Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Redes neuronales artificiales

Clasificador basado en redes neuronales

Autores: Carlos Golvano, Carlos Yanguas, Jorge Casajus.

Emails: carlos.golvano@alumnos.upm.es, carlos.yanguas@alumnos.upm.es, jorge.casajus@alumnos.upm.es.

Fecha: 17/12/2021

ELCOSTE

Contenidos

Lista de Figuras

Capítulo 1

Introducción

1.1. Organización

A continuación se exponen los diferentes capítulos en los que esta dividido el trabajo, procurando aportar en cada una de ellas la explicación que cubre una parte fundamental del trabajo realizado.

- Capítulo 2: Conjunto de datos. En este capítulo se describe el conjunto de datos utilizado para el entrenamiento y prueba del clasificador, así como las operaciones de preparación y limpieza aplicadas sobre el mismo previo uso de éste.
- Capítulo 3: Diseño y creación de Modelos. En este capítulo se desarrollan los enfoques y arquitecturas utilizadas para construir y entrenar los clasificadores basados en redes neuronales que se emplearán en el siguiente capítulo.
- Capítulo 4: Evaluación de los modelos. En este capítulo se ofrece una descripción las métricas bajo las que se ha estimado la eficacia de los clasificadores implementados, así como una comparación cuantitativa entre los diferentes modelos.
- Capítulo 5: Proceso de búsqueda de los mejores hiperparámetros. En este capítulo se expone el procedimiento de optimización de los hiperparámetros de las redes.
- Capítulo 5: Conclusiones. En este capítulo se sumarizan las conclusiones más relevantes extraídas de los resultados obtenidos en el apartado anterior.

1.2. Contexto

Inspiradas por la biología neurosináptica del cerebro, las **redes neuronales** toman la estructura de un grafo ponderado y orientado cuyos nodos (neuronas artificiales) están conectados unos con otros a través de conexiones que emulan las uniones axión-sinápsis-dendrita de las neuronas, de forma que cada enlace entre éstas tiene un peso asociado que determina la influencia de cada una de ellas en la activación lógica de la siguiente.

Los pesos y la arquitectura de las conexiones neuronales son los elementos en que se codifica la función que aproxima la red neuronal, por lo que se requiere de un proceso de diseño y entrenamiento del modelo (aprendizaje de los pesos y desplazamientos) para que cumpla satisfactoriamente una tarea. Los valores aprendidos de los pesos de las conexiones y desplazamientos (*bias*) de las neuronas constituyen el conjunto de **parámetros** de la red.

Se dice que una red de neuronas artificiales (RNA) es **alimentada hacia delante** (*feedforward*) si sus neuronas se estratifican en capas en una estructura tal que las neuronas de una capa sólo pueden tener una conexión dirigida hacia las de la capa siguiente. A las capas intermedias,

aquellas que se encuentran entre las que recogen a los nodos de entrada y los de salida se les denomina **capas ocultas**.

Una red neuronal con una única capa oculta es un **aproximador universal**, pues es capaz de aproximar cualquier función con un error arbitrariamente bajo. Por esta característica, las **redes de neuronas artificiales** resultan adecuadas para la resolución de infinidad de problemas, incluyendo la clasificación de instancias [1].

1.3. Redes de neuronas profundas

Una red de neuronas se considera profunda cuando tiene más de una capa oculta, lo que le permite codificar en sus parámetros funciones de gran complejidad.

Hasta recientemente no se había trabajado con redes neuronales con numerosas capas ocultas, pues no la capacidad de cómputo de los ordenadores no permitía entrenarlas. No obstante, cuando se pudo comenzar a experimentar con ellas, los científicos se dieron cuenta de que existían nuevos problemas con el entrenamiento de esta clase de redes: los ***Deep Learning problems***. Por ello, al dejar de valer las técnicas clásicas de entrenamiento de redes, como la retropropagación del gradiente, se empezaron a usar *tweaks* en los algoritmos de aprendizaje, que son precisamente lo que se conforma el conjunto de las técnicas de Deep Learning.

1.4. Clasificación con redes neuronales

El problema de clasificación consiste en, a partir de un conjunto de datos correspondientes a una serie de instanciaciones de un conjunto de variables predictoras, elaborar un modelo que permita asignar un valor a una variable de clase asociada a una nueva instancia de datos comprensivos del valor de variables predictoras.

A grandes rasgos, los métodos de aprendizaje de redes neuronales dentro del paradigma de clasificación se dividen en tres ramas [2]:

- **Aprendizaje supervisado**: A la red neuronal se le proporcionan datos de entrenamiento ya clasificados que se utilizan para inferir reglas generales que se pueden aplicar sobre nuevos casos sin clasificar y que quedan codificadas en los pesos de las conexiones entre las neuronas.
- **Aprendizaje sin supervisar**: Los datos sobre los que aprende la red no están etiquetados, por lo que infiere las reglas reconociendo patrones en el conjunto de datos de entrenamiento. Este método de aprendizaje limita el tipo de predicciones que se puede hacer sobre nuevas instancias de datos al *clustering*.
- **Aprendizaje por refuerzo**: Inspirado en la psicología conductista, la idea de este tipo de aprendizaje consiste en reforzar de forma positiva o negativa, por medio de premios y castigos (respectivamente) las decisiones del modelo, a través de lo que se denomina proceso decisor de Markov [3]

El trabajo que se expone y analiza en esta memoria consiste en la aplicación de un modelo de clasificación supervisada sobre un conjunto de datos que recogen los atributos de una serie de futbolistas del videojuego FIFA19 para etiquetarlos según su puntuación en pobre, intermedio, bueno y excelente.

Con este objetivo se han diseñado y entrenado múltiples modelos de red neuronales, probando diferentes combinaciones de hiper-parámetros, aquellos parámetros que describen el funcionamiento de la red y que no se obtienen del entrenamiento, sino que deben ser ajustados de forma manual.

Capítulo 2

Conjunto de datos

El conjunto de datos a partir del cual se van a entrenar y probar los clasificadores, proviene de la edición del 2021 del videojuego FIFA, en el cual se describe el perfil de cada jugador por medio de 89 atributos. Originalmente, el conjunto de datos contiene 18207 instancias de estos atributos.

Para lograr la mayor precisión posible en los modelos se necesita de una serie de transformaciones aplicadas sobre el conjunto de datos en bruto, al que, en primer lugar, se le han eliminado las instancias correspondientes a porteros, pues el blanco a clasificar son los jugadores de campo.

En primer lugar es necesario determinar qué subconjunto de atributos de entre los originales es el óptimo para predecir la calificación general de un jugador (y a partir de ésta, clasificarlo en una de las cuatro categorías establecidas). Por ello se sigue un proceso de **selección de características** (*feature selection*) para extraer los atributos que más información aporten de cara a la predicción de la calificación de un jugador, consiguiendo que las estimaciones sean más precisas y eficientes.

Los atributos irrelevantes (tales como el nombre o el ID del jugador) han sido descartados directamente. La importancia del resto de características es cuantificable por múltiples métodos, y para este trabajo se ha escogido la **correlación lineal**, una medida estadística que expresa en qué grado dos variables están relacionadas linealmente (esto es, cambian conjuntamente a una tasa constante). La correlación lineal es una herramienta común para describir relaciones simples sin hacer afirmaciones sobre causa y efecto. De entre las variantes de la fórmula para el calculo de la correlación que existen se ha optado por utilizar el **coeficiente de correlación de Pearson** de una muestra [4], que formalmente se escribe como:

$$r = \frac{\sum(x_i - \bar{a})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{a})^2(y_i - \bar{y})^2}} \in [-1, 1] \tag{2.1}$$

- Si $r = 1$ se dice que hay **correlación positiva** perfecta entre dos variables: cuando una variable aumenta, la otra también lo hace.
- Si $r = -1$ se dice que hay **correlación negativa** perfecta entre dos variables: cuando una de las dos aumenta, la otra disminuye.
- Si el valor es 0, no hay ninguna correlación entre las variables. Se dice que no hay relación entre su variación dado que al compararlas, cambian de forma aparentemente aleatoria entre sí.

Cualquier valor en el intervalo $(-1, 1)$ diferente a 0 representa una correlación parcial (positiva o negativa) entre dos variables, lo que implica que el conocimiento de una no aporta información completa sobre la otra.

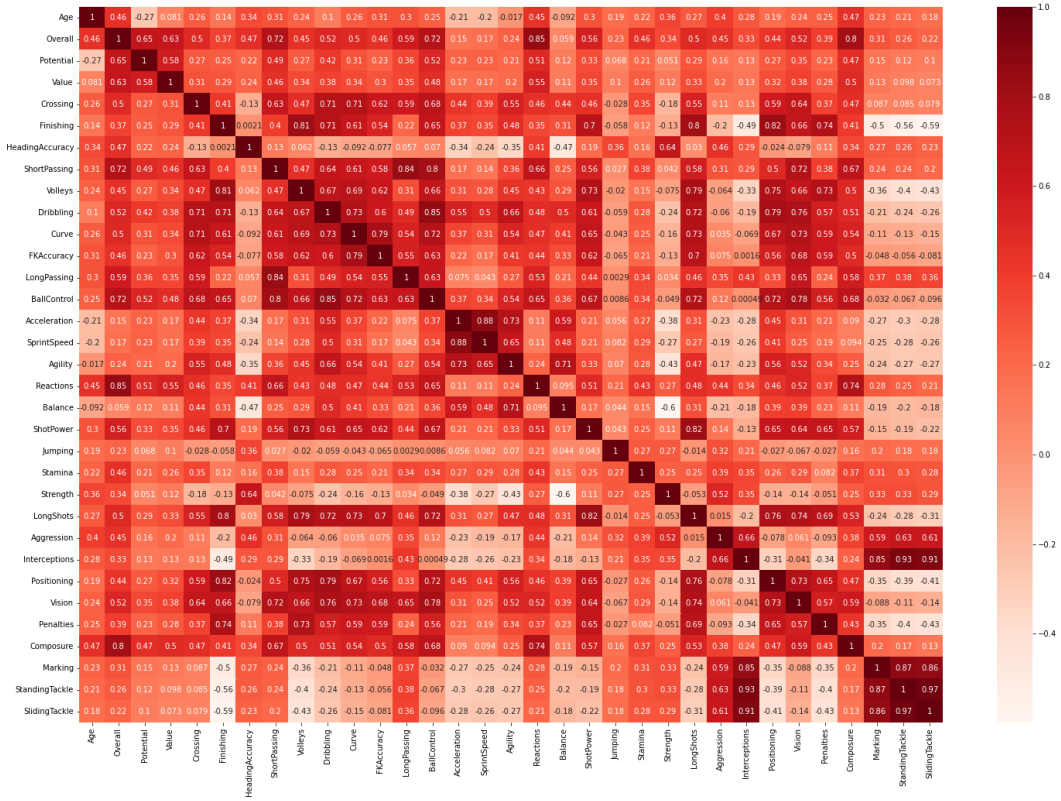


Figura 2.1: Matriz de correlación de los atributos que describen a un jugador presentes en el set de datos original.

En la figura 2.1 aparece representada en escala cromática la matriz de correlación del set de datos original. Atendiendo a los valores de la correlación entre las variables predictoras y de éstas con la variable de clase, se han creado varios conjuntos de atributos a considerar para la labor de clasificación:

- **Conjunto de datos por defecto:** Es el *dataset* proporcionado originalmente. El umbral en la correlación mínima con la calificación general (*Overall*) para la consideración de un atributo es del 40 %, descartando los atributos *age*, *potential* y *value*. En la figura 2.2 vienen recogidos los atributos seleccionados y la correlación con la puntuación *Overall* calculada a través del coeficiente de Pearson.

- **Conjuntos de datos 2, 3 y 4:** Tras hacer un análisis visual del conjunto de atributos y sus correlaciones, se llegó a la conlusión de que *age*, *potential* y *value*, podrían estar efectivamente correlacionados con la calificación general. La correlación de estos atributos con la varible *Overall* resultó ser de 0.455, 0.650 y 0.634 respectivamente. Se optó por incluir estos atrubutos y establecer tres nuevos conjuntos de datos en los que el umbral de inclusión de un atributo se situara en 0.50, 0.60 y 0.70 de correlación con la calificación global o superior. Esto dió como resultado los conjuntos de atributos mostrados en la figura 2.3 junto con su respectiva correlación.

- **Selección de variables mediante XGBoost:** Finalmente se ejecutó un proceso de selección de variables usando el algoritmo XGBoost [5], que tras una investigación en la materia determinamos como uno de los métodos de selección de características más populares y recomendados actualmente.

  Tal y como se puede ver en el código recogido en el fichero *Preprocesado_ XGBoost.ipynb*, localizado dentro de la carpeta *Preprocesado*, primero se estableció establecimos un umbral de correlación de un 40 % para realizar un filtro preliminar de características y sobre éstas se implementó un clasificador basado en el algoritmo XGBoost, cuya salida, dado que devuelve el número de variables utilizadas y la precisión obtenida por el mismo, permitió realizar un análisis del número de variables óptimo a utilizar en los modelos de red neuronal. Sin embargo, como se puede apreciar en la figura 2.4 los resultados no son congruentes, ya que el nivel de acierto del clasificador aumenta y disminuye de forma aparentemente arbitraria respecto a dicho número de variables a partir de la selección de tres de ellas.

Una vez seleccionados los atributos a utilizar para predecir la variable objetivo, *Overall*, a todos los conjuntos de datos se han aplicado las mismas operaciones:

1. Se han dividido las puntuaciones de *Overall* en cuatro clases dadas por los rangos, [46-62] pobre, [63-66] intermedio, [67-71] bueno y [72-94] excelente, intentando que todas las clases tengan aproximadamente el mismo número de jugadores

2. Las clases se han codificado siguiendo el método *one-hot encoding* [6], que permite pasar de etiquetas categóricas a datos númericos en forma de vector binario

3. Se ha aplicado una normalización *minmax* sobre las características a utilizar, que matemáticamente equivale a calcular para cada variable:

$$X_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.2}$$

Finalmente se vuelcan los atributos normalizados y las etiquetas en formato *one-hot encoded* por separado en dos archivos *.csv*, que se utilizarán a modo de entrada de los diferente modelos diseñados.

De los datos recopilados, un 80 % se destinarán al entrenamiento, un 10 % al desarrollo de los modelos y un 10 % a probar los clasificadores.

Atributos	Correlación con Overall
Reactions	0,85
Composure	0,80
ShortPassing	0,72
BallControl	0,72
LongPassing	0,59
ShotPower	0,56
Vision	0,52
Dribbling	0,52
Curve	0,50
LongShots	0,50
Crossing	0,50
HeadingAccuracy	0,47
Stamina	0,46
FKAccuracy	0,46
Aggression	0,45
Volleys	0,45
Positioning	0,44

Figura 2.2: Conjuntos de atributos inicial.

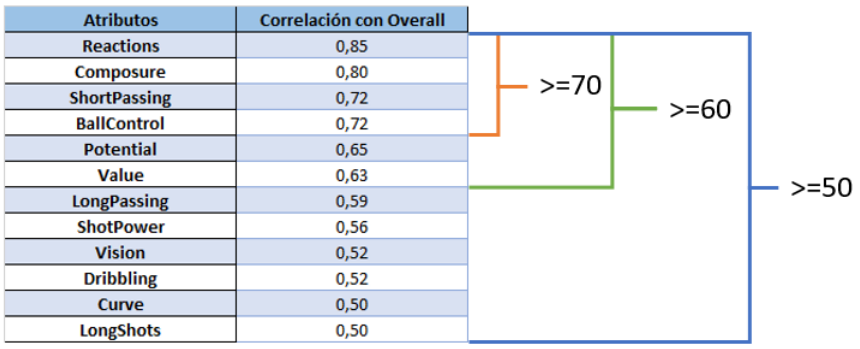


Figura 2.3: Conjuntos de atributos con umbral de correlación $\geq$ 0.5 (azul), 0.6 (verde), 0.7(naranja).

Nº Variables	Precisión del clasificador XGBoost
1	76.93
2	87.26
3	96.37
4	95.88
5	96.09
6	96.34
7	96.53
8	96.47
9	96.65
10	96.59
11	96.62
12	96.65
13	96.62
14	96.56
15	96.59
16	96.65
17	96.62
18	96.65
19	96.47
20	96.59

Figura 2.4: Precisión del clasificador XGBoost según número de variables.

Capítulo 3

Diseño y creación de modelos

A lo largo de este capítulo se desarrolla el proceso de construcción de modelos. Para poder trabajar en paralelo, la carpeta *Modelos* contiene otras tres carpetas correspondientes a cada uno de los miembros del grupo. Además, con el objetivo de optimizar el tiempo disponible para realizar el máximo número posible de pruebas de diferentes configuraciones, se ha implementado una automatización del proceso de toma de resultados, de modo que cada vez que se crea un modelo, se guardan de forma automática:

- La **configuración del modelo** en una hoja de cálculo (*model_ conf.xlsx*) que contiene el conjunto de hiperparámetros utilizados para el modelo en cuestión, a saber:
  - **Número de épocas** (veces que se le muestra el set de datos de entrenamiento a la red) máximas con el que se pretende entrenar el modelo, sujeto al *Early Stopping* [7, 8] proporcionado por Keras. Este recurso permite establecer un número de épocas máximo, dado por la paciencia o *patience* del modelo, en el que si una métrica (en nuestro caso *val_ accuracy*) no mejora, se detiene el entrenamiento.
  - **Tasa de aprendizaje** a la cual aprende el modelo. El valor de este hiperparámetro determina cuánto cambian los parámetros del modelo en respuesta al error de la vez cada vez que son actualizados. Su elección es determinante para el buen funcionamiento del modelo, pues un valor demasiado pequeño hará que el proceso de aprendizaje sea muy lento y propenso al estancamiento, mientras que uno demasiado grande puede hacer que el proceso de aprendizaje sea inestable [9].
  - **Tamaño de *batch***, pues los parámetros de la red neuronal no se actualizan a cada vez que se le presenta una instancia de datos, sino que se acumula la actualización por un subconjunto (*batch*) de instancias de entrenamiento de un determinado tamaño. El uso de grandes *batches* de datos para el entrenamiento mejora la eficiencia del algoritmo por paralelización de tareas, pero al coste de perder en calidad de generalización del modelo [10]
  - **Número de capas y de neuronas** por cada una, almacenadas en forma de un vector $n_{\text{capas}}-$dimensional, siendo n_{capas} el número de capas de la forma: $[a_1, ..., a_{n_{\text{capas}}}]$ donde a_i representa el número de neuronas en la capa $i-$ésima.
  - **Funciones de activación** de las capas ocultas y de la capa de salida: Las funciones de activación son el constructo matemático que permite calcular el valor de salida (*output*) de un nodo a partir de una serie de entradas (*inputs*). En el caso de las redes neuronales, modelizan el comportamiento y transmisión de una señal por una neurona a través su axón (su salida hacia la siguiente capa) en función de los estímulos que llegan a sus dendritas (entradas de la capa anterior). Las funciones lineales no permiten la codificación de funciones arbitrareamente complejas, por lo que se utilzan funciones con no-linearidades, como la Sigmoide, la tangente hiperbólica o las más

modernas ELU, ReLU y *Softmax* [11]. Las funciones de activación de las capas ocultas vienen almacenadas en un vector de $n_{\text{capas}} - 1$ dimensiones, en el que el $i-$ésimo elemento determina qué función de activación emplea la $i-$ésima capa oculta. En cuanto la función de la capa de salida, esta se guarda en una variable separada. Precisa aclaración que en algún caso y debido al gran número de pruebas que se han llevado a cabo, en algunos casos se ha mantenido un vector de funciones de activación para las capas ocultas de más dimensiones que capas ocultas, que aunque es un error, no lanza una extensión, pues sólo se emplean los $n_{\text{capas}} - 1$ primeros términos de éste.

- ***Dropout***: Es un tipo de regularización (penalización en el aprendizaje de un modelo demasiado complejo para evitar el *overfitting* de los datos) consistente en ignorar ciertas unidades neuronales, dando lugar a una red reducida y evitando la generación de co-dependencias entre las neuronas [12]. El valor de *Dropout* representa la fracción de neuronas que serán aleatoriamente ignoradas durante cada etapa de entrenamiento.
- **Normalización del *batch***, codificada en la variable booleana *batch_ normalization*, que toma un valor de 1 si se ha efectuado (se aplica a todas las capas intermedias) y de 0 en caso contrario. La normalización de batch es una forma de reparametrizar la red para coordinar mejor la actualización de pesos de las capas de neuronas, reduciendo el desplazamiento covariante interno, es decir, el cambio en la distribución de las activaciones neuronales al actualizar los parámetros durante el entrenamiento [13]. Además, también tiene un efecto de regularización sobre el modelo, haciendo que en ocasiones no sea necesario incluir *Dropout*.
- **Regularización L2**, el método de regularización más común en *Deep Learning*, que introduce una penalización proporcional al cuadrado de la magnitud de los pesos. En caso de utilizarse, el valor de la variable *L2* será el que adopte la constante de proporcionalidad de la regularización para cada una de las capas intermedias [14].

- Los **resultados del entrenamiento**, recopilados en una hoja de cálculo (*model_ res.xlsx*). Las métricas extraídas en la fase de entrenamiento son:
  - ***train_ accuracy***: Precisión lograda por el modelo para el conjunto de entrenamiento
  - ***train_ dev***: Precisión obtenida por modelo sobre el conjunto de desarrollo
  - ***train_ err***: Pérdida del modelo para el conjunto de entrenamiento
  - ***train_ dev***: Pérdida del modelo para el conjunto de desarrollo
  - ***human_ err***: Error humano a la hora de hacer la misma tarea que el modelo (clasificar los jugadores en las 4 clases)
  - ***Bias***: Diferencia entre la pérdida de entrenamiento y el error humano
  - ***Variance***: Diferencia entre la pérdida de desarrollo y la pérdida de entrenamiento
  - ***Time***: Tiempo en segundos que ha implicado el entrenamiento del modelo
- **Informe sobre el conjunto de test**: Este informe, guardado de nuevo como una hoja de cálculo (*informe.xlsx*) ofrece para cada una de las clases las métricas *precision*, *recall*, F1-*score* y *support*, descritas en el siguiente capítulo, así como la métrica media de cada una de ellas calculada de manera uniforme, y de manera ponderada.
- **Gráfico de entrenamiento** (*error_ plot.png*): Representación gráfica de la evolución de las métricas pérdida de entrenamiento, pérdida de desarrollo, precisión de entrenamiento y precisión de desarrollo a lo largo de las épocas de entrenamiento.
- **Matriz de confusión** sobre el conjunto de test: Ilustra el número de ejemplos bien y mal clasificados para cada una de las clases, y se almacena como un gráfico cromático (*cm.png*).

- **Estructura del modelo**, guardada como un archivo *nombre_modelo.h5* que contiene la estructura de la red y el valor de cada uno de los parámetros que lo conforma.
- **Histórico** (*nombre_modelohistory*), un diccionario en el cual se guarda el histórico de entrenamiento.

Teniendo todos estos resultados, es más facil desentrañar el proceso de intuición que lleva a ejecutar los cambios adecuados para tratar de mejorar el modelo.

En líneas generales, la mecánica de construcción de modelos de redes neuronales profundas parte de la búsqueda de un diseño del modelo que ofrezca un gran sobreentrenamiento (*overgitting*) para después regularizarlo mediante recursos como *dropout*, normalización de *batch*, regularización L2 o reduciendo las capas y/o las neuronas por capa.

También se ha utilizado un inicializador de pesos apropiado para la función de activación utilizada con el objetivo de evitar el problema del *vanishing/exploding gradient*. Esto es, para ReLU o ELU se ha utilizado la inicialización de kernel He uniforme, mientras que para la función de activación *Softmax*3 se ha utilizado GlorotNormal.

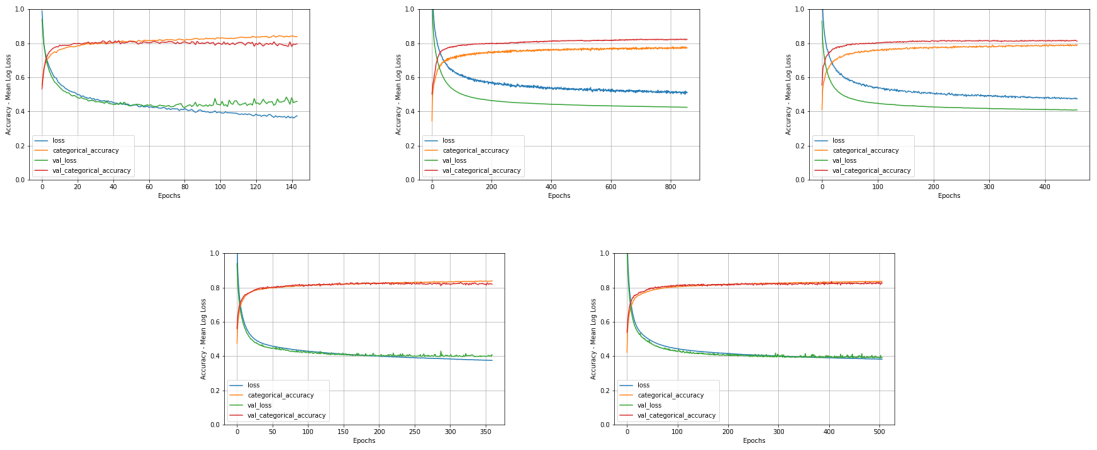


Figura 3.1: Evolución de las pruebas realizadas, desde *prueba_elu_9* (arriba a la izquierda) hasta *prueba_elu_13* (abajo a la derecha).

	Epocas	LR	Batch	Neuronas_por_capa	act_int	act_out	Dropout	BatchNormalization	L2
prueba_elu_9	1000	0,01	256	[256, 200, 160, 80, 20]	[elu, elu, elu, elu, elu, elu]	softmax	0	1	
prueba_elu_10	3000	0,005	256	[256, 200, 160, 80, 20]	[elu, elu, elu, elu, elu, elu]	softmax	0.1	1	
prueba_elu_11	1000	0,005	256	[256, 200, 160, 80, 20]	[elu, elu, elu, elu, elu]	softmax	0.1	0	
prueba_elu_12	1000	0,005	256	[256, 200, 160, 80, 20]	[elu, elu, elu, elu, elu]	softmax	0	0	
prueba_elu_13	1000	0,005	256	[200, 100, 80, 40, 10]	[elu, elu, elu, elu, elu]	softmax	0	0	

Figura 3.2: Evolución del cambio de hiperparámetros en las pruebas realizadas.

En las figuras 3.1 y 3.2 se pueden observar las gráficas de entrenamiento y los hiperparámetros obtenidos en la configuración de cinco modelo que son ilustrativos de varios de los conceptos mencionados previamente en este capítulo:

1. En la *prueba_elu_9* se puede apreciar claramente que el modelo esta sobreentrenado, ya que si nos fijamos por ejemplo en categorical_accuracy (nivel de acierto en el set de entrenamiento, curva naranja) vemos que supera claramente a val_categorical_accuracy (nivel de acierto en el set de prueba, curva roja), lo que parece lógico, pues que la configuración del modelo entrenado para este ejemplo carece de ningún tipo de regularización.
2. *prueba_elu_10*: En esta prueba se introduce *dropout* al 0.1 y normalización de *batch*, reduciendo además el learning rate de 0.01 a 0.005, ya que los dientes de sierra de la gráfica de entrenamiento anterior denotan inestabilidad en el aprendizaje del modelo y

son ocasionados por un *learning rate* alto. El resultado en este caso ha sido la sobre-regularización del modelo, pues ahora existe una gran diferencia entre las curvas roja y naranja, estando por encima la primera y, por tanto, implicando una mala generalización del modelo.

3. *prueba_elu_11*: En esta configuración sólo cambia respecto a la anterior que no se introduce normalización de los *batches*. Ahora son muy semejantes las curvas roja y naranja, si bien aún aparece ligeramente por encima la roja, lo que significa que, a pesar de haberse reducido, aún hay más regularización de la cuenta.

4. prueba_elu_12: En el entrenamiento de este modelo se ha elimina el *dropout*, estableciendo su valor a 0, por lo que se diferencia de la aproximación prueba_elu_9 únicamente en que su tasa de aprendizaje se ha reducido a la mitad, lo que ocasiona que haya un ligero grado de sobreentreno.

5. prueba_elu_13: Para esta última prueba se reduce el número de neuronas y se prescinde de todo tipo de regularización. Como se puede apreciar en el gráfico correspondiente, las curvas roja y naranja convergen de forma prácticamente idéntica, por lo que se puede decir que se ha logrado un buen equilibrio para esta solución.

Al igual que se ha automatizado el proceso de prueba y extracción de resultados, como se disponía de un enorme número de modelos a cotejar, también se ha desarrollado un código para efectuar una comparación inmediata sobre todas las redes entrenadas. Tal programa viene implementado en el cuaderno *Comparativa-resultados.ipynb*, y permite ordenar los modelos construídos en función de la precisión que hayan obtenido para el conjunto test. La configuración de los 20 mejores modelos de este ranking se puede ver en la figura 3.3.

Epocas	LR	Batch	Neuronas_por_capa	act_int	act_out	Dropout	BatchNormalization	L2	precision_test
1000	0,1	512	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001	0,8283
10000	0,075	256	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001	0,8221
1000	0,005	256	[200, 100, 80, 40, 10]	[elu, elu, elu, elu, elu]	softmax	0	0		0,8208
1000	0,01	128	[512, 256, 128, 64, 32]	[elu, elu, elu, elu, elu]	softmax	0	0		0,8202
1000	0,1	512	[30, 20, 10, 5]	[elu, elu, elu, elu]	softmax	0,01	0	0,001	0,8184
1000	0,01	128	[512, 256, 128, 64, 32]	[elu, elu, elu, elu, elu]	softmax	0,05	0		0,8184
10000	0,01	512	[500, 250, 75, 25]	[relu, relu, relu, relu]	softmax	0,05	0	0	0,8177
10000	0,1	512	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001	0,8177
1000	0,1	512	[256, 200, 160, 120, 80]	[relu, relu, relu, relu, relu]	softmax	0,1	0	0	0,8171
1000	0,01	512	[256, 200, 160, 120, 80, 50, 20]	[elu, elu, elu, elu, elu, elu, elu]	softmax	0,1	0		0,8165
1000	0,1	512	[500, 250, 100, 75]	[elu, elu, elu, elu]	softmax	0,2	0		0,8165
1000	0,01	256	[512, 256, 128, 64, 32]	[elu, elu, elu, elu, elu]	softmax	0	0		0,8165
1000	0,1	256	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001	0,8165
1000	0,01	512	[256, 200, 160, 120, 80, 50, 20]	[elu, elu, elu, elu, elu, elu, elu]	softmax	0	0		0,8159
1000	0,005	256	[400, 200, 160, 80, 20]	[elu, elu, elu, elu, elu]	softmax	0	0		0,8159
1000	0,001	512	[256, 200, 160, 120, 80]	[relu, relu, relu, relu, relu]	softmax	0	0	0	0,8153
1000	0,1	128	[512, 256, 128, 64, 32]	[elu, elu, elu, elu, elu]	softmax	0,15	0		0,8153
1000	0,1	512	[1000, 500, 250, 75]	[elu, elu, elu, elu]	softmax	0,5	1	0,001	0,8140
1000	0,1	2048	[256, 200, 160, 120, 80]	[relu, relu, relu, relu, relu]	softmax	0,1	0	0	0,8140
1000	0,01	128	[512, 256, 128, 64, 32]	[elu, elu, elu, elu, elu]	softmax	0,2	0		0,8140

Figura 3.3: 20 mejores configuraciones de modelos según precisión sobre el preprocesado original de datos.

Capítulo 4

Evaluación de los modelos

Las redes de neuronas artificiales emplean una **función de pérdida** para determinar cuánto se alejan las predicciones efectuadas por el modelo de los valores reales, de modo que funciona como una medida de cuánto de bien se están modelizando los datos [15]. La optimización de esta función de pérdida es el mecanismo que gobierna el entrenamiento de la red, por lo que la elección de una función de pérdida adecuada para el problema que intenta resolver la red es uno de los factores más determinantes para el desempeño de un modelo.

En el caso en cuestión, el problema es de **clasificación multi-clase**: se debe predecir la clase de una instanciación de variables predictoras en un escenario en el que cada ejemplo puede pertenecer a una de varias categorías. Este ejercicio es equivalente a, dada una instancia, asignar una clase de un conjunto como positiva y el resto como negativas.

El estándar *de-facto* para la función de pérdida para aplicaciones de clasificación multi-etiqueta es la función de **entropía cruzada categórica** (*categorical cross-entropy loss*) [16]. Dicha función permite calcular el **error local** cometido en un ejemplo de entrenamiento, y se formaliza matemáticamente como:

$$L^{(p)}(y,t) = -\sum_{i=1} n_y t_i \log(y_i) \tag{4.1}$$

Donde:

- p: El número de ejemplo de entrenamiento para el que se calcula
- y: Salida binarizada (clasificación) producida por el modelo
- t: Salida real (probabilidad de cada clase) producida por el modelo
- n_y: Número de clases
- $t_i \in [0,1]$: Valor del vector t para la clase i
- y_i: Valor del vector y para la clase i (0 ó 1)

Intuitivamente, la entropía categórica cruzada es una métrica de cómo de lejos están las etiquetas asociadas a las instancias de datos de las clases en que se deberían haber clasificado. A más diferencia, mayor será pérdida y peor la precisión del clasificador.

La pérdida asociada a todo el conjunto de datos se puede calcular como la media aritmética del error sobre cada ejemplo de entrenamiento, y será la función final a minimizar, pues es la que evalúa el error global del clasificador:

$$J(W) = -\frac{1}{m}\sum_{p=1} m \cdot L^{(p)} \tag{4.2}$$

Tras diseñar y entrenar cuantos diferentes clasificadores ha sido posible, es el momento de hacer una evaluación de los mismos, es decir, cuantificar su bondad en la tarea que los atañe. Para ello, tal y como se introdujo en el capítulo anterior, se atiende a varios aspectos.

El primero de estos aspectos a tener en cuenta para la evaluación del modelo viene dado por las **métricas extraídas durante la fase de entrenamiento**, que derivan de los costes del conjunto de entrenamiento y desarrollo y fueron ya presentadas en el Capítulo 3. El conjunto de entrenamiento está formado por los datos empleados para ajustar los parámetros (los pesos y desplazamientos), mientras que el conjunto de desarrollo se emplea para medir lo bien que el modelo clasifica los datos no vistos y ajustar los hiperparámetros (como la tasa de aprendizaje) en consecuencia. No obstante, hay que tener en cuenta que como el modelo se ajusta en función de los resultados sobre el conjunto de datos de desarrollo, el clasificador tiene un rendimiento aparentemente superior sobre este conjunto, pero hay que tener en cuenta que este resultado es, en realidad, un espejismo, pues las métricas extraídas del proceso de desarrollo están altamente sesgadas. Del entrenamiento se obtiene el *bias* y la varianza combinando las métricas de estos dos conjuntos de datos.

Por otro lado, para evaluar cómo se ajusta al problema el modelo creado en un contexto distinto al de entrenamiento se hace uso del conjunto de *test*, que no ha sido visto por el modelo antes de la aplicación. Para evaluar el **rendimiento del modelo sobre el conjunto de pruebas** existen diferentes métricas, de entre las que se han utilizado aquellas que proporciona la función *classification_report* de la librería *SciKit Learn* [17, 18], dado que son las más ampliamente extendidas y permiten evaluar de manera clara el comportamiento de los modelos:

1. **Precisión**: La precisión (*precision* en inglés) se puede describir intuitivamente como la capacidad del clasificador de no etiquetar como positiva una muestra que es negativa. Su fórmula es:
$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}} \in [0, 1] \tag{4.3}$$
Donde TP es el número de verdaderos positivos (*True Positives*) y FP el de falsos positivos (*False Positives*)

2. **Sensibilidad**: También conocido como *recall* en ingés, esta métrica se define de forma sencilla como la capacidad del clasificador para encontrar todas las muestras positivas. Su calcula como:
$$\text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}} \in [0, 1] \tag{4.4}$$
Donde FN es el número de falsos negativos (*False Negatives*).

3. **Puntuación F1**: La F1-*score* se calcula como la media armónica de la precisión y la sensibilidad obtenidas. La contribución relativa de la precisión y la sensibilidad a la puntuación F1 son iguales, y su fórmula se escribe:
$$F1 = 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}} \in [0, 1] \tag{4.5}$$

4. **Support**: Número de ocurrencias de cada clase en el conjunto de datos a predecir. Permite comprobar que la muestra es justa.

5. **Matriz de confusión**: Por definición, la matriz de confusión C es aquella para la que los coeficientes $C_{i,j}$ equivalen al número de observaciones cuya clasificación verdadera es i y que el modelo clasificador predice como j. Es una matriz que permite expresar el número de verdaderos positivos de cada clase frente a la predicción que se ha hecho sobre esa determinada instancia (la clase asociada por el modelo a un ejemplo del *dataset*).

   Las métricas de precisión, sensibilidad y puntuación F1 son mejores cuanto más se acercan a la unidad.

Capítulo 5

Proceso de búsqueda de los mejores hiperparámetros

El procedimiento general seguido ha consistido en la realización de numerosas pruebas variando de forma aislada determinados hiperparámetros y evaluando cómo afecta tal cambio al rendimiento del clasificador. De esta manera se puede establecer el rango de valores que proporciona el mejor funcionamiento de cada hiperparámetro por separado, para después afinar la configuración del modelo estudiando cómo interactúan entre sí estas variaciones, ya que la asunción de independencia del efecto de los parámetros sobre el rendimiento del clasificador es realista, aunque se haya usado como guía general.

El otro aspecto determinante para la estimación de los hiperparámetros óptimos ha sido la búsqueda de un equilibrio entre sobreentrenamiento (*overfitting*) y subajuste (*underfitting*), ya que a grandes rasgos, la **variación de cualquier hiperparámetro romperá el equilibrio** entre ambos aspectos en el aprendizaje, proporcionando un modelo que se ajuste mejor a los datos de entrenamiento o uno que generalice mejor, siempre teniendo en cuenta que la red se tenía que poder entrenar con los recursos (tanto de tiempo como computacionales) disponibles.

5.1. Acotando los hiperparámetros

5.1.1. Ajuste del tamaño de *batch*

Para establecer el efecto de las variaciones del tamaño de *batch* sobre el rendimiento del clasificador se comenzó probando con una estructura básica de red y comprobando cómo afectaba al modelo la variación de este valor. Las prueba realizadas están plasmadas en la figura 5.1.

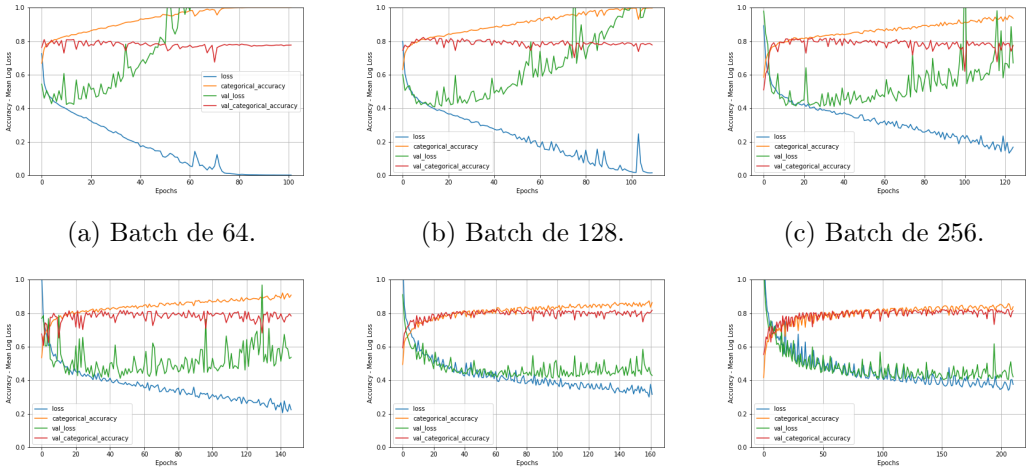
Se puede observar que según decrece el tamaño de batch aumenta considerablemente el sobrentrenamiento. Mientras que con el aumento del tamaño del batch disminuye:

$$\downarrow \text{Tamaño de } batch \implies \uparrow \text{Sobreentrenamiento}, \uparrow \text{ Tiempo de ejecución}$$

Como referencia se tomó un tamaño de *batch* de 512 que, con técnicas para evitar el sobreentrenamiento, era el que mejores resultados proporcionaba, como se puede ver en la figura **??**.

5.1.2. Elección de las funciones de activación

Otro conjunto de hiperparámetros estudiado en este trabajo fueron las funciones de activación para las capas ocultas, ya que la función de activación para la capa de salida se fijó como *Softmax* para asegurarse de que los datos de salida del clasificador pertenezcan al intervalo $[0, 1]$ y puedan, por tanto, ser usados como probabilidades predichas de cada clase.



(a) Batch de 64. (b) Batch de 128. (c) Batch de 256.

(d) Batch de 512. (e) Batch de 1024. (f) Batch de 2048.

Figura 5.1: Resultados de las pruebas respecto a la variación del batch.

De un estudio de la bibliografía sobre el tema se pudo comprobar que la función de activación para las capas ocultas de uso más extendido en problemas de clasificación multietiqueta era la ReLU [11]. No obstante, se probó también en modelos idénticos la función de activación *ELU* y en la mayoría de los casos se obtuvieron mejores resultados (ver tabla 3.3, donde se puede apreciar que la mayoría de los modelos que han proporcionado los mejores resultados empleaban función de activación ELU en sus capas ocultas, aunque no se pudo determinar el efecto de las diferentes funciones sobre el equilibrio entre sobreentrenamiento y subajuste.

5.1.3. Ajuste de la arquitectura neuronal

Se ha estudiado el efecto del número de capas y de neuronas por capa en el rendimiento del clasificador, estableciendo como referencia el uso de alrededor de cuatro capas ocultas con un número de neuronas relativamente bajo, en prácticamente todos los casos acotado por 500 neuronas en la capa más ancha. Respecto a la distribución de las neuronas. la estructura más eficaz ha sido la que toma forma triangular, estrechándose de la entrada hacia la salida.

5.1.4. Ajuste del número de épocas máximo y de la tasa de aprendizaje

Estos dos parámetros no se deben considerar por separado, ya que su equilibrio es clave para el correcto entrenamiento del modelo. Una tasa de aprendizaje baja necesitará de un alto número de épocas para converger, mientras que un número corto de épocas requerirá de una tasa de aprendizaje alta por el mismo motivo. Experimentalmente, se ha comprobado que:

$$\downarrow \text{Tasa de aprendizaje} \implies \uparrow \text{Sobreentrenamiento}, \uparrow \text{ Estabilidad del modelo}$$

$$\uparrow \text{Número de épocas} \implies \uparrow \text{Sobreentrenamiento}, \uparrow \text{Tiempo de entrenamiento}$$

En general, el valor escogido para el número de épocas ha sido de unas 1000, aunque se ha disminuído cuando se han empleado tamaños de *batch* mucho menores de 500 (por motivos de tiempo de ejecución), llegando hasta las 100 épocas máximas. Para este rango de valores del número máximo de épocas, la tasa de aprendizaje se ha fijado en torno a 0.1, aunque se ha probado con muchos valores alrededor de esta marca. En casos excepcionales se ha prolongado la ejecución hasta, incluso, 10000 épocas, lo que ha requerido una disminución de la tasa de aprendizaje.

Como idealmente la tasa de aprendizaje se desea alta al inicio del entrenamiento y más pequeña a medida que avanza el proceso para asegurar la convergencia [19], también se han empleado un **sistema de decaimiento** de la tasa de aprendizaje y un **sistema adaptativo** de selección de tasa de aprendizaje en algunos de los modelos entrenados, que ha proporcionado muy buenos resultados.

5.1.5. Ajuste de los parámetros de regularización

Respecto a los valores de los parámetros de regularización, se han utilizado como referencia *dropout* = 0,2 y $L2$ = 0,001, variándolos en función de la red pero siempre en torno a esos valores, que se han comprobado a ser los más efectivos para disminuir el sobreentrenamiento producido por un aumento del número de épocas o una disminución excesiva del tamaño de *batch*.

La normalización del *batch* se ha aplicado en algunos, lo que ha proporcionado una resistencia adicional al sobreentrenamiento en estos casos.

5.2. Modelos probados y resultados

Dado que el objetivo de la práctica es el análisis de los hiperparámetros y no el del preprocesado de datos, el entrenamiento de los modelos que se analizarán de cara al resultado final de la práctica se ha realizado sobre los ficheros de datos preprocesados originales, si bien luego, con la intención de conseguir un rendimiento óptimo, se ha estudiado a modo de ampliación el funcionamiento de los clasificadores desarrollados sobre datos preprocesados de diferentes formas.

A lo largo de esta sección se mostrarán algunos de los modelos más interesantes e ilustrativos que se han probado, pues es imposible incluir un informe completo de todas las redes entrenadas. De cada modelo se proporcionará la configuración escogida, la tabla con los resultados de las métricas tomadas, la gráfica que muestra el progreso del entrenamiento y la matriz de confusión. En todos ellos se han almacenado los pesos del modelo que haya alcanzado la mejor precisión sobre los datos de validación, por lo que los modelos expuestos no son necesariamente el último obtenido en el proceso de entrenamiento.

Los primeros modelos probados estaban configurados con los hiperparámetros que había de base en el programa. Como no tenían ningún tipo de regularización los resultados eran redes sobrentrenadas (como se puede apreciar en la figura 5.2), por lo que inmediatamente se comenzaron a introducir métodos de regularización.

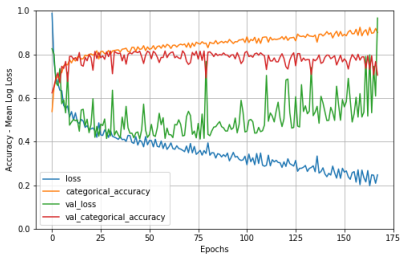


Figura 5.2: Modelo con los parámetros por defecto.

5.2.1. Mejores modelos obtenidos

La medida más importante para determinar la bondad de un clasificador es su nivel de acierto o *accuracy*, que se calcula siguiendo la fórmula:

$$\text{Acc} = \frac{TP + TN}{N}$$

Donde TP y TN son, respectivamente, el número de verdaderos positivos y negativos y N es el número total de ejemplos clasificados. A continuación se presentan los resultados de los mejores modelos obtenidos al introducir regularización y aplicarlos sobre el conjunto original de datos.

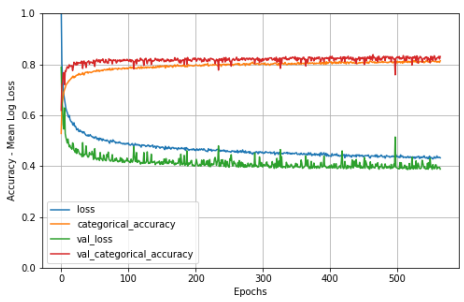
Para el primero que se va a presentar, la configuración de los hiperparámetros se puede ver en la tabla 5.1 y los resultados del modelo con los datos de test en la tabla 5.2. También podemos ver el gráfico del entrenamiento y la matriz de confusión en Fig.5.3.

	Épocas	LR	Batch	Neuronas por capa	Funciones de activación	Función de salida	Dropout	BatchNormalization	L2
Modelo1	10.000	0,1	512	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001

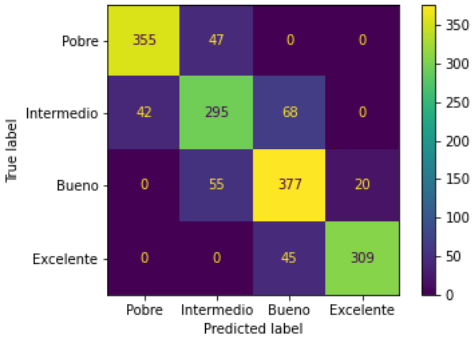
Tabla 5.1: Configuración para el primer modelo.

	precision	**recall**	**f1-score**	**support**
Pobre	0,89	0,88	0,89	402,00
Intermedio	0,74	0,73	0,74	405,00
Bueno	0,77	0,83	0,80	452,00
Excelente	0,94	0,87	0,90	354,00
accuracy	0,83	0,83	0,83	0,83
macro avg	0,84	0,83	0,83	1613,00
weighted avg	0,83	0,83	0,83	1613,00

Tabla 5.2: Resultados del primer modelo.



(a) Evolución del entrenamiento



(b) Matriz de confusión

Figura 5.3: Métricas obtenidas del primer modelo.

El *accuracy* de este modelo es el más alto obtenido sobre el set de datos sin procesar, llegando hasta 0,83. En el gráfico se puede apreciar cómo converge del algoritmo y de qué forma evoluciona la precisión. En este caso, vemos que converge de forma asintótica correctamente y sin sufrir de sobreentrenamiento, pues apenas hay *gap* entre el valor del *accuracy* sobre el set de entrenamiento y sobre el set de validación.

Igualmente, la matriz de confusión muestra que en la mayoría de los casos las predicciones se realizan de forma correcta, y, si acaso, las instancias mal clasificadas se han etiquetado en las categorías adyacentes.

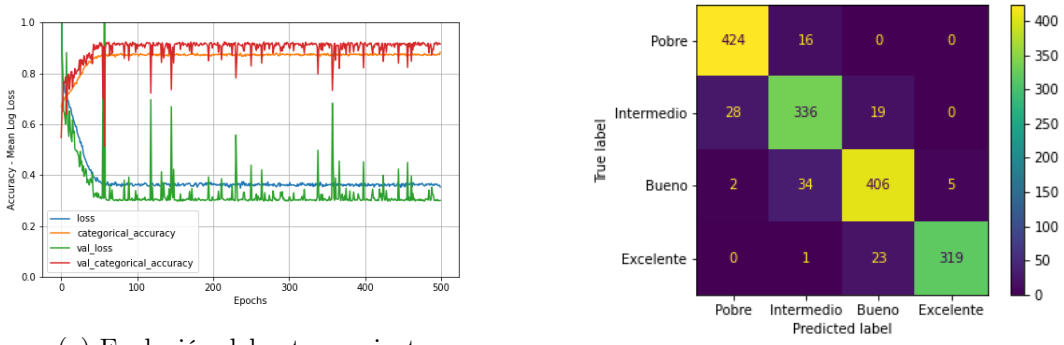
Este es el modelo con mayor nivel de acierto que se ha conseguido utilizando el preprocesado de los datos proporcionado para la práctica. Sin embargo, modificando el preprocesado tal y

como se introdujo en el Capítulo 2 se ha podido mejorar el nivel de acierto incluso por encima de este umbral. En la figura 5.4 se muestra un ranking de los 15 mejores modelos obtenidos, esta vez entrenados sobre los datos preprocesados sesgados bajo un umbral del 60 % de correlación con la calificación general del jugador.

Epocas	LR	Batch	Neuronas_por_capa	act_int	act_out	Dropout	BatchNormalization	L2	precision_test
500	scheduled	32	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,921
500	scheduled	64	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,918
500	scheduled	32	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,910
80000	0,0005	512	[75, 50, 25, 10]	[elu, elu, elu, elu]	softmax	0	0	0	0,903
55000	0,0005	512	[75, 50, 25, 10]	[elu, elu, elu, elu]	softmax	0	0	0	0,898
100	scheduled	16	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,890
500	scheduled	32	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0,1	1	0,01	0,885
1000	0,1	512	[100, 80, 60, 40, 20, 10]	[elu, elu, elu, elu, elu]	softmax	0,05	1		0,884
500	scheduled	16	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,872
20000	0,001	512	[75, 50, 25, 10]	[elu, elu, elu, elu]	softmax	0	0	0	0,868
1000	0,1	512	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001	0,867
500	scheduled	32	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,867
500	scheduled	16	[100, 80, 60, 40, 20, 10]	[relu, relu, relu, relu, relu]	softmax	0	1	0,01	0,863
1000	0,1	512	[500, 250, 75, 25]	[elu, elu, elu, elu]	softmax	0,2	1	0,001	0,862
1000	0,1	512	[100, 80, 60, 40, 20, 10]	[elu, elu, elu, elu, elu]	softmax	0,05	1		0,850

Figura 5.4: Top 15 modelos entrenados con otros preprocesados.

En particular, es interesante el primer modelo de este ranking, que obtuvo un nivel de acierto en el set de datos de prueba del 92 %. Fue entrenado haciendo uso de pequeños minibatches de 32 datos y utilizando una tasa de aprendizaje regida por un modelo de decaimiento exponencial. Sus resultados se pueden apreciar en la figura 5.5.



(a) Evolución del entrenamiento

(b) Matriz de confusión

Figura 5.5: Evolución de entrenamiento y matriz de confusión del mejor modelo.

Capítulo 6

Conclusiones

Tras el estudio de los resultados, las conclusiones más relevantes extraídas son las siguientes:

- Los resultados obtenidos y las variaciones producidas por las modificaciones de los hiperparámetros viene muy condicionada por el número de datos de entrenamiento, ya que se cuenta con tan solo 16.122 y el rendimiento del clasificador habría aumentado considerablemente de disponer de un *dataset* mayor.
- Es conveniente en muchas ocasiones reducir el número de neuronas antes que introducir recursos de regularización, tal y como demuestra el hecho de que el modelo que mejor ha funcionado sobre los datos preprocesados originales explotaba precisamente esta propiedad.
- Las técnicas de aprendizaje automático como XGBoost (que hemos utilizado para intentar mejorar la selección de características, sobrepasando el 0.95 de nivel de acierto) proporcionan un mejor nivel de acierto que los modelos neuronales bajo los mismos datos de entrenamiento, lo que permite concluir que para este problema, debido a la escasez de datos, un enfoque basado en redes neuronales no es óptimo.
- El uso de funciones de activación 'ELU' en lugar de 'RELU' ha proporcionado mejores resultados para la tarea en cuestión.

Por otro lado, nótese que la inicialización de los pesos se ha realizado con una misma semilla por todo el experimento, pues si variamos un hiperparámetro pero los pesos se inicializan en cada prueba de diferente manera, no podremos ser capaces de conocer si la mejora o el empeoramiento producido se debe a la modificación de dicho parámetro o a que los pesos se han inicializado de una manera más cerca o más lejos de la óptima.

De cara a futuros trabajos sobre el tema, se podría tratar mejorar la precisión de los modelos usando aquellos que convergen sin una semilla en la inicialización de los pesos, pues no se ha podido considerar dicha posibilidad en esta práctica.

Bibliografía

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] O. Knocklein, "Classification using neural networks," 2019. [Online]. Available: https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f

[3] K. Chandrakant, "Reinforcement learning with neural network," 2020. [Online]. Available: https://www.baeldung.com/cs/reinforcement-learning-neural-network

[4] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988. [Online]. Available: http://www.jstor.org/stable/2685263

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[6] J. Brownlee, "Why one-hot encode data in machine learning?" 2017. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[7] ——, "Use early stopping to halt the training of neural networks at the right time," 2018. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

[8] Keras, "Earlystopping." [Online]. Available: https://keras.io/api/callbacks/early_stopping/

[9] J. Brownlee, "Understand the impact of learning rate on neural network performance," 2020. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[10] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *CoRR*, vol. abs/1804.07612, 2018. [Online]. Available: http://arxiv.org/abs/1804.07612

[11] J. Brownlee, "How to choose an activation function for deep learning," 2021. [Online]. Available: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[12] A. Budhiraja, "Dropout in (deep) machine learning," 2016. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334c

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[14] N. Tyagi, "L2 and l1 regularization in machine learning," 2021. [Online]. Available: https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning

[15] S. C. Nerella, "Loss functions in neural networks," 2020. [Online]. Available: https://becominghuman.ai/loss-functions-in-neural-networks-ec6482a15e97

[16] A. Demirkaya, J. Chen, and S. Oymak, "Exploring the role of loss functions in multiclass classification," in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, 2020, pp. 1–5.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] S. L. Team, "sklearn.metrics.classification_report," 2011. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[19] S. Lau, "Learning rate schedules and adaptive learning rate methods for deep learning," 2017. [Online]. Available: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

ELCOSTE