Universidad Politécnica de Madrid

Máster Universitario en Inteligencia Artificial

Práctica 1: FIFA Players Classification

Asignatura: Artificial neural networks and deep learning

Autores:
Juan José Flores Arellano
Borja Reinoso Hidalgo
Daniel Carmona Pedrajas

Correos:
jj.flores.arellano@alumnos.upm.es
borja.reinoso@alumnos.ump.es
daniel.carmonap@alumnos.upm.es

16 Diciembre 2022

Índice

1. Introducción

Este proyecto tiene como objetivo diseñar una red neuronal artificial para resolver un problema de clasificación supervisada de jugadores del videojuego FIFA 19. Para ello se hará uso de las técnicas y herramientas aprendidas en la clase de "Redes de neuronas artificiales y deep learning".

Se hará uso del lenguaje de programación Python 3 junto a las librerías Tensorflow 2 y Keras. Los entornos utilizados serán: Jupyter notebook en local y Google Colab.

El dataset a utilizar es . Este dataset contiene 18.207 instancias con 89 atributos cada una. Tras un preprocesado se han reducido tanto las instancias como los atributos a utilizar, 16.122 instancias con 17 atributos cada una. Los atributos están relacionados con la habilidad de cada jugador en ciertas facetas relacionadas con el fútbol: Crossing, Heading Accuracy, Pase Corto, Voleas, Regate, Curva, Precisión de Tiro Libre, Pase Largo, Control del Balón, Reacción, Potencia de Tiro, Resistencia, Tiros lejanos, Agresividad, Posicionamiento, Visión y Compostura.

Para etiquetar las clases se ha hecho una media de los atributos de los jugadores y se ha repartido de la siguiente manera:

1. Poor para medias entre [46,62]
2. Intermediate para medias entre [63,66]
3. Good para medias entre [67,71]
4. Excellent para medias entre [72,94]

1.1. Estructura

La investigación se divide en dos fases:

- Familiarización con hiperparámetros y técnicas de regularización.
- Optimización sistemática de hiperparámetros

En ambas partes realizaremos cambios atómicos para comprobar el efecto de los mismos en el modelo, pero en la primera fase trabajamos de una forma reactiva, solucionando los problemas que nos surgen para comprender el impacto de cada cambio que hacemos. Una vez interiorizados estos efectos pasamos a comprobar distintos valores para cada hiperparámetro para quedarnos con el mejor, fijarlo en la configuración y pasar al siguiente.

Por último, presentaremos nuestras conclusiones.

2. Fase 1: Cambios reactivos

Como hemos comentado, en esta fase iremos realizando pequeños cambios, observando cómo afectan a los resultados del modelo y al proceso de entrenamiento.

2.1. Arquitectura 1: Feed Forward Neural Network

La primera arquitectura que usaremos es tan simple como:

1. Capa densa con 512 neuronas.

2.1.1. Experimento 1: Configuración base arbitraria

Usamos la siguiente configuración:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
100	0.1	512	ReLU	Categorical Crossentropy	SGD	None

Tabla 1: Hiperparámetros para el Experimento 1 de la Arquitectura 1

Y entrenamos 5 veces para obtener los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	79.38	77.94	15.61	1.44	14
Std	0.05	0.14	0.05	0.19	0

Tabla 2: Resultados del Experimento 1 de la Arquitectura 1

Tener un *bias* alto y una *variance* baja significa que hay margen de mejora antes de llegar al *overfitting* y hay varias posibilidades para conseguir una mejor *accuracy*: añadir más neuronas, entrenar con más *epochs*, ...

2.1.2. Experimento 2: Aumentamos *epochs*

Tras el experimento anterior, nos decantamos por entrenar el modelo durante más *epochs* para reducir el *bias* usando la misma configuración.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	512	ReLU	Categorical Crossentropy	SGD	None

Tabla 3: Hiperparámetros para el Experimento 2 de la Arquitectura 1

Tras 5 entrenamientos obtenemos los siguientes resultados: Con respecto al experimento anterior hemos

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	84.02	81.5	10.97	2.47	199
Std	0.03	0.17	0.03	0.18	8.8

Tabla 4: Resultados del Experimento 2 de la Arquitectura 1

aumentado el *accuracy* tanto en el entrenamiento como en validación, reduciendo así el *bias* del modelo en un 5 % aunque ha aumentado ligeramente el *variance*. Como es lógico el tiempo de entrenamiento ha crecido, aunque no de forma lineal.
En la figura 1 vemos que a partir del epoch 400 no hay una mejora en *accuracy* para el conjunto de validación aunque sí para el conjunto de entrenamiento lo que nos indica que un número de *epochs* tan elevado como el que

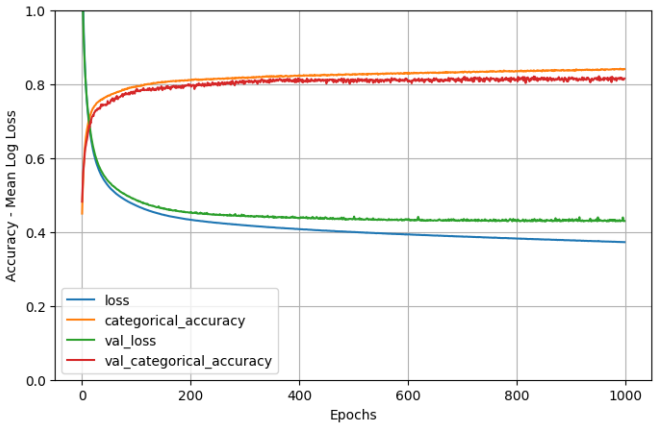


Figura 1: Entrenamiento durante el Experimento 2 de la Arquitectura 1

hemos usado en este experimento con esta arquitectura y configuración conduce a un *overfitting* del modelo, aunque por el momento no es excesivo como nos indica la *variance*.

2.1.3. Experimento 3: Cambiamos a *tanh* y reducimos *epochs*

Para este experimento decidimos reducir las *epochs* ya que como hemos visto en el experimento anterior, no hay una mejora significativa en validación con más epochs.
Además de esto, cambiaremos la función de activación a *tanh*. Hasta el momento hemos usado $ReLU$ pero no hay razón para usarla para esta arquitectura porque resuelve el problema del *vanishing gradient* que se da en arquitecturas profundas.
La configuración que usamos para el experimento 3 es:

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	0.1	512	**tanh**	Categorical Crossentropy	SGD	None

Tabla 5: Hiperparámetros para el Experimento 3 de la Arquitectura 1

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	79.58	78.3	15.41	1.28	72
Std	0.2	0.51	0.22	0.33	3.27

Tabla 6: Resultados del Experimento 3 de la Arquitectura 1

Con esta configuración el modelo ha vuelto a aumentar el *bias* y obtenemos unos resultados prácticamente idénticos al experimento uno con la función de activación $ReLU$ aunque con más epochs. Parece que con la función *tanh*, el modelo se queda atrapado en mínimos globales como podemos apreciar en la figura 2 y necesita más epochs para escapar de ellos.

También observamos que no ha habido overfitting hasta la epoch 400, al contrario de lo que habíamos supuesto al inicio de este experimento.

2.1.4. Experimento 4: Aumentamos *epochs*

El objetivo de este experimento es comprobar cuántas *epochs* podemos realizar antes de que el modelo comience a dirigirse hacia un *overfitting* por lo que la configuración es la misma que en la ejecución anterior, excepto que volvemos a incrementar las epochs a 1000:

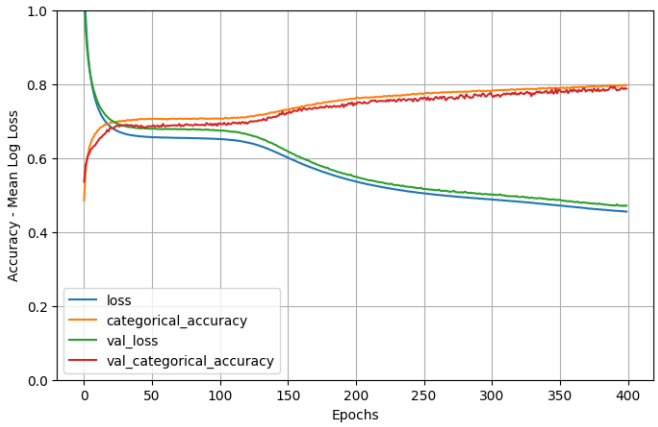


Figura 2: Entrenamiento durante el Experimento 3 de la Arquitectura 1

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
1000	0.1	512	tanh	Categorical Crossentropy	SGD	None

Tabla 7: Hiperparámetros para el Experimento 4 de la Arquitectura 1

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	81.17	80.23	13.82	0.94	189
Std	0.14	0.62	0.14	0.53	2.3

Tabla 8: Resultados del Experimento 4 de la Arquitectura 1

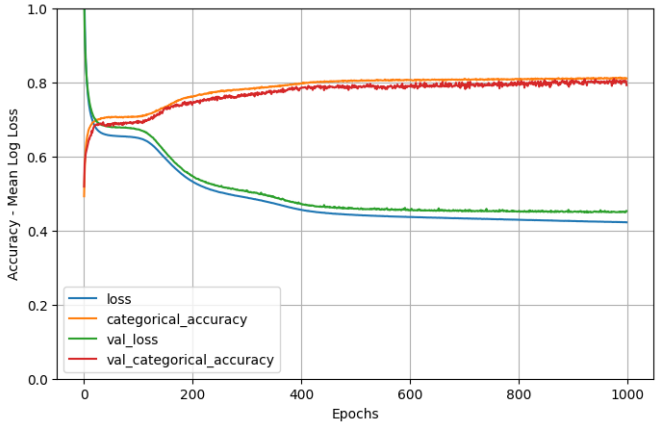


Figura 3: Entrenamiento durante el Experimento 4 de la Arquitectura 1

Aunque hemos doblado las *epochs* con respecto al experimento 3, la mejora ha sido de apenas un 2 % en *accuracy*. Por otra parte, como se muestra en la figura 3, el modelo no ha llegado al punto de *overfitting* aun habiendo usado un número tan alto de epochs.
Esto quiere decir que todavía hay margen de mejora si seguimos entrenando con más epochs aunque llevaría mucho tiempo porque el aprendizaje es lento.

2.1.5. Experimento 5: Reducimos *batch size*

Como se necesitaría aumentar exponencialmente el número de *epochs* para conseguir una mejora en el *accuracy*, decidimos reducir el *batch size* y comprobar si de esta forma el modelo consigue mejores resultados.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	**128**	tanh	Categorical Crossentropy	SGD	None

Tabla 9: Hiperparámetros para el Experimento 5 de la Arquitectura 1

Tras 5 entrenamientos obtenemos los siguientes resultados:
Como vemos en la tabla 10 no conseguimos mejorar de forma significativa el *accuracy*. Aumentar el *batch size*

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	82.92	80.84	12.07	2.08	517.2
Std	0.19	0.29	0.19	0.28	25.61

Tabla 10: Resultados del Experimento 4 de la Arquitectura 1

ha incrementado exponencialmente el tiempo de entrenamiento por lo que no merece la pena reducir el *batch size* en este caso. Lo que hemos conseguido reduciendo el *batch size* ha sido que el modelo escape rápidamente del primer mínimo local con el que se topa como vemos en la figura 4 así que concluimos que con esta técnica podemos obtener un resultado aceptable en menor tiempo.

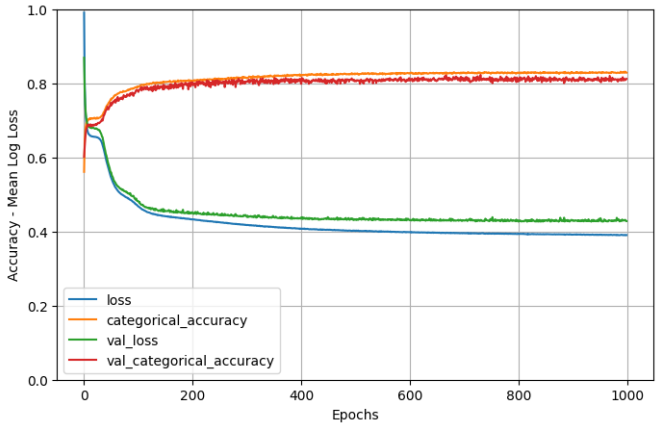


Figura 4: Entrenamiento durante el Experimento 5 de la Arquitectura 1

2.1.6. Conclusiones de la Arquitectura 1

- *ReLU* llega a un estado de *overfitting* con menos *epochs* que *tanh*
- *tanh* tiene un proceso de aprendizaje más lento que *ReLU*
- Reducir el *batch size* implica llegar a un óptimo de forma más rápida con *tanh*.

2.2. Arquitectura 2: Deep Feed Forward Neural Network

Hemos visto que con la arquitectura anterior obtenemos un *accuracy* máximo de un 84 % a partir de donde el aprendizaje es lento y llegamos a un *overfitting* con las técnicas utilizadas. Es por esto que decidimos usar una Deep Feed Forward Neural Network para intentar reducir el *bias* del modelo.
La arquitectura que utilizaremos para la nueva serie de experimento será:

1. Capa densa de 128 neuronas
2. Capa densa de 64 neuronas
3. Capa densa de 32 neuronas

2.2.1. Experimento 1: Comparación con arquitectura 1

En este experimento utilizaremos la misma configuración que en el experimento anterior:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
1000	0.1	128	tanh	Categorical Crossentropy	SGD	None

Tabla 11: Hiperparámetros para el Experimento 1 de la Arquitectura 2

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	99.89	74.97	-4.89	24.91	578
Std	0.04	0.26	0.04	0.26	4.54

Tabla 12: Resultados del Experimento 1 de la Arquitectura 2

Hemos obtenido un modelo con un *overfitting* muy alto, tan alto que ha cometido menos error que un humano y por eso el *bias* es negativo. Podemos ver el *overfitting* reflejado en el *variance* de un 25 % de la tabla 12 y la evolución del *validation categorical accuracy* en la figura 5 que empeora a lo largo del entrenamiento.

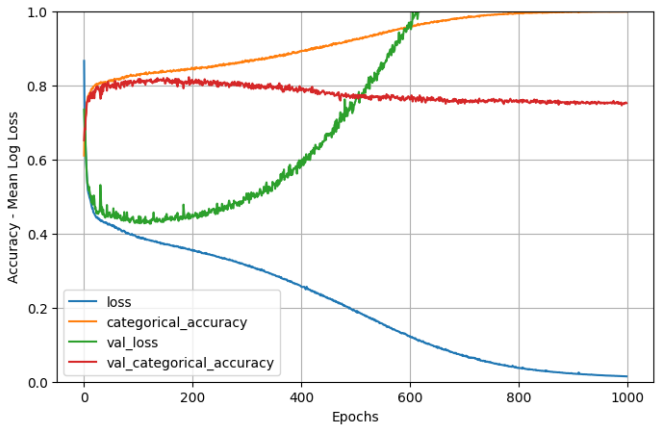


Figura 5: Entrenamiento durante el Experimento 1 de la Arquitectura 2

2.2.2. Experimento 2: Reducimos *epochs* y utilizamos regularización

Una de las técnicas que podemos utilizar para reducir el *variance* es usar regularización para controlar que los pesos de las neuronas no se hagan demasiado grande. Además para ahorrar algo de tiempo también reduciremos las epochs. La configuración para este experimento queda reflejada en la tabla 13.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.1	128	tanh	Categorical Crossentropy	SGD	**l2 0.001**

Tabla 13: Hiperparámetros para el Experimento 2 de la Arquitectura 2

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	81.19	79.91	13.8	1.28	249
Std	0.2	0.95	0.2	0.92	3.29

Tabla 14: Resultados del Experimento 2 de la Arquitectura 2

Como se observa en la tabla 14, hemos conseguido reducir el *variance* un 24 % simplemente usando regularización, ahora bien, en cuanto a *accuracy* seguimos sin conseguir una mejora significativa.

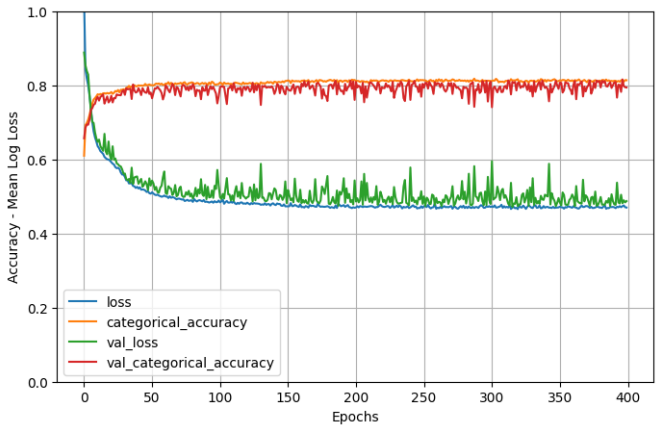


Figura 6: Entrenamiento durante el Experimento 2 de la Arquitectura 2

En la figura 6 se aprecia ruido en las métricas del conjunto de validación, esto es una consecuencia de la regularización, sin ella ese ruido se descontrolaría y llegaríamos al *overfitting* como en el experimento anterior.

2.2.3. Experimento 3: Cambiamos el optimizador

En este experimento vamos a comprobar si con un optimizador distinto podemos obtener mejor *accuracy* sin llegar al *overfitting*, para ello utilizaremos ADAM. La configuración es la siguiente:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.1	128	tanh	Categorical Crossentropy	**ADAM**	l2 0.001

Tabla 15: Hiperparámetros para el Experimento 3 de la Arquitectura 2

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	25.34	26.25	69.65	-0.91	266.8
Std	0.29	2.61	0.29	2.44	4.81

Tabla 16: Resultados del Experimento 3 de la Arquitectura 2

Como vemos en la tabla resultados son los peores obtenidos hasta el momento y con mucha diferencia, apenas hemos conseguido un 25 % de *accuracy*. Analizando la figura 7, el modelo se queda estancado en un mínimo local muy temprano en el entrenamiento.

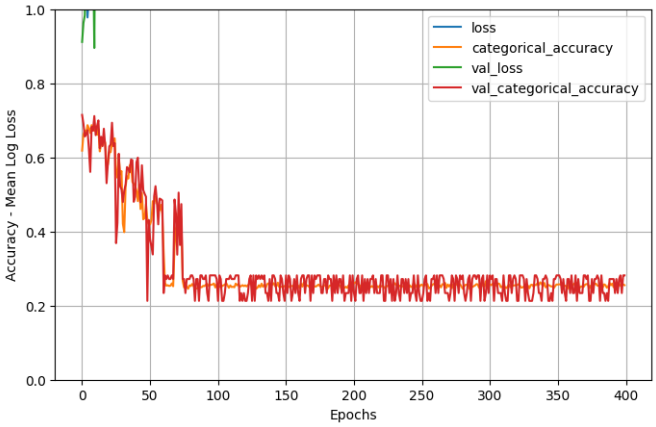


Figura 7: Entrenamiento durante el Experimento 4 de la Arquitectura 2

Observando la matriz de confusión confirmamos nuestra teoría, clasifica todos los jugadores en la misma clase.

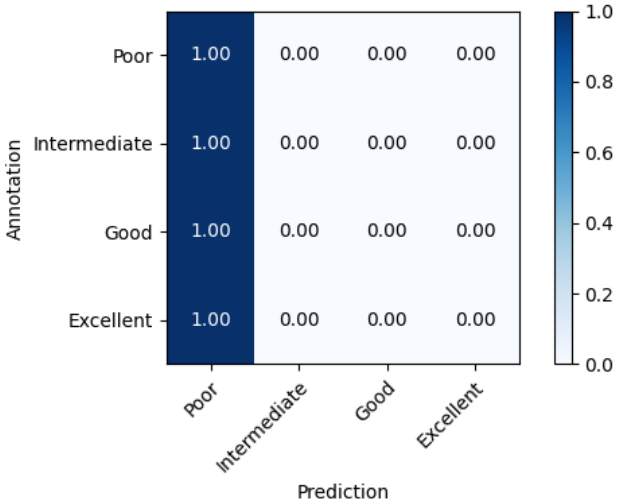


Figura 8: Matriz de confusión en el Experimento 3 de la Arquitectura 2

Esto puede deberse a que no estamos usando correctamente el optimizador ADAM.

2.2.4. Experimento 4: Ajustamos el optimizador ADAM

Consultando el artículo en el que se presentó ADAM (*Adam: A Method for Stochastic Optimization*, (2015), D. Kingma y J. Ba), vemos que este optimizador es una combinación entre AdaGrad y RMSProp y que tiene unos parámetros que controlan el ratio de decrecimiento de los momentos (beta1=0.9 y beta2=0.999) que no hemos definido en el experimento anterior. Además de esto, los autores sugieren usar un *learning rate* más bajo del que estamos usando.

La configuración queda así:

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	**0.001**	128	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 17: Hiperparámetros para el Experimento 4 de la Arquitectura 2

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	81.97	80.48	13.02	1.49	250.2
Std	0.15	0.15	0.15	0.19	19.35

Tabla 18: Resultados del Experimento 4 de la Arquitectura 2

Al haber configurado correctamente el optimizador volvemos a conseguir un *accuracy* similar al que hemos estado obteniendo. Observando la figira 9 vemos qe podríamos seguir entrenando el modelo por más *epochs* antes de llegar al *overfitting*. Aun con todo esto, no hemos obtenido mejores resultados que con el optimizador *SGD*.

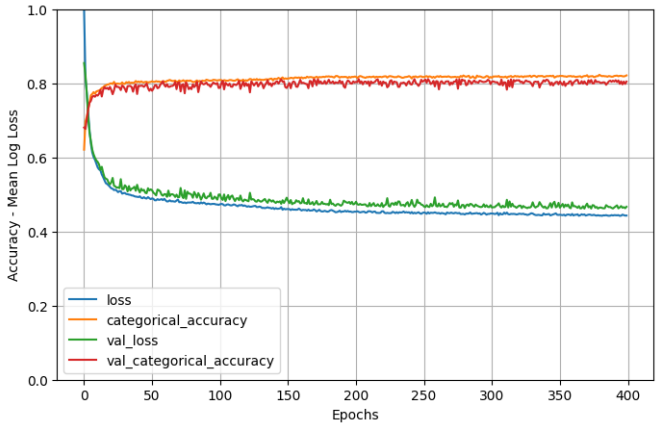


Figura 9: Entrenamiento durante el Experimento 4 de la Arquitectura 2

Analizando la matriz de confusión en la figura 10 vemos que ahora hemos solucionado el problema del modelo anterior que clasificaba todos los jugadores en la misma clase

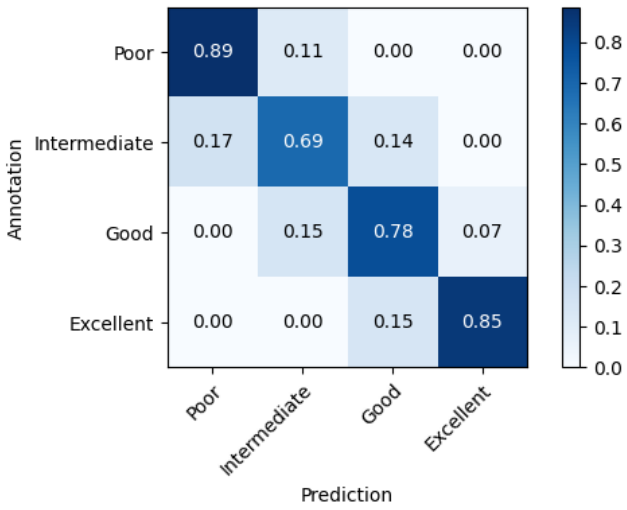


Figura 10: Matriz de confusión en el Experimento 4 de la Arquitectura 2

2.2.5. Experimento 5: Reducimos *batch-size*

Decidimos reducir el *batch size* para intentar reducir el *bias* como en el experimento 2.1.5. La configuración para este experimento es:

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	0.001	**64**	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 19: Hiperparámetros para el Experimento 5 de la Arquitectura 2

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	81.87	80.03	13.12	1.83	506.4
Std	0.22	0.77	0.22	0.89	38.48

Tabla 20: Resultados del Experimento 5 de la Arquitectura 2

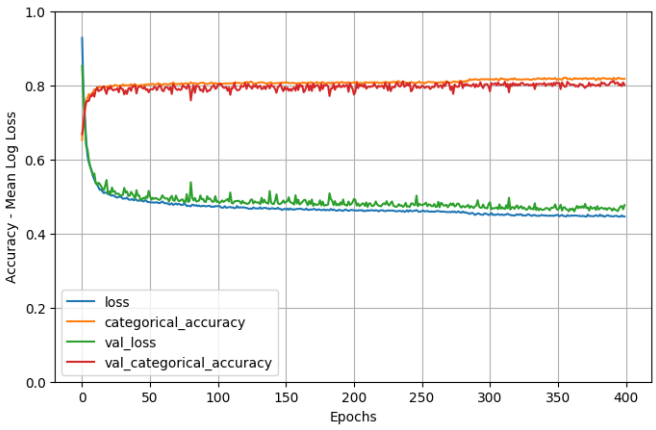


Figura 11: Entrenamiento durante el Experimento 5 de la Arquitectura 2

No hemos obtenido mejora, este cambio no merece la pena porque aumenta mucho el tiempo de entrenamiento. Como vemos en la figura 11, parece que el modelo deja de aprender, quizás por el problema del *vanishing gradient* que intentaremos resolver con la siguiente arquitectura.

2.2.6. Conclusiones de la Arquitectura 2

A lo largo de estos experimentos hemos concluido que:

- La regularización evita el *overfitting*.
- El optimizador ADAM necesita un *learning rate* bajo.
- El optimizador ADAM no ha proporcionado mejoras significativas con respecto al SGD para esta arquitectura.

2.3. Arquitectura 3: Batch normalization

Después de no conseguir reducir el *bias* de nuestro modelo con más capas ocultas suponemos que estamos ante un problema de *vanishing gradient*, una de las soluciones que existe es utilizar capas de *batch normalization* para estandarizar los pesos de las neuronas. La arquitectura que utilizaremos durante los siguientes experimentos es:

1. Capa densa de 128 neuronas
2. Capa de *Batch Normalization* antes de la activación.
3. Capa densa de 64 neuronas
4. Capa de *Batch Normalization* antes de la activación.
5. Capa densa de 32 neuronas
6. Capa de *Batch Normalization* antes de la activación.

2.3.1. Experimento 1: Probamos la configuración del experimento 2.2.5

Para comprobar como afecta la introducción del *batch normalization*, utilizamos la configuración del último experimento realizado.

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	0.001	64	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 21: Hiperparámetros para el Experimento 1 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	89.23	76.23	5.76	15.59	950.4
Std	0.46	1.54	0.46	1.75	55.11

Tabla 22: Resultados del Experimento 1 de la Arquitectura 3

La tabla 22 nos muestra que con *batch normalization* hemos obtenido una buena *accuracy* en el conjunto de entrenamiento pero a cambio de obtener un modelo con *overfitting* como nos indica el *variance* de 15 %. También ha sido la ejecución más larga dado el tamaño del batch y que hemos usado *batch normalization* que ralentiza el proceso de entrenamiento.

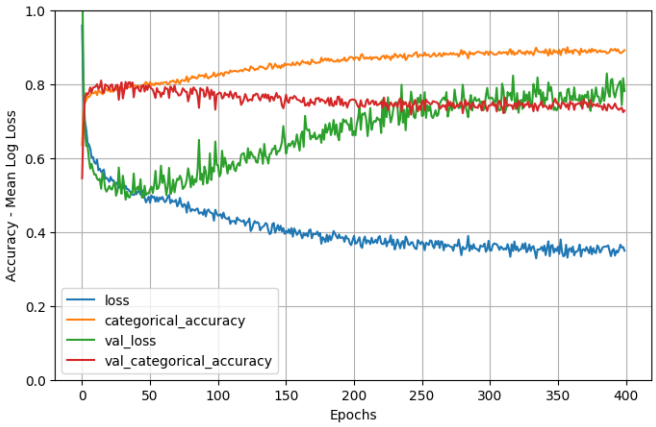


Figura 12: Entrenamiento durante el Experimento 1 de la Arquitectura 3

En la figura 12 vemos como el modelo llega al punto de *overfit* alrededor de la *epoch* 50 donde el *accuracy* de validación se separa del de entrenamiento.

2.3.2. Experimento 2: Aumentamos *batch size*

Aún no hemos conseguido mejorar el *accuracy* en el conjunto de validación, pero en este experimento vamos a intentar retrasar el *overfitting* del modelo a la vez que el tiempo de entrenamiento aumentando el *batch size*. La configuración para este experimento es la siguiente:

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	0.001	**256**	tanh	Categorical Crossentropy	ADAM	l2 0.001

Tabla 23: Hiperparámetros para el Experimento 2 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	97.02	73.89	-2.02	23.17	254
Std	0.53	0.48	0.53	0.61	14.04

Tabla 24: Resultados del Experimento 2 de la Arquitectura 3

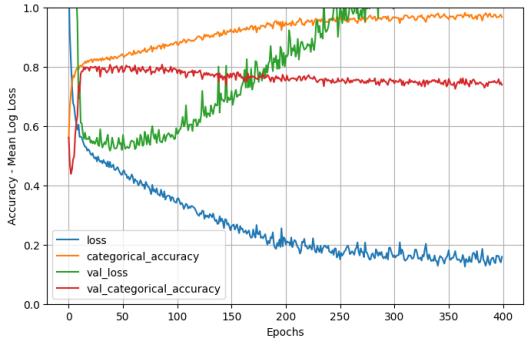


Figura 13: Entrenamiento durante el Experimento 2 de la Arquitectura 3

Lo único que hemos conseguido reducir con este experimento ha sido el tiempo de entrenamiento, el *variance* ha aumentado hasta un 23 %. En la figura 13 vemos reflejado el *overfitting*, lo que explica el *bias* negativo.

2.3.3. Experimento 3: Aumentamos la penalización del regularizador

El regularizador L2 añade una penalización en base al tamaño de los pesos, como no queremos que se descontrolen como ha estado ocurriendo, aumentamos esta penalización para evitar el overfitting. La configuración para este experimento es:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	tanh	Categorical Crossentropy	ADAM	**l2 0.1**

Tabla 25: Hiperparámetros para el Experimento 3 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	80.92	75.14	14.08	5.72	255.2
Std	0.51	2.41	0.51	2.27	9.25

Tabla 26: Resultados del Experimento 3 de la Arquitectura 3

En este experimento hemos conseguido reducir considerablemente el *variance* así que podemos concluir que el aumento de la penalización del regularizador sirve para evitar el *overfitting*. En la figura 14 observamos una

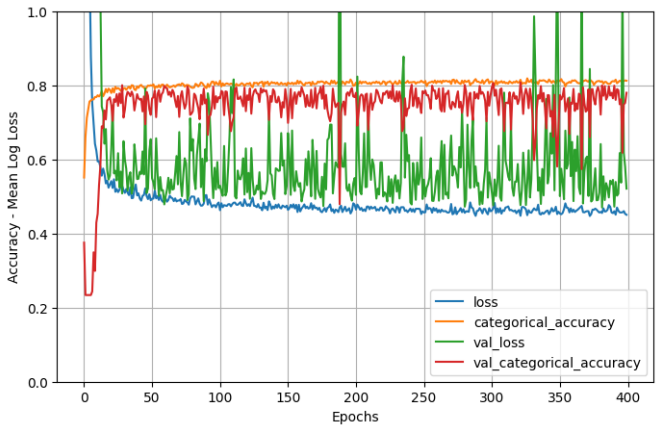


Figura 14: Entrenamiento durante el Experimento 3 de la Arquitectura 3

evolución errática en el *loss* en validación lo que puede indicar que el problema del vanishing gradient puede estar presente en el modelo.

2.3.4. Experimento 4: Volvemos a ReLU

Cambiamos la función de activación a ReLU para comprobar si ayuda con el problema del *vanishing gradient*.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	**ReLU**	Categorical Crossentropy	ADAM	l2 0.1

Tabla 27: Hiperparámetros para el Experimento 4 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	80.92	77.03	14.08	4.07	258.4
Std	0.46	1.8	0.46	1.56	7.46

Tabla 28: Resultados del Experimento 4 de la Arquitectura 3

Hemos obtenido unos resultados similares al experimento anterior, además en la figura 15 se aprecia que no hay una fase de entrenamiento tan errática.

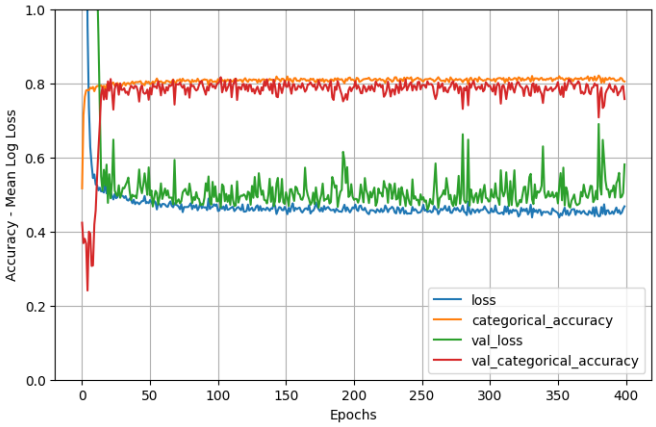


Figura 15: Entrenamiento durante el Experimento 4 de la Arquitectura 3

2.3.5. Experimento 5: Eliminamos la regularización

Nos preguntamos si tantos mecanismos contra el *vanishing gradient* y *overfitting* (*batch normalization*, ReLU, regularización L2) están interfiriendo entre ellos así que para comprobarlo retiramos la regularización L2 de las capas ocultas.

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	**No**

Tabla 29: Hiperparámetros para el Experimento 5 de la Arquitectura 3

Tras 5 entrenamientos obtenemos los siguientes resultados:

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Mean	97.31	74.55	-2.31	22.76	248
Std	0.44	0.99	0.44	0.68	6.12

Tabla 30: Resultados del Experimento 5 de la Arquitectura 3

Definitivamente la regularización es necesaria para evitar el *overfitting*

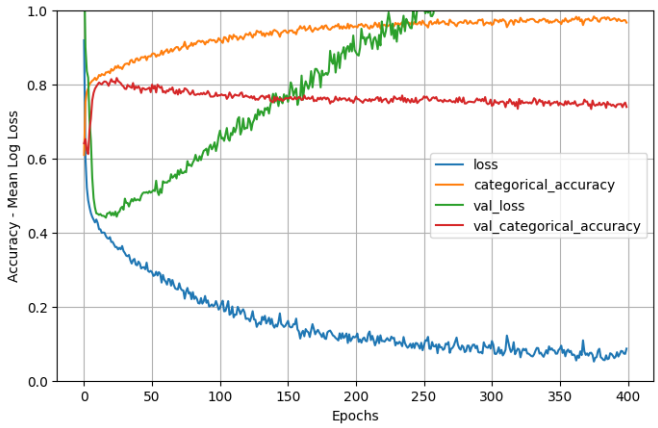


Figura 16: Entrenamiento durante el Experimento 5 de la Arquitectura 3

2.3.6. Conclusiones de la Arquitectura 3

- La regularización L2 es necesaria para evitar el *overfitting*.
- A mayor penalización en L2 menor *overfitting*.
- ReLU ayuda a mitigar el *vanishing gradient*.
- Batch Normalization no ayuda a reducir el *bias*.

2.4. Arquitectura 4: Añadimos profundidad

Hasta el momento el máximo *accuracy* en validación que hemos conseguido ha sido en el experimento 2.1.2 con un 81.5 %, nos parece que es un resultado malo para un problema de juguete como es este. Tras haber intentado distintos métodos para reducir el *bias* sin aumentar el *variance* sin éxito decidimos aumentar la profundidad de la red a 7 capas:

1. Capa densa de 2048 con Batch Normalization.
2. Capa densa de 1048 con BN.
3. Capa densa de 512 con BN.
4. Capa densa de 256 con BN.
5. Capa densa de 128 con BN.
6. Capa densa de 64 con BN.
7. Capa densa de 32 con BN.

Cabe destacar que hemos estado repitiendo cada experimento 5 veces y no hemos encontrado diferencias significativas entre ellos, además una muestra aleatoria de tamaño 5 no suele ser representativa del conjunto total. Creemos que el objetivo de esta práctica es variar los hiperparámetros y analizar su efecto en el modelo así que a partir de este punto solo realizaremos cada experimento otra vez para poder probar más combinaciones.

2.4.1. Experimento 1: Repetimos la última configuración

Vamos a usar la configuración del experimento 2.3.5 para comprobar como afecta el cambio de arquitectura.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	No

Tabla 31: Hiperparámetros para el Experimento 1 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
99.88	74.13	-4.88	25.87	426

Tabla 32: Resultados del Experimento 1 de la Arquitectura 4

Como vemos hemos vuelto a obtener *overfitting*, dado que no hemos reintroducido la regularización desde el experimento 2.3.5.

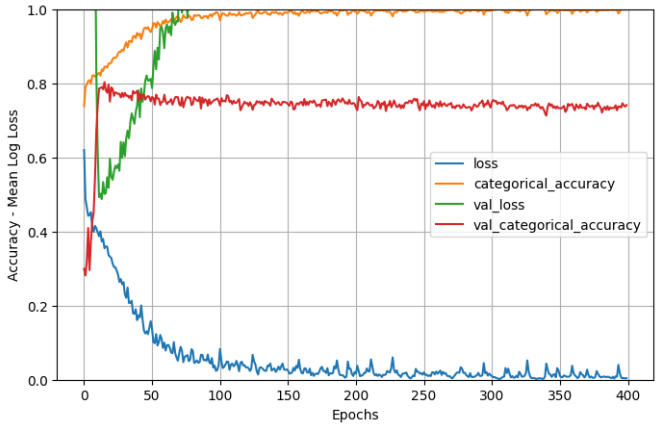


Figura 17: Entrenamiento durante el Experimento 1 de la Arquitectura 4

2.4.2. Experimento 2: Reintroducimos la regularización L2

Para evitar el *overfitting* volvemos a usar la regularización L2 con penalización 0.1

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	**l2 0.1**

Tabla 33: Hiperparámetros para el Experimento 2 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.99	74.75	14.01	6.24	457

Tabla 34: Resultados del Experimento 2 de la Arquitectura 4

Hemos evitado que el modelo haga *overfitting* pero en la figura 18 se ve un entrenamiento demasiado errático en cuanto al conjunto de validación, el *loss* varía enormemente en cada iteración, puede ser porque estemos estancados en un mínimo local.

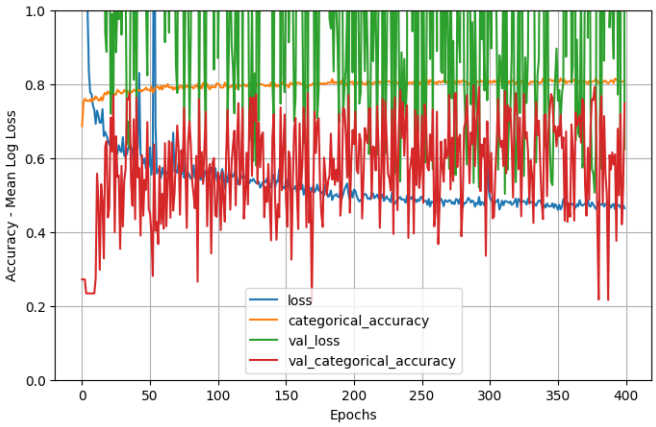


Figura 18: Entrenamiento durante el Experimento 2 de la Arquitectura 4

En la matriz de confusión, hemos detectado que por primera vez las clases Ïntermediatez "Good"son las que mayor *recall* tienen lo que refuerza nuestra teoría del mínimo local.

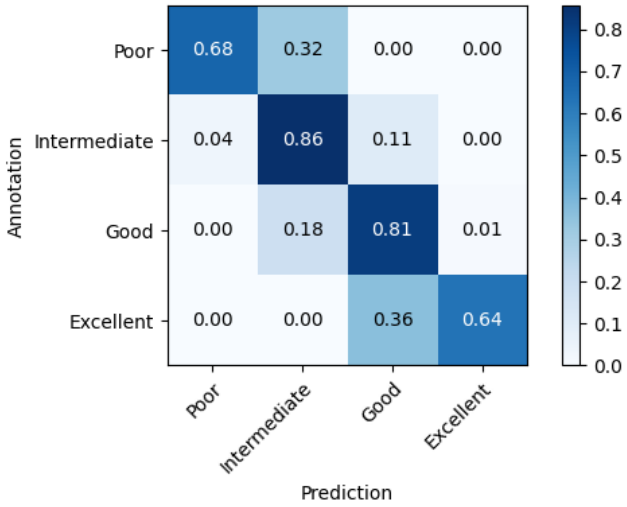


Figura 19: Matriz de confusión en el Experimento 2 de la Arquitectura 4

2.4.3. Experimento 3: Desactivamos el bias de las capas ocultas

En este momento reparamos en que estamos usando el vector *bias* de las capas ocultas, pero no es necesario tenerlo en cuenta porque al estar usando *Batch Normalization* ya estamos incluyendo un parámetro de *offset* β. Vamos a comprobar si mejora el proceso de entrenamiento.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization
400	0.001	256	ReLU	Categorical Crossentropy	ADAM	l2 0.1

Tabla 35: Hiperparámetros para el Experimento 3 de la Arquitectura 4

Y obtenemos los resultados:

Como vemos, el cambio introducido no ha afectado al proceso de entrenamiento, simplemente hemos reducido el tiempo en 30 segundos.

En la matriz de confusión comprobamos que la diagonal ha vuelto a los valores usuales.

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
81.16	75.81	13.84	5.35	422

Tabla 36: Resultados del Experimento 3 de la Arquitectura 4

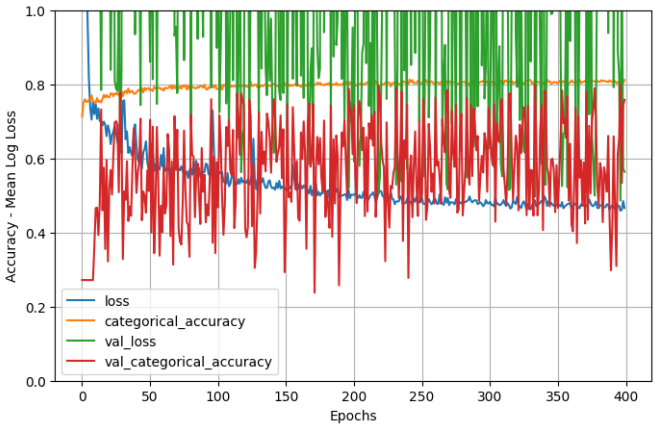


Figura 20: Entrenamiento durante el Experimento 3 de la Arquitectura 4

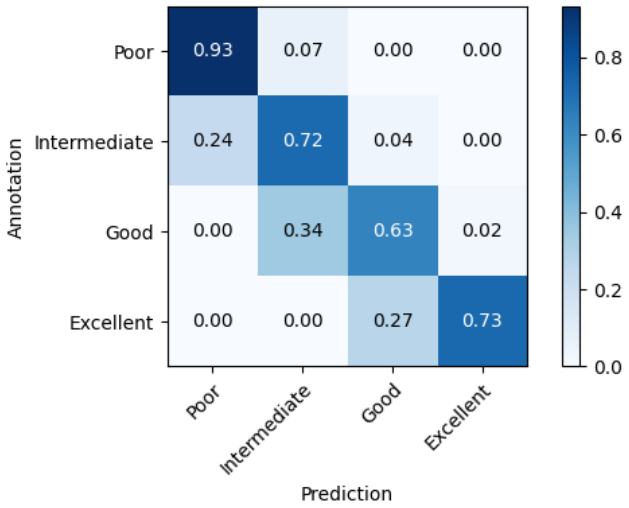


Figura 21: Matriz de confusión en el Experimento 3 de la Arquitectura 4

2.4.4. Experimento 4: Usamos inicializadores

Al estar usando una arquitectura profunda es importante que los pesos no se inicializen con valores demasiado altos ya que estos valores pueden crecer descontroladamente en las últimas capas, así que utilizaremos el initicializador *He Normal*. La configuración queda así:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	256	ReLU	C.C.	ADAM	l2 0.1	**He Normal**

Tabla 37: Hiperparámetros para el Experimento 4 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
81.16	75.81	13.84	5.35	422

Tabla 38: Resultados del Experimento 4 de la Arquitectura 4

Como vemos, el cambio introducido no ha afectado al proceso de entrenamiento, simplemente hemos reducido el tiempo en 30 segundos. Puede que tengamos que cambiar el regularizador.

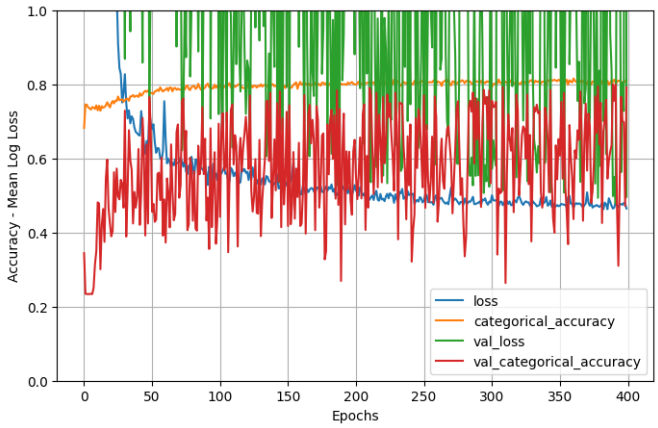


Figura 22: Entrenamiento durante el Experimento 4 de la Arquitectura 4

2.4.5. Experimento 5: Cambiamos el regularizador

Creemos que el problema del entrenamient errático está producido por el regularizador, hemos aumentado mucho la penalización y L2 añade penalización proporcional a la media del cuadrado de los pesos. Pensamos que utilizando L1 y reduciendo λ conseguiremos solucionar nuestro problema.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	256	ReLU	C.C.	ADAM	**l1 0.0001**	He Normal

Tabla 39: Hiperparámetros para el Experimento 5 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
95.86	74.01	-0.86	21.85	455

Tabla 40: Resultados del Experimento 5 de la Arquitectura 4

Relajando la penalización en la regularización hemos vuelto al *overfitting* como se ve en la tabla 40, ya que tenemos un *variance* de 22 %. Lo bueno es que hemos reducido la variación en el entrenamiento como vemos en la figura 23.

2.4.6. Experimento 6: Aumentamos *batch size*

Antes de pasar a la siguiente arquitectura decidimos aumentar el tamaño del *batch size* para probar si así reducimos algo el *overfitting*.

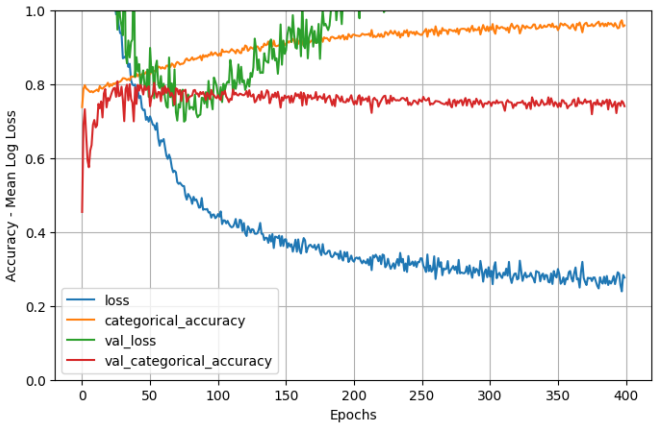


Figura 23: Entrenamiento durante el Experimento 5 de la Arquitectura 4

Epochs	**Learning rate**	**Batch size**	**Activation**	**Loss**	**Optimizer**	**Regularization**	**Initializer**
400	0.001	**1024**	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 41: Hiperparámetros para el Experimento 6 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
99.8	74.88	-4.8	24.92	455

Tabla 42: Resultados del Experimento 6 de la Arquitectura 4

Ha aumentado el *ovefitting*, de hecho el *loss* del conjunto de validación no aparece en el gráfico de evolución durante el entrenamiento.

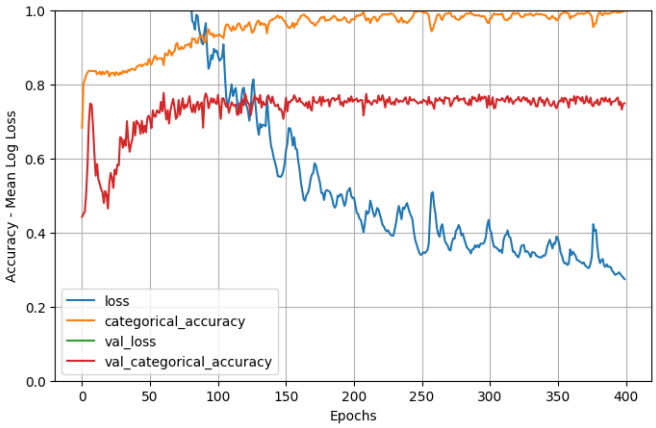


Figura 24: Entrenamiento durante el Experimento 6 de la Arquitectura 4

2.4.7. Experimento 7: Reducimos *batch size*

No estamos satisfechos con el último experimento que además nos sugiere que a mayor *batch size* mayor *bias* por lo que lo reducimos a un tamaño de 32.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	**32**	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 43: Hiperparámetros para el Experimento 7 de la Arquitectura 4

Y obtenemos los resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
78.14	77.79	16.85	0.35	3239

Tabla 44: Resultados del Experimento 7 de la Arquitectura 4

Como vemos, reducir drásticamente el *batch size* ha eliminado por completo el *overfitting* y de hecho el *accuracy* del conjunto de entrenamiento y validación han convergido.

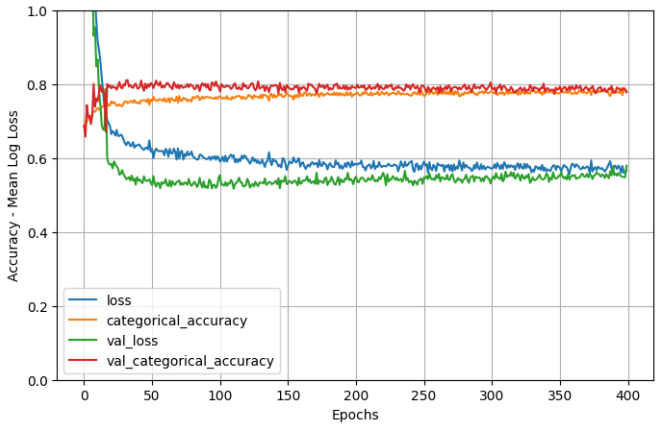


Figura 25: Entrenamiento durante el Experimento 7 de la Arquitectura 4

2.4.8. Conclusiones Arquitectura 4

- Sin regularización el *overfitting* persiste.
- A menor tamaño del *batch size* menor *overfitting*.
- La regularización L2 con un λ grande hace que el entrenamiento sea errático.
- Una arquitectura con mayor profundidad no ha producido una mejora en el *bias*.

2.5. Arquitectura 5: Arquitectura reloj de arena

Aunque no llega a ser un encoder-decoder, decidimos usar una arquitectura algo distinta a las que hemos estado usando hasta el momento. La arquitectura es la siguiente:

1. Capa densa de 512 neuronas con *Batch Normalization*
2. Capa densa de 128 neuronas con *BN*
3. Capa densa de 32 neuronas con *BN*
4. Capa densa de 128 neuronas con *BN*
5. Capa densa de 512 neuronas con *BN*

2.5.1. Experimento 1: Probamos con la misma configuración del experimento 2.4.7

Aquí describir el cambio que hemos hecho con respecto al experimento anterior.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	32	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 45: Hiperparámetros para el Experimento 1 de la Arquitectura 5

Y obtenemos los siguientes resultados:

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.81	76.74	14.19	4.07	3414

Tabla 46: Resultados del Experimento 1 de la Arquitectura 5

Como vemos, el *variance* ha aumentado un 4 % y en cuanto a *accuracy* hemos mejorado un 2 % en el conjunto de entrenamiento aunque empeorado mínimamente en cuanto al de validación. Es interesante el efecto que ha tenido esta arquitectura en el proceso de entrenamiento, los *accuracys* de los dos conjuntos se cruzan siendo el de validación el que comienza con un mejor valor.

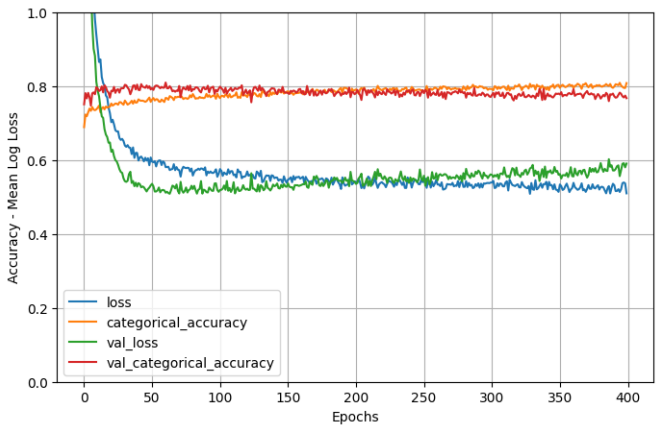


Figura 26: Entrenamiento durante el Experimento 1 de la Arquitectura 5

Esta arquitectura no ha mejorado el *bias* y en la matriz de confusión mostrada en la figura 27 tampoco apreciamos cambios, las clases Ïntermediatez "Good"siguen siendo las clases con peor *recall*.

2.5.2. Experimento 2: Aumentamos *batch size* y *learning rate*

El problema de tener un *batch size* pequeño es que el tiempo de entrenamiento aumenta de forma drástica. En el paper *Understanding Batch Normalization (2018)* de J. Bjorck et al. se menciona que el optimizador está afectado por un ruido acotado superiormente por $\frac{lr}{bs}$, es decir por el *learning rate* dividido por el *batch size*. Esto implica que aumentar el *batch size* tiene el mismo efecto que reducir el *learning rate*. Como actualmente tenemos un buen equilibrio entre estos dos hiperparámetros que no conduce al *overfitting* vamos a aumentar ambos para mantener esta proporción y además reducir el tiempo de entrenamiento.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	**0.01**	**256**	ReLU	C.C.	ADAM	l1 0.0001	He Normal

Tabla 47: Hiperparámetros para el Experimento 2 de la Arquitectura 5

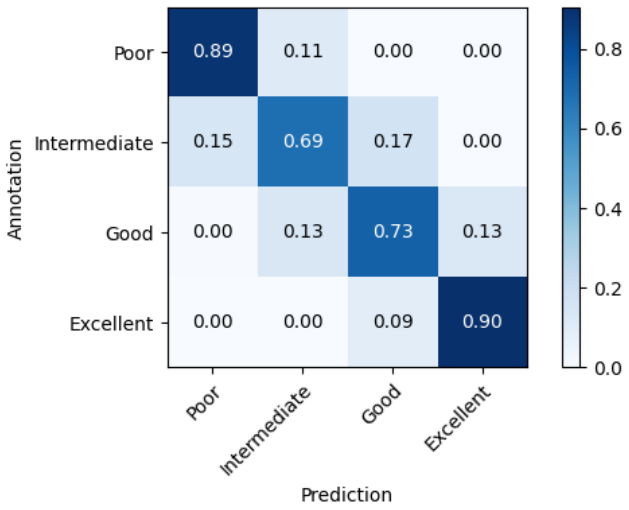


Figura 27: Matriz de confusión en el Experimento y de la Arquitectura x

Y obtenemos los siguientes resultados:

Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
84.43	80.21	10.57	4.22	379

Tabla 48: Resultados del Experimento 2 de la Arquitectura 5

El efecto que hemos conseguido con este cambio ha sido muy satisfactorio, el tiempo de entrenamiento se ha reducido considerablemente y además hemos obtenido un mejor *accuracy*. La pega es que con más *epochs* llegaríamos a un *overfitting* como se aprecia en la figura 28.

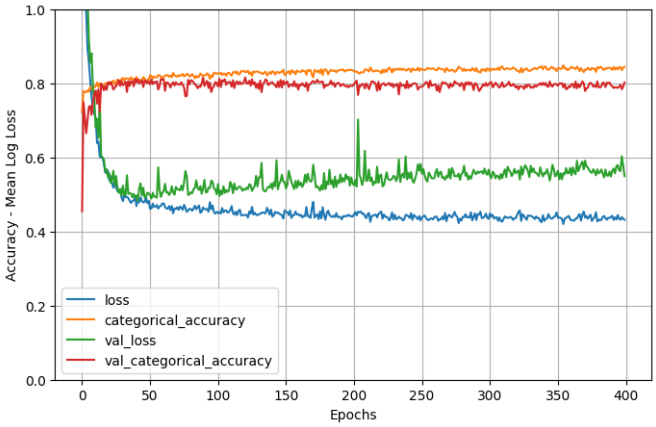


Figura 28: Entrenamiento durante el Experimento 2 de la Arquitectura 5

2.5.3. Experimento 3: Aumentamos *batch size* y *learning rate*

Para evitar el problema del *overfitting* a largo plazo que hemos detectado en el experimento 2.5.2 introducimos una técnica que hasta ahora no habíamos tenido en cuenta, el *dropout*.

Y obtenemos los siguientes resultados:

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer	Dropout
400	0.01	256	ReLU	C.C.	ADAM	l1 0.0001	He Normal	**0.1**

Tabla 49: Hiperparámetros para el Experimento 3 de la Arquitectura 5

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.13	75.37	14.87	4.76	382

Tabla 50: Resultados del Experimento 3 de la Arquitectura 5

Hemos empeorado el *bias* aunque hemos mantenido el *variance* y como se ve en la figura 29 el entrenamiento se ha vuelto más errático aunque la divergencia que veíamos entre los *accuracy* de los conjuntos de entrenamiento y validación han desaparecido.

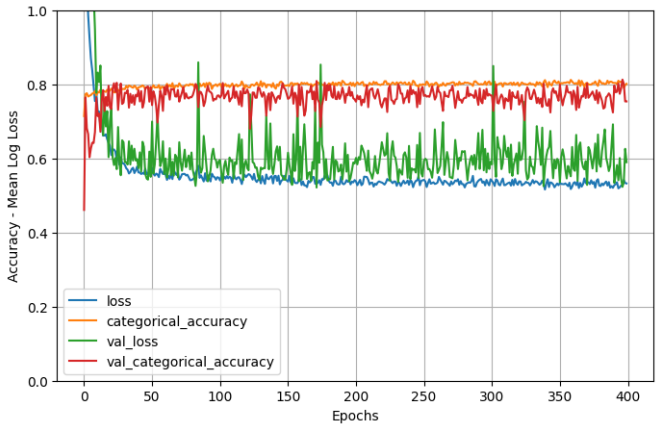


Figura 29: Entrenamiento durante el Experimento 3 de la Arquitectura 5

2.5.4. Experimento 4: Cambiamos la función de activación

Otra opción que no hemos probado hasta ahora es usar la función de activación *eLU*, así que en este experimento la usaremos con la esperanza de mejorar el *bias* del modelo.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer	Dropout
400	0.01	256	**eLU**	C.C.	ADAM	l1 0.0001	He Normal	0.1

Tabla 51: Hiperparámetros para el Experimento 4 de la Arquitectura 5

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
80.25	77.67	14.75	2.58	388

Tabla 52: Resultados del Experimento 4 de la Arquitectura 5

No hemos conseguido reducir el *bias* pero si el *variance* un 2 %. Como consecuencia de usar esta función de activación, el proceso de entrenamiento se ha visto afectado, la variación tanto en el *loss* como en el *accuracy* del conjunto de validación tiene una variación muy grande.

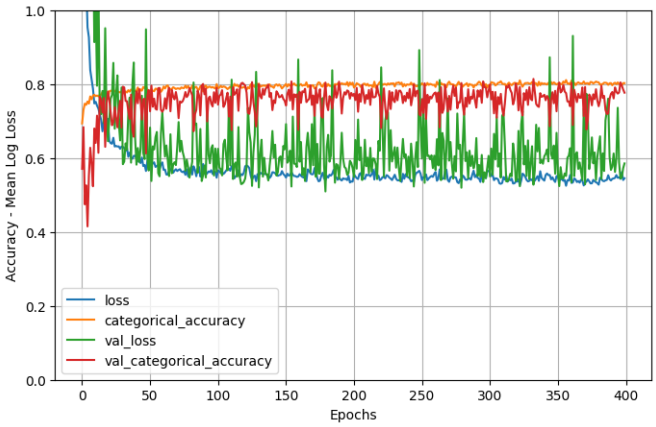


Figura 30: Entrenamiento durante el Experimento 4 de la Arquitectura 5

2.5.5. Conclusiones Arquitectura 5

- El *dropout* consigue mitigar el *overfitting*.
- La función de activación *eLU* consigue un menor *variance* en esta arquitectura.

3. Fase 2: Optimización sistemática

En esta fase iremos realizando cambios atómicos, es decir, para cada experimento nos centraremos en un hiperparámetro a optimizar, quedándonos con el modelo que tenga mejor *accuracy* en el conjunto de validación. Todos los experimentos a continuación han sido realizados 10 veces.

3.1. Arquitectura 1: MLP4

Probaremos una arquitectura de complejidad media para buscar el overfitting y aplicaremos técnicas de regularización para reducir la varianza.

- Capa densa de 256 con Batch Normalization y Dropout = 0
- Capa densa de 124 con BN y Dropout = 0
- Capa densa de 64 con BN y Dropout = 0
- Capa densa de 32 con BN y Dropout = 0

3.1.1. Experimento 1: Primera configuración

Como se puede observar en la figura 31, tenemos un overfitting bastante grande y debemos aplicar técnicas de regularización para mejorar la generalización de la red.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
400	0.001	64	ReLU	C.C.	ADAM	No	0

Tabla 53: Hiperparámetros para el Experimento 1 de la Arquitectura 6

3.1.2. Experimento 2: Añadimos Dropout 0.2

Se añade un Dropout de 0.2 para reducir la varianza y mejorar la generalización. La configuración es la siguiente:

Los resultados obtenidos son los siguientes: Se consigue una reducción de la varianza a cambio de un empeoramiento del Bias.

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	98.7	70.47	-3.7	28.23	817
Std	0.83	0.28	0.83	0.86	8.4

Tabla 54: Resultados del Experimento 1 de la Arquitectura 6

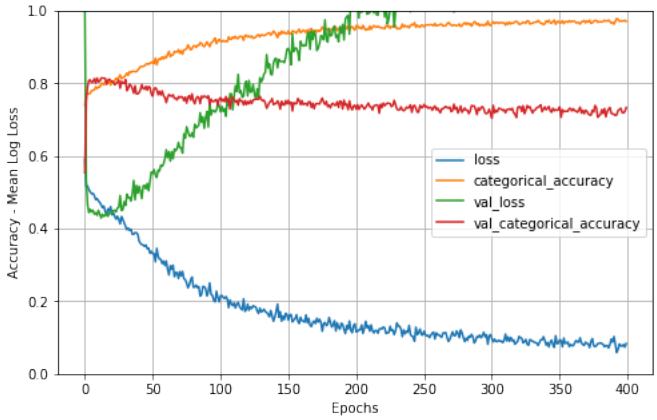


Figura 31: Entrenamiento durante el Experimento 1 de la Arquitectura 6

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
400	0.001	64	ReLU	C.C.	ADAM	No	0.2

Tabla 55: Hiperparámetros para el Experimento 1 de la Arquitectura 6

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Mean	88.28	76.67	6.72	11.61	743
Std	1.15	0.79	1.15	1.89	8.9

Tabla 56: Resultados del Experimento 2 de la Arquitectura 6

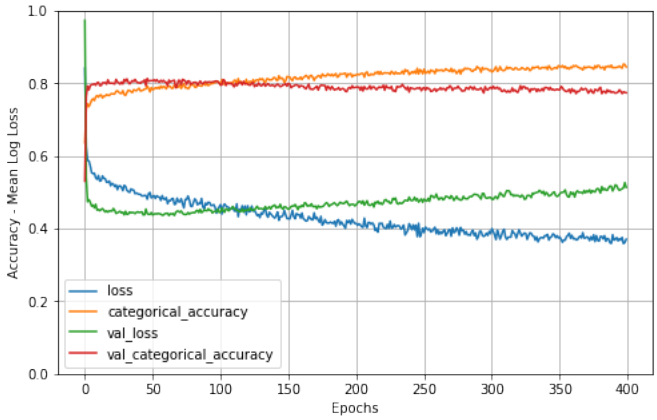


Figura 32: Entrenamiento durante el Experimento 2 de la Arquitectura 6

3.1.3. Experimento 3: Regularizacón L1, L2 y L1-L2

Seguimos buscando la reducción de la varianza. Para ello probaremos los regularizadores L1, L2 y L1-L2. Como se puede observar la regularizacón L1 ha mejorado drasticamente la varianza y se ha conseguido una

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
400	0.001	64	ReLU	C.C.	ADAM	?	0.2

Tabla 57: Hiperparámetros para el Experimento 3 de la Arquitectura 6

	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
L1 0.1	63.6	68.98	31.4	-5.38	765
L2 0.1	73.06	76.86	21.94	-3.8	864
L1-L2 0.1	63.7	72.02	31.3	-8.32	804

Tabla 58: Resultados del Experimento 3 de la Arquitectura 6

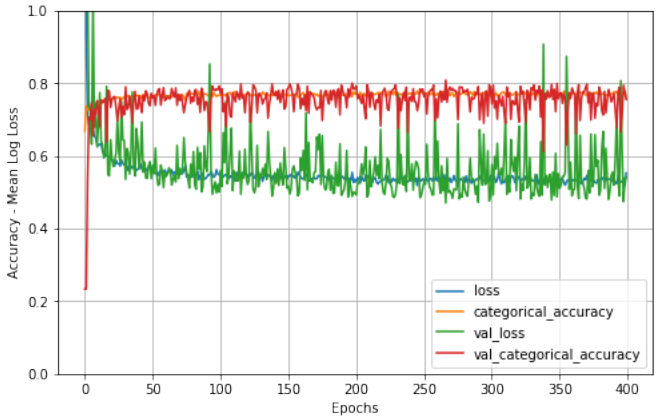


Figura 33: Entrenamiento durante el Experimento 3 (L2) de la Arquitectura 6

configuración que generaliza mejor. Sin embargo se ha aumentado demasiado el Bias, es por ello que en los siguientes experimentos habrá que intentar reducir el Bias.

La regularizacón L2 tambien ha conseguido reducir considerablemente la varianza, pero tiene la ventaja de que hemos conseguido menor empeoramiento que con L1. Se concluuye que en este caso es comveniente usar L2 frente a L1. En los siguientes experimentos habrá que intentar reducir el Bias, las soluciones partirían de una reducción del tamaño del Batch hasta una aumento de epochs.

La regularización L1-L2 también mejora drásticamente la varianza, pero empeora en exceso el Bias.

Se concluye que el mejor regularizador para nuestra configuración actual es el L1.

3.1.4. Experimento 4: Aumentar Epochs

En el experimento anterior se ha conseguido una mejora de la generalización de la red a cambio de un alto Bias, es por ello que en este experimento se busca la reducción del Bias utilizando más epochs de entrenamiento.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
?	0.001	64	ReLU	C.C.	ADAM	L2 0.1	0.2

Tabla 59: Hiperparámetros para el Experimento 3 de la Arquitectura 6

Duplicando o triplicando el número de epochs no se está logrando una mejora significativa de la varianza. Esto puede ser debido a que la red se está quedando atrapada en un mínimo local y hay que aplicar más reconfiguraciones para solventar este problema.

3.2. Experimento 5: Optimizadores

Se van a probar los diferentes optimizadores con los hiper parámetros por defecto para buscar una mejor aproximación e intentar salir en el mínimo local en que se encuentra la red.

epochs	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
400	73.06	76.86	21.94	-3.8	864
800	73.52	77.98	21.48	-4.46	1585
1200	72.64	75.37	22.36	-2.73	1944

Tabla 60: Resultados del Experimento 4 de la Arquitectura 6

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
800	0.001	64	ReLU	C.C.	?	L2 0.1	0.2

Tabla 61: Hiperparámetros para el Experimento 5 de la Arquitectura 6

Batch Size	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Adam	73.52	77.98	21.48	-4.46	1585
RMSprop	71.39	79.53	23.61	-8.14	865
SGD	77.42	78.35	17.58	-0.93	683

Tabla 62: Resultados del Experimento 5 de la Arquitectura 6

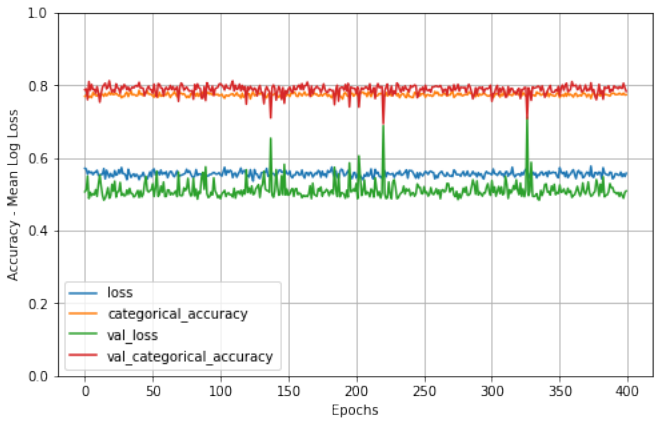


Figura 34: Entrenamiento durante el Experimento 5 (SGD) de la Arquitectura 6

Como se puede observar con el optimizador SGD se consigue una liguera mejora en los resultados. Sin embargo, creemos que se puede conseguir mejores resultados ya que el Bias sigue siendo demasiado alto y la curva de aprendizaje es plana, lo que nos puede indicar que seguimos atrapados en un mínimo local.

3.2.1. Experimento 6: Variando el Batch size

Se sigue buscando una reducción del Bias, para ello en este experimento se probarán diferentes tamaños de Batch para comprobar si se consigue una mejora.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
800	0.001	?	ReLU	C.C.	SGD	L2 0.1	0.2

Tabla 63: Hiperparámetros para el Experimento 6 de la Arquitectura 6

El tamaño del batch óptimo para la configuración actual es 1024. Obtenemos el mejor validation accuracy y Bias de los últimos. Se concluye que al aumentar el tamaño del batch se consigue explorar más y salir del mínimo local en el que nos encontrábamos antes.

Optimizer	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
64	77.42	78.35	17.58	-0.93	683
32	73.9	79.16	21.1	-5.26	863
128	81.24	79.53	13.76	1.71	443
256	84.74	78.47	10.26	6.27	383
1024	84.33	80.65	10.67	3.68	263
2048	75.45	79.34	19.55	-3.89	323

Tabla 64: Resultados del Experimento 6 de la Arquitectura 6

3.2.2. Experimento 7: Usando los nuevos datos

En este experimento se probará la mejor configuración con el nuevo dataset. Se observa que los resultados mejoran ligeramente, con lo cual se probará a aumentar los epochs para comprobar si mejora el rendimiento.

Epochs	L.R	Batch size	Activation	Loss	Optimizer	Regularization	Dropout
800	1024	64	ReLU	C.C.	SGD	L2 0.1	0.2

Tabla 65: Hiperparámetros para el Experimento 7 de la Arquitectura 6

Epochs	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
800	81.51	82.45	13.49	-0.94	383
1600	85.88	85.85	9.12	0.03	785
3000	88.61	85.72	6.39	2.89	1643

Tabla 66: Resultados del Experimento 7 de la Arquitectura 6

El mejor modelo se obtiene con 1600 epochs, a partir de esos epochs sufrimos estancamiento en el validation accuracy, aunque se ha logrado una ligera mejora en el train accuracy.

4. Procesamiento de datos

En esta sección comprobaremos cómo afecta en distintas arquitecturas el procesamiento de datos

4.1. Data imputation

Usando el notebook base de *PreparingFootballPlayerDataset* lo modificamos para seleccionar distintas *features* e imputar datos mediante **regresión lineal** para el campo de *Release Clause* e *imputación de medias* para 48 jugadores con una puntuación global de 82.

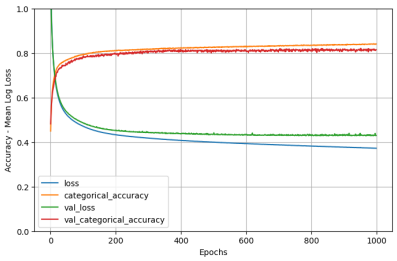
4.2. Comparación de Arquitectura 2.1 con y sin data imputation

Usaremos la configuración del experimento 2.1.2 y comparamos resultados:

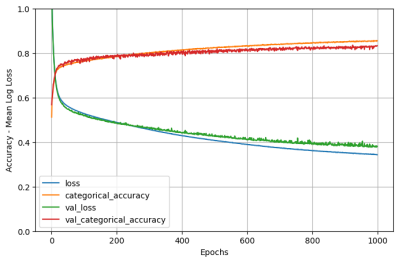
	Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
Sin DI	84.02	81.5	10.97	2.47	199
Con DI	85.52	83.31	9.48	2.21	180

Tabla 67: Comparación de resultados del Experimento 2 de la Arquitectura 1

La figura 35 nos muestra la diferencia entre usar distintos datos en el entrenamiento. Con *data imputation* obtenemos un entrenamiento en el que la red aprende de manera más constante, además obtenemos mejores valores en todas las métricas.



(a) Entrenamiento sin *data imputation*



(b) Entrenamiento con *data imputation*

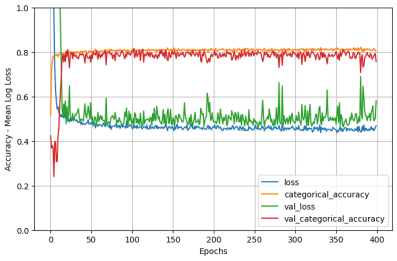
Figura 35: Comparación del entrenamiento

4.3. Comparación de Arquitectura 2.2 con y sin data imputation

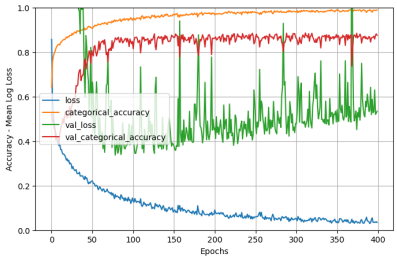
Usaremos la configuración del experimento 2.3.4 y comparamos resultados: La figura 36 nos muestra la

	Train accuracy (%)	**Validation accuracy (%)**	**Bias (%)**	**Variance (%)**	**Training time (s)**
Sin DI	80.92	77.03	14.08	4.07	258
Con DI	98.86	87.52	-3.86	11.34	272

Tabla 68: Comparación de resultados del Experimento 4 de la Arquitectura 3



(a) Entrenamiento sin *data imputation*



(b) Entrenamiento con *data imputation*

Figura 36: Comparación del entrenamiento

diferencia entre usar distintos datos en el entrenamiento. Con *data imputation* obtenemos *overfitting*, aún así, el porcentaje de *accuracy* en validación es el mayor obtenido hasta el momento y podemos asegurar que el procesamiento de los datos tiene un gran peso a la hora de mejorar resultados.