores.

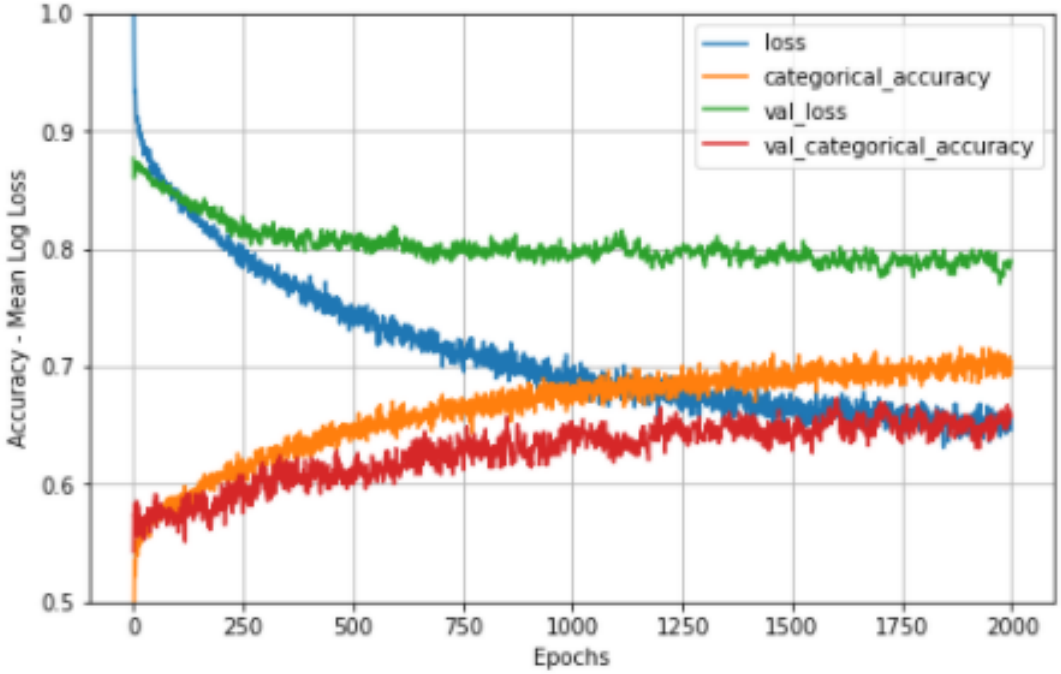

Figura 2.16: Gráfica Resultados - Modelo III modificado

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
1999	0.646326	0.703931	0.790117	0.657143

Figura 2.17: Parámetros - Modelo III modificado

La tabla de parámetros asociada a esta configuración se puede observar en la tabla 2.6. Donde la función de inicialización *GlorotNormal* es el inicializador *Xavier*.

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	2000
m_b (tamaño del batch)	250
función de activación	tanh
función de inicialización	GlorotNormal
función de salida	softmax
dropout	$[0{,}8, 0{,}4, 70{,}2, 0{,}1]$
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	Sí, dropout
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.6: Parámetros - Modelo III modificado

Por tanto, el modelo que seguimos seleccionando como mejor es el Modelo III sin ninguna otra modificación, y el objetivo sigue siendo reducir el bias.

2.5. Modelo IV

En este modelo, la única modificación realizada ha consistido en incrementar el número de épocas ($n_{epochs} = 8000$). Porque se ha observado en la gráfica 2.13 que la precisión (*accuracy*) tiene una tendencia a aumentar si hay un mayor número de épocas, también se observa que el bias tiene una tendencia a descender a mayor número de épocas, y además que es altamente probable que no se cree sobreentrenamiento (*overfitting*) por la tendencia, podría decirse, constante del error de validación. De este modo, mejoramos un poco tanto el bias como el *accuracy*, pero la varianza sigue siendo alta (aunque reducir la varianza no es nuestro objetivo actualmente por eso este modelo será nuestro nuevo mejor modelo).

Los resultados se pueden visualizar en las figuras 2.18 y 2.19.

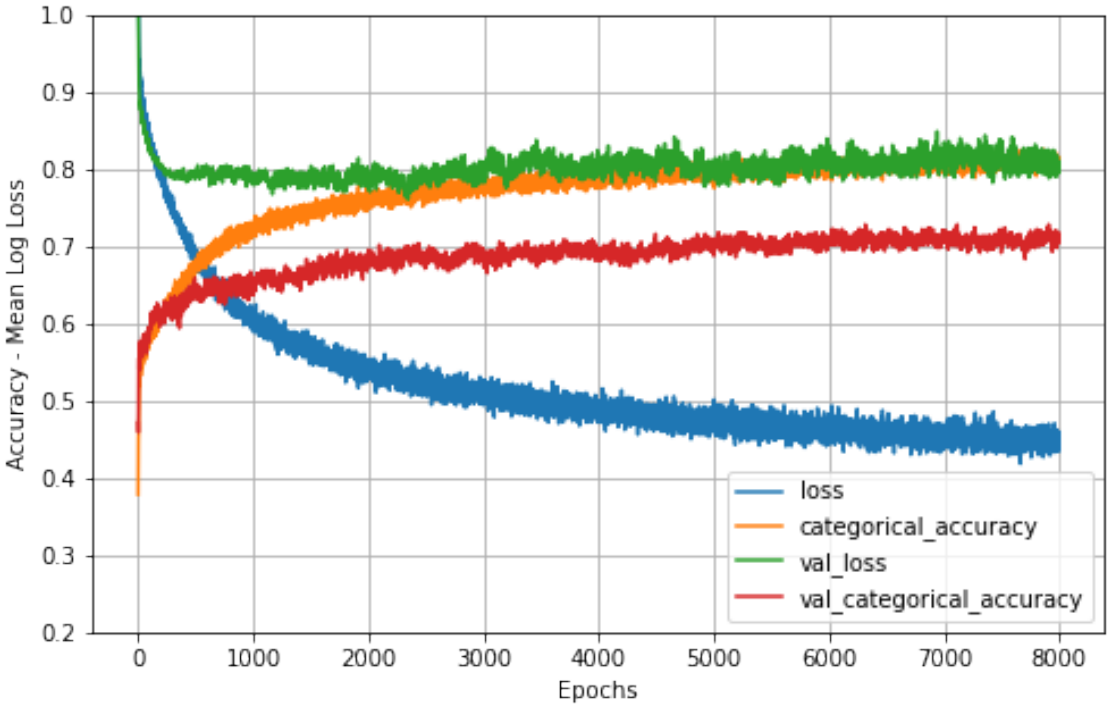

Figura 2.18: Gráfica Resultados - Modelo IV

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
7999	0.45384	0.809086	0.806364	0.706122

Figura 2.19: Modelo IV: accuracies y pérdidas

Los valores de los parámetros e hiperparámetros empleados para conseguir dichos resultados constan en la siguiente tabla

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	8000
m_b (tamaño del batch)	250
función de activación	ReLU
función de inicialización	he_normal
función de salida	softmax
dropout	$[0{,}8, 0{,}4, 70{,}2, 0{,}1]$
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	Sí, dropout
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 2.7: Parámetros - Modelo IV

Observando los resultados, pese a que se ha reducido el bias en comparación al Modelo III, el bias continúa siendo alto. Por tanto, reducir el bias continuará siendo nuestro principal objetivo.

2.5.1. Pruebas intermedias sin mejoras

Para reducir el bias, se han probado distintas arquitecturas neuronales. En primer lugar, como demasiadas pocas neuronas aumentan el bias, se ha considerado ampliar la red neuronal para reducir el bias, introduciendo una capa más justo tras la entrada con 1000 neuronas y un *dropout* de $p = 0{,}1$. Al igual que en el caso anterior, esta configuración y variantes similares (por ejemplo, con un *dropout* de $p = 0{,}9$) han dado lugar a resultados peores en las precisiones que los conseguidos hasta ahora.

Pese a que el objetivo es reducir el bias, se ha probado si era posible reducir la varianza, para ello se ha reducido el número de neuronas por capa, manteniendo el resto de parámetros constantes. Una de las configuraciones probadas ha sido aquella con 4 capas de neuronas y con la configuración [200, 100, 75, 25]. Tanto esta como el resto que se han probado, funcionan igual o peor.

Por todo ello, todas estas configuraciones se han descartado.

Nuestro objetivo continúa siendo reducir el bias ya que sigue siendo alto (0.45284).

Usando el modelo 4, vamos a aplicar una *bacthnormalization* a posteriori de aplicar la función de activación con el objetivo de añadir complejidad al modelo.

Los resultados que se han obtenido han sido catastróficos ya que a aumentado el bias y la varianza e incluso ha disminuido la precisión (*accuracy*) tanto en entrenamiento como en validación, empeorando demasiado el modelo. Por tanto, este cambio se ha deshecho.

2.6. Modelo V

Si hay demasiadas pocas neuronas hay un alto bias. Para intentar bajar el bias, se han cambiado los parámetros de dropout. En los modelos anteriores teníamos un dropout de [0.8, 0.4,

0.2, 0.1] en correspondencia a cada capa oculta. Por tanto, se cambia el orden y se ha instancia un dropout de [0.1 , 0.4, 0.4, 0.4]. Esto hará que se eliminen aleatoriamente menos neuronas ya que se da una probabilidad menor a capas con mayor número de neuronas. Afirmativamente, conseguimos bajar el bias pero la varianza aumenta drásticamente y se crea un overfitting excesivo.

Los resultados asociados a este cambio en los valores del dropout los podemos ver en las figuras 2.21 y 2.20.

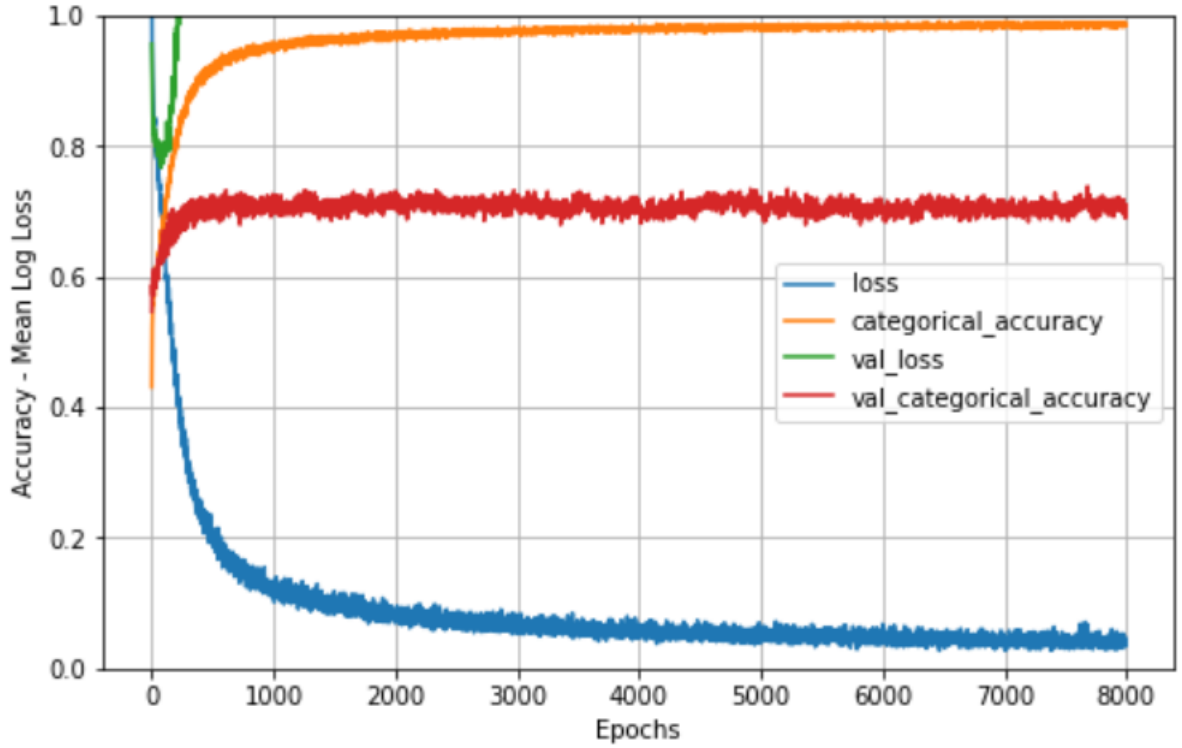

Figura 2.20: Gráfica resultados - Modelo V

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
7999	0.041392	0.984941	3.65761	0.695918

Figura 2.21: Modelo V: accuracies y pérdidas

Si comparamos los resultados de este modelo V con los del modelo IV, observamos que el bias se reduce considerablemente y la precisión aumenta en un $18\,\%$. Aún así, se ha creado un sobreentrenamiento bastante grande en comparación al modelo IV y además la precisión de validación ha disminuido un poquito.

Se intentará reducir el sobreentrenamiento de este modelo V, por si no es posible se dará al modelo IV como final.

Por tanto, seguidamente se ha probado a cambiar la arquitectura aumentando el nº de neuronas para mejorar el bias, teniéndose así 4 capas con [500,250,200,200,100]. El bias a aumentado

0.03 y la precisión se ha reducido a un 79 % en entrenamiento, aunque se ha obtenido una pequeña mejora en el error (3.45324) y precisión de validación (0.697961). Pese a todo ello, la varianza y el sobreentrenamiento continúan siendo excesivamente grandes.

Se ha observado que el error de entrenamiento tiene una tendencia a bajar, así que se ha decidido aumentar el nº de épocas. Pero esto ha ofrecido peores resultados, así que descartamos este cambio.

También se ha probado con otros tipos de funciones como las sigmoide pero no se ha encontrado mejora.

Para reducir la complejidad del modelo y por tanto intentar reducir así la varianza, se ha intentado realizar una combinación del regulador l2 con dropout, ya que se ha investigado que esta combinación suele dar buenos resultados. Sin embargo, en nuestro caso, no ha mejorado el modelo.

Por tanto, este modelo pese a reducir mucho el bias produce una varianza y un sobreentrenamiento excesivos que no hemos sido capaces de reducir. Por ello, consideramos que el mejor modelo que hemos sido capaces de alcanzar es el Modelo IV.

Capítulo 3

Resultados finales

De todas las combinaciones que hemos probado, la configuración del modelo IV es la que arroja resultados más prometedores.

Recordamos que los resultados de precisión (*accuracy*) y pérdida (*loss*) obtenidos para los conjunto de entrenamiento (80 % de los datos del dataset) y desarrollo (10 % de los datos del dataset) son los siguientes que se muestran en las siguientes figuras 3.1 y 3.2:

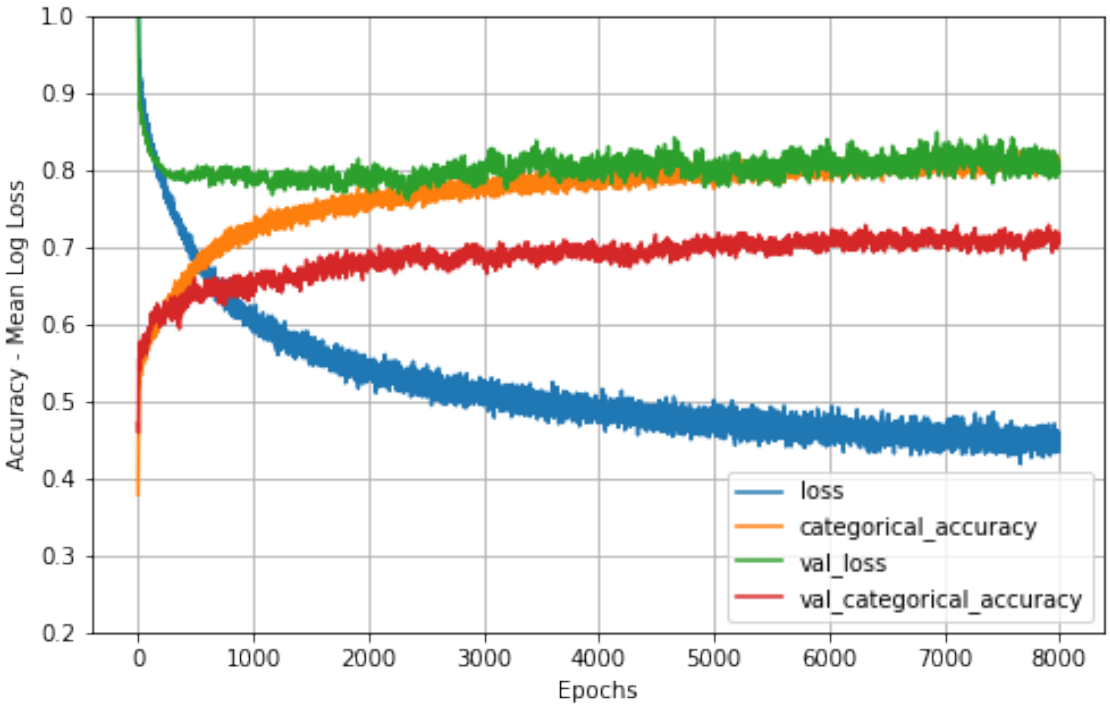

Figura 3.1: Gráfica Resultados - Modelo IV - Modelo Final

	loss	categorical_accuracy	val_loss	val_categorical_accuracy
7999	0.45384	0.809086	0.806364	0.706122

Figura 3.2: Modelo IV: accuracies y pérdidas - Modelo Final

Y que los valores de los parámetros e hiperparámetros empleados para conseguir dichos resultados han sido los de la siguiente tabla 3.1:

(hiper)parámetros	Valor
Neuronas por capa	$[500, 250, 75, 25]$
Tasa de aprendizaje	0.001
Nº de épocas	8000
m_b (tamaño del batch)	250
función de activación	ReLU
función de inicialización	he_normal
función de salida	softmax
dropout	$[0{,}8, 0{,}4, 70{,}2, 0{,}1]$
optimizador	Adam ($lr = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999$)
regularizador	Sí, dropout
normalización	No
función loss	categorical_crossentropy
métrica	categorical_accuracy

Tabla 3.1: Parámetros - Modelo IV - Modelo Final

Por tanto, ha sido el modelo IV el seleccionado para evaluar el modelo con el 10 % de datos que habíamos reservado para realizar el test final.

La evaluación del modelo en el test final ha arrojado los resultados que se pueden observar en la siguiente figura 3.3:

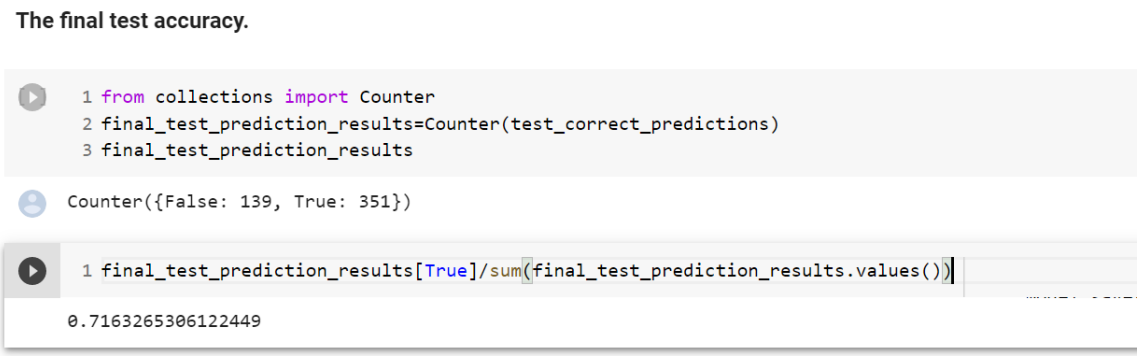

Figura 3.3: Resultados finales - Test final - Modelo IV

Capítulo 4

Conclusiones

Esta claro que una precisión en el test final del 0.72 % no es un resultado muy bueno. Pero que no se hayan obtenido buenos resultados puede deberse a diversos factores. En este caso, achacamos el problema a que el dataset no es el más adecuado. Quizás con una cantidad mayor de datos hubiera arrojado mejores resultados. También, es posible que hubiera sido mejor si hubiera habido más variabilidad de datos, no tantos datos clasificados como *regular* como se da en nuestro dataset.

También destacar que es un dataset complejo en el que ha sido difícil tratar la varianza, ya que se ha mantenido alta constantemente. También achacamos este resultado al dataset.

Por último, hemos obtenido bias muy bajos pero con un sobreentrenamiento (*overfitting*) muy alto y una varianza altísima. Hemos intentado tratar estos sobreentrenamientos y siempre el bias volvía a subir considerablemente, y así en bucle. Por ello, hemos considerado que el Modelo IV, pese a tener un bias y una varianza alta, es el mejor modelo que podemos alcanzar.

Por otro lado, nos gustaría mostrar una tabla (4.1) comparativa de los tiempos empleados en cada modelo final.

Modelo	**Tiempo en entrenamiento (s)**
Modelo I	18.204424000000003
Modelo II	110.70132
Modelo III	119.546223
Modelo IV	490.11348100000004
Modelo V	469.394318

Tabla 4.1: Modelos/Tiempo

Se observa como el tiempo en el modelo IV es el mayor en comparación a los demás, esto se debe al paso de 2000 a 8000 épocas. Aún así, 8 minutos (490.11348100000004s) es un tiempo más que aceptable para una red de neuronas profunda. Y, como el modelo IV además nos proporciona los mejores resultados, podemos concluir que es el mejor clasificador que hemos encontrado para el dataset *Wine Quality*.