

# Práctica 1: FIFA Players Classification

Daniel Carmona Pedrajas

## 1 Arquitectura 1: Feed Forward Neural Network

La primera arquitectura que usaremos es tan simple como:

1. Capa densa con 512 neuronas.

### 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

Table 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

Table 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*, ...

### 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

Table 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

Table 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

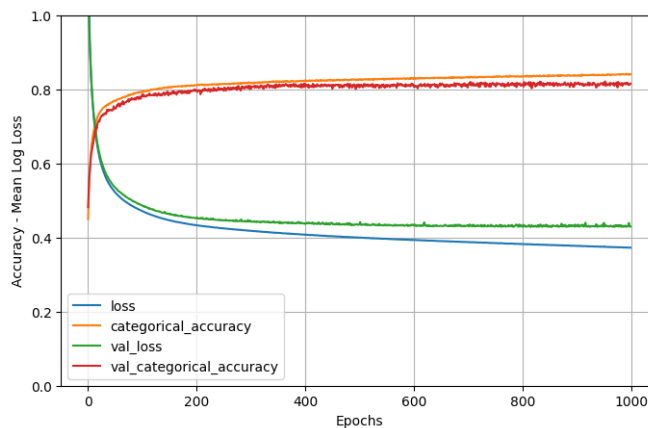


Figure 1: Entrenamiento durante el Experimento 2 de la Arquitectura 1

aunque sí para el conjunto de entrenamiento lo que nos indica que un número de *epochs* tan elevado como el que 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*.

### 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

Table 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

Table 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.



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

#### 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:

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

Table 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

Table 8: Resultados del Experimento 4 de la Arquitectura 1

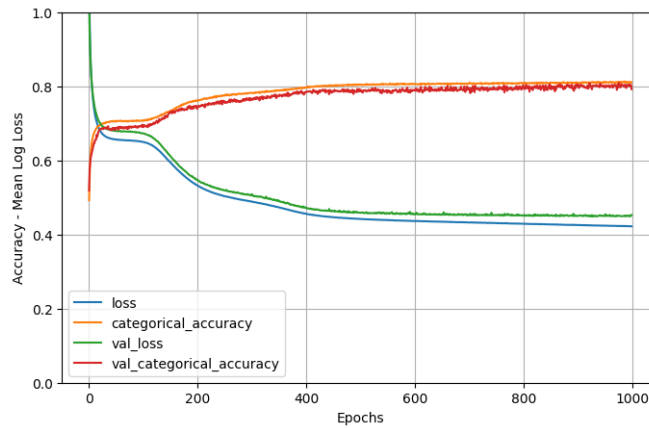


Figure 3: Entrenamiento durante el Experimento 4 de la Arquitectura 1

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.

## 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

Table 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

Table 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.

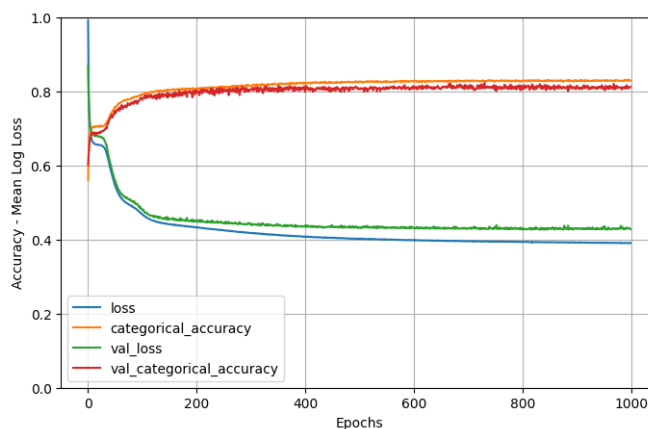


Figure 4: Entrenamiento durante el Experimento 5 de la Arquitectura 1

## 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 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

### 2.1 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

Table 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

Table 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.

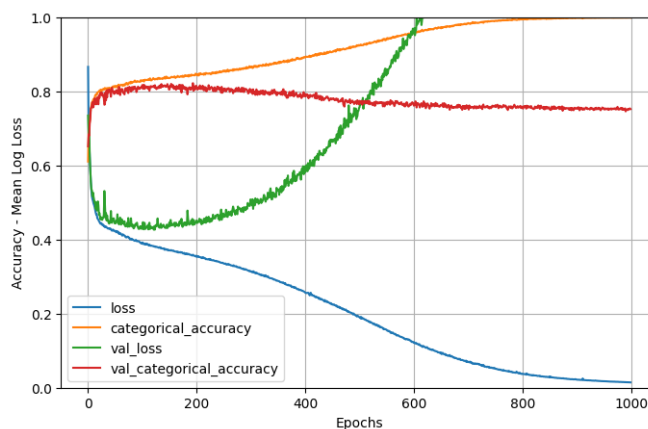


Figure 5: Entrenamiento durante el Experimento 1 de la Arquitectura 2

## 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	12 0.001

Table 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

Table 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.

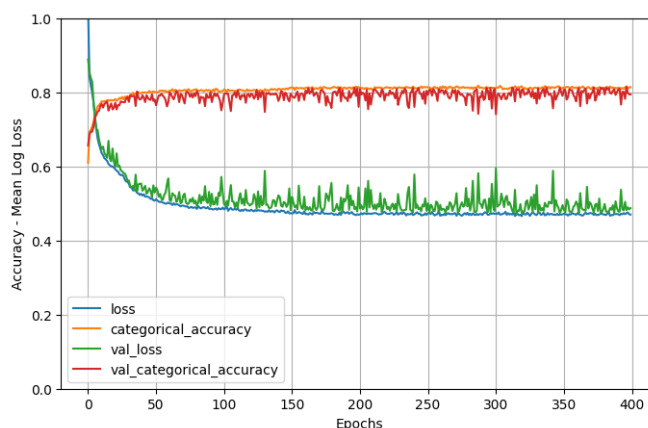


Figure 6: Entrenamiento durante el Experimento 2 de la Arquitectura 2

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.

## 2.3 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	12 0.001

Table 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

Table 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.

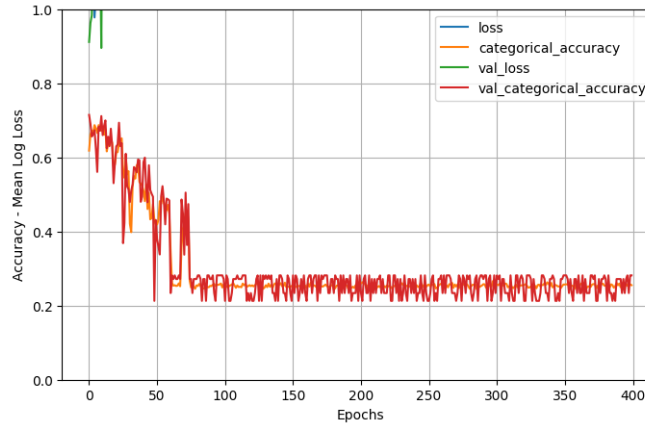


Figure 7: Entrenamiento durante el Experimento 4 de la Arquitectura 2

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

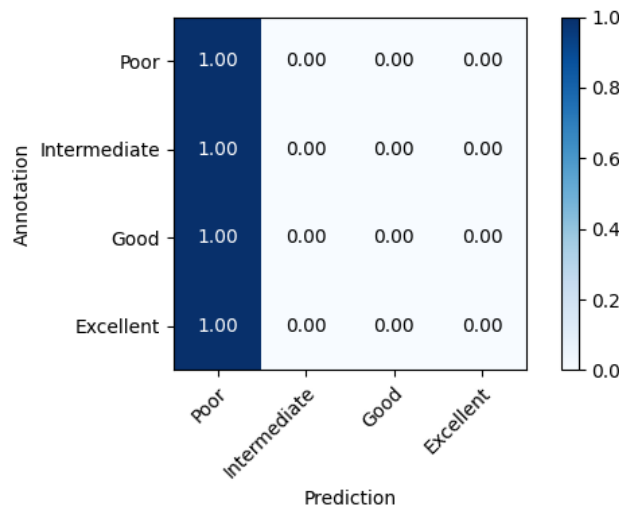


Figure 8: Matriz de confusión en el Experimento 3 de la Arquitectura 2

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

## 2.4 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 ( $\beta_1=0.9$  y  $\beta_2=0.999$ ) 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	<b>0.001</b>	128	tanh	Categorical Crossentropy	ADAM	12 0.001

Table 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

Table 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 figura 9 vemos que 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*.

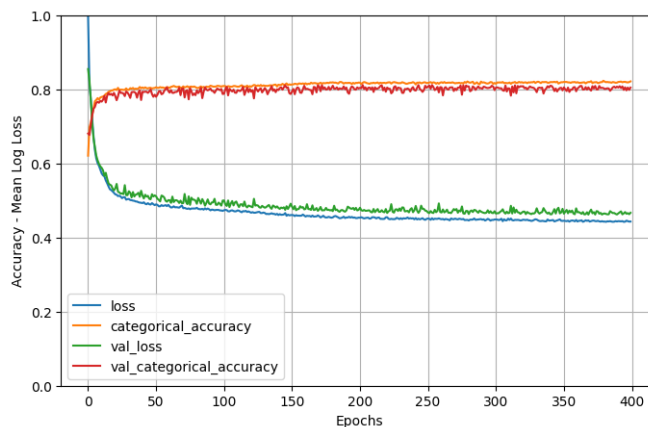


Figure 9: Entrenamiento durante el Experimento 4 de la Arquitectura 2

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



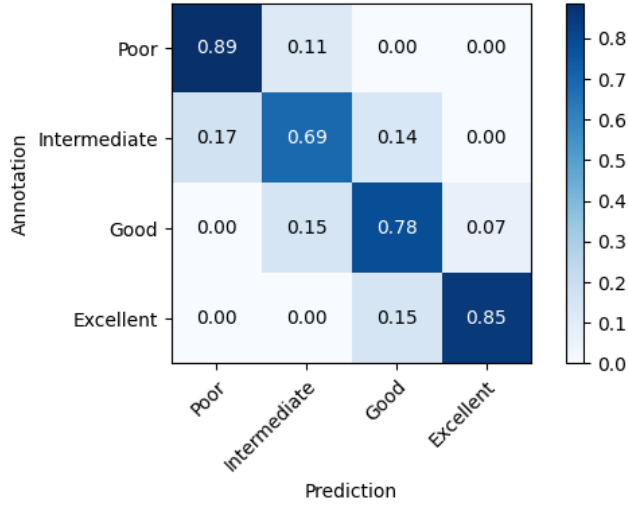


Figure 10: Matriz de confusión en el Experimento 4 de la Arquitectura 2

## 2.5 Experimento 5: Reducimos *batch-size*

Decidimos reducir el *batch size* para intentar reducir el *bias* como en el experimento 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	12 0.001

Table 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

Table 20: Resultados del Experimento 5 de la Arquitectura 2

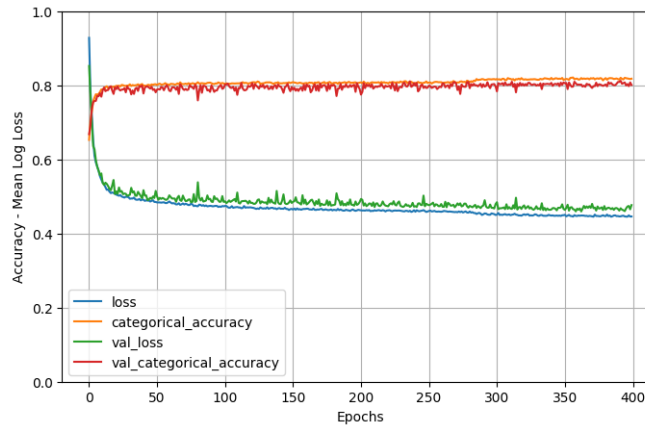


Figure 11: Entrenamiento durante el Experimento 5 de la Arquitectura 2

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.

## 2.6 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.

## 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.

### 3.1 Experimento 1: Probamos la configuración del experimento 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	12 0.001

Table 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

Table 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.

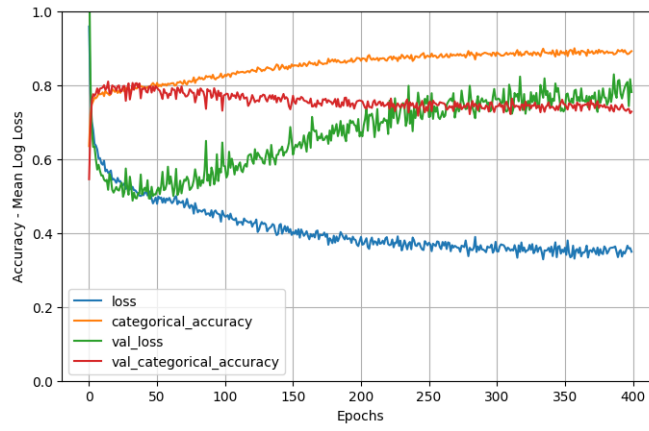


Figure 12: Entrenamiento durante el Experimento 1 de la Arquitectura 3

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.

### 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	12 0.001

Table 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

Table 24: Resultados del Experimento 2 de la Arquitectura 3

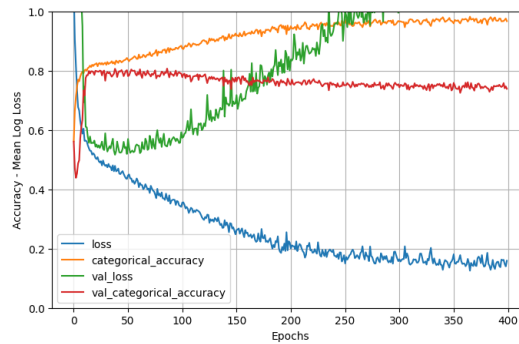


Figure 13: Entrenamiento durante el Experimento 2 de la Arquitectura 3

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.

### 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	12 0.1

Table 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

Table 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

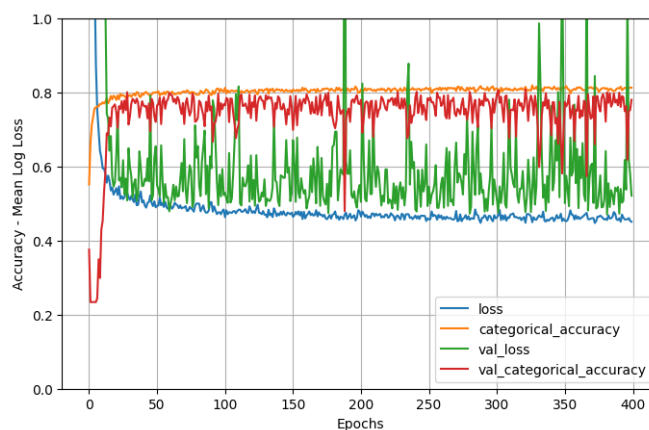


Figure 14: Entrenamiento durante el Experimento 3 de la Arquitectura 3

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

### 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	12 0.1

Table 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)
<b>Mean</b>	80.92	77.03	14.08	4.07	258.4
<b>Std</b>	0.46	1.8	0.46	1.56	7.46

Table 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.

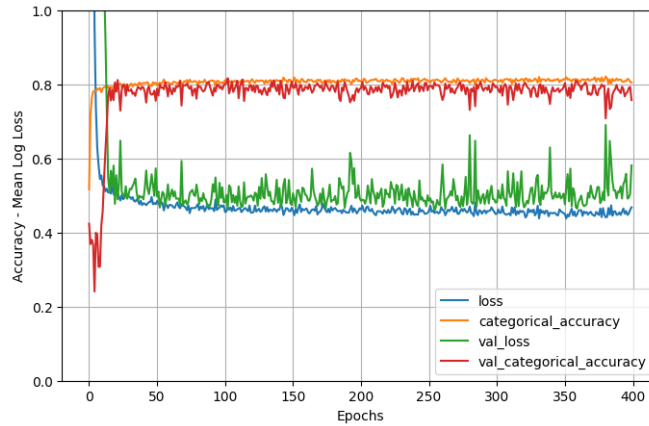


Figure 15: Entrenamiento durante el Experimento 4 de la Arquitectura 3

### 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

Table 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)
<b>Mean</b>	97.31	74.55	-2.31	22.76	248
<b>Std</b>	0.44	0.99	0.44	0.68	6.12

Table 30: Resultados del Experimento 5 de la Arquitectura 3

Definitivamente la regularización es necesaria para evitar el *overfitting*

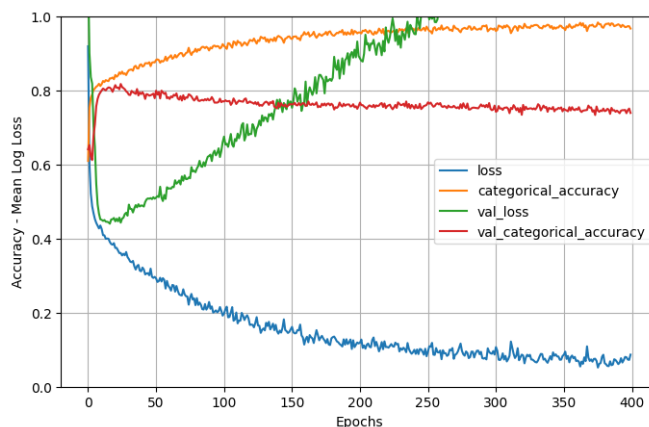


Figure 16: Entrenamiento durante el Experimento 5 de la Arquitectura 3

### 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*.

## 4 Arquitectura 4: Añadimos profundidad

Hasta el momento el máximo *accuracy* en validación que hemos conseguido ha sido en el experimento 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.

## 4.1 Experimento 1: Repetimos la última configuración

Vamos a usar la configuración del experimento 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

Table 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

Table 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 3.5.

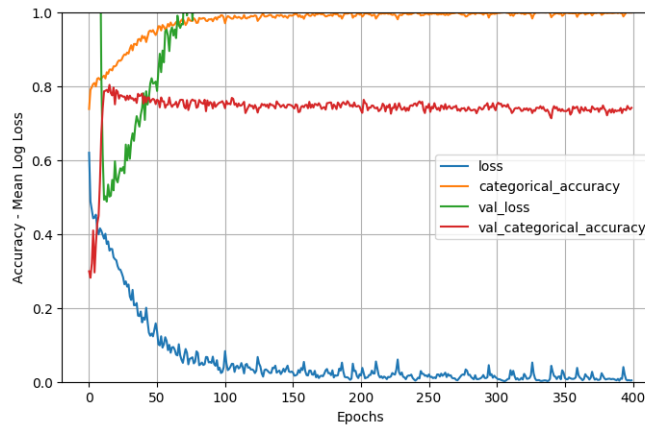


Figure 17: Entrenamiento durante el Experimento 1 de la Arquitectura 4

## 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	12 0.1

Table 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

Table 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.

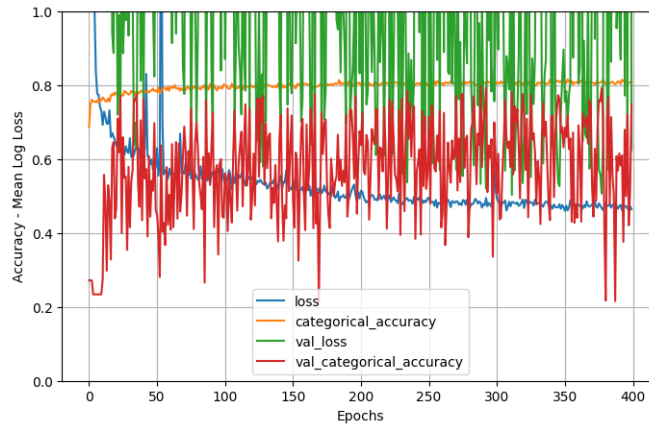


Figure 18: Entrenamiento durante el Experimento 2 de la Arquitectura 4

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

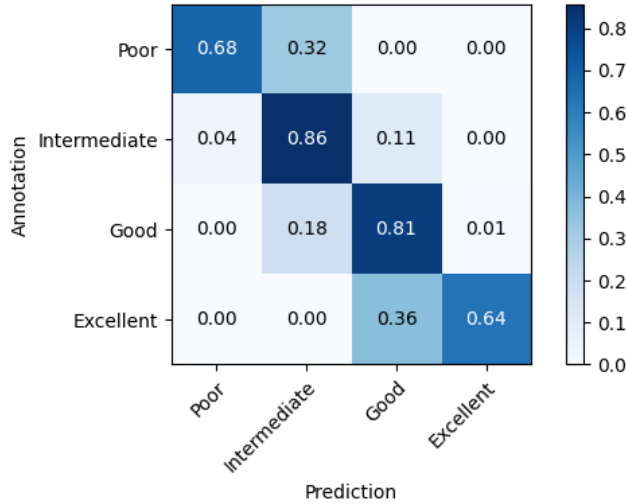


Figure 19: Matriz de confusión en el Experimento 2 de la Arquitectura 4

### 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*  $\beta$ . 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	12 0.1

Table 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

Table 36: Resultados del Experimento 3 de la Arquitectura 4

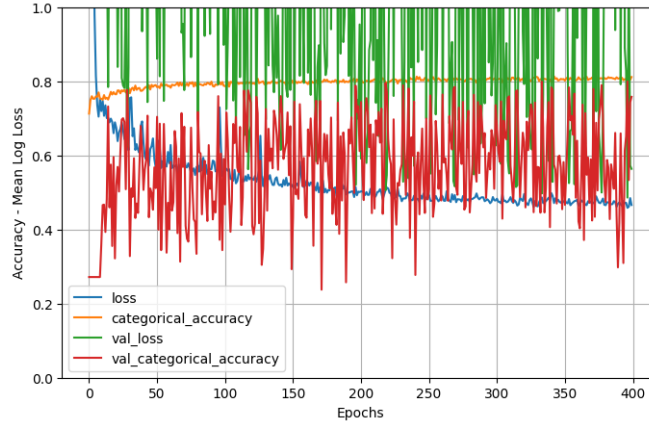


Figure 20: Entrenamiento durante el Experimento 3 de la Arquitectura 4

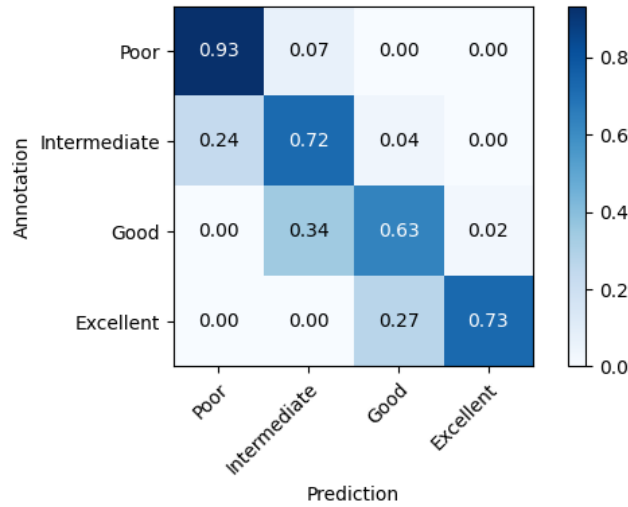


Figure 21: Matriz de confusión en el Experimento 3 de la Arquitectura 4

#### 4.4 Experimento 4: Usamos inicializadores

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

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	256	ReLU	C.C.	ADAM	12 0.1	He Normal

Table 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

Table 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.

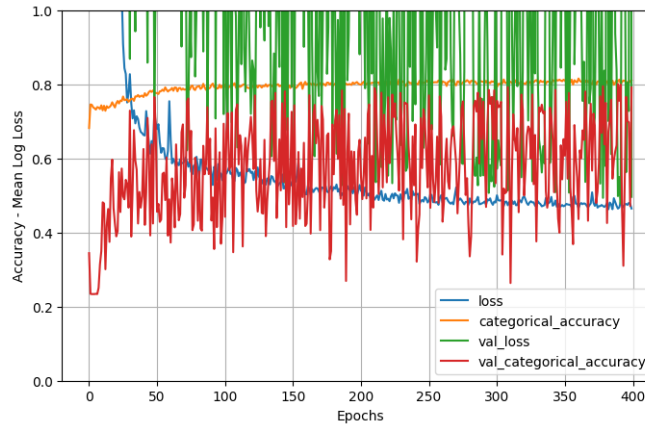


Figure 22: Entrenamiento durante el Experimento 4 de la Arquitectura 4

## 4.5 Experimento 5: Cambiamos el regularizador

Creemos que el problema del entrenamiento 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  $\lambda$  conseguiremos solucionar nuestro problema.

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

Table 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

Table 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.

## 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*.

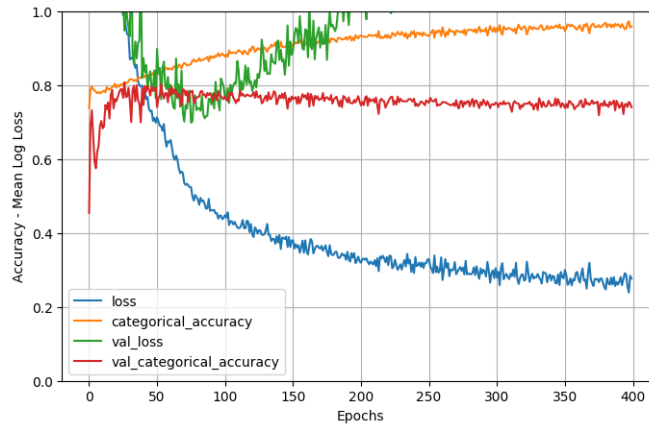


Figure 23: Entrenamiento durante el Experimento 5 de la Arquitectura 4

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.001	1024	ReLU	C.C.	ADAM	11 0.0001	He Normal

Table 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

Table 42: Resultados del Experimento 6 de la Arquitectura 4

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

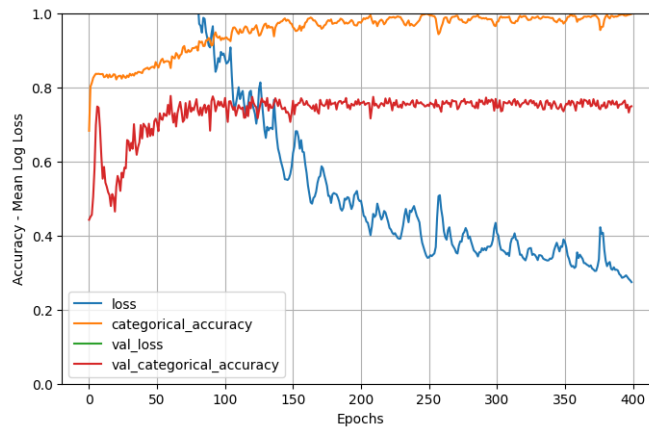


Figure 24: Entrenamiento durante el Experimento 6 de la Arquitectura 4

## 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	11 0.0001	He Normal

Table 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

Table 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.

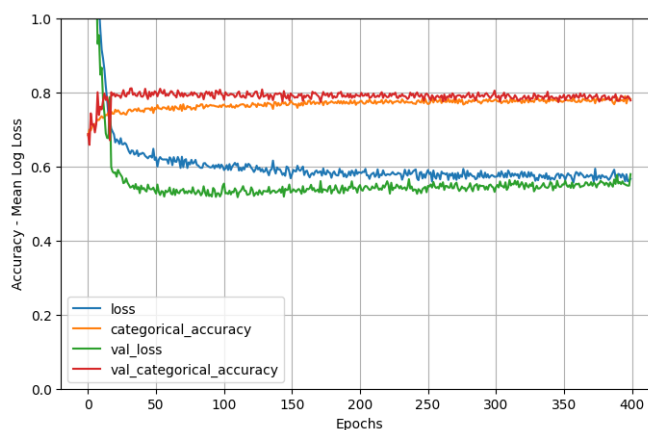


Figure 25: Entrenamiento durante el Experimento 7 de la Arquitectura 4

## 4.8 Conclusiones Arquitectura 4

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

## 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*

## 5.1 Experimento 1: Probamos con la misma configuración del experimento 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	11 0.0001	He Normal

Table 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

Table 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.

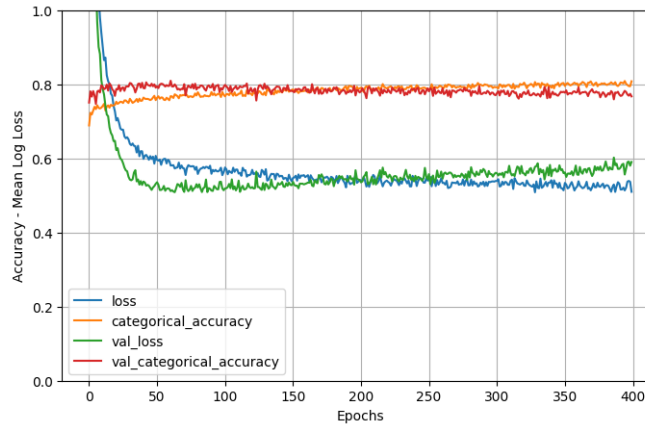


Figure 26: Entrenamiento durante el Experimento 1 de la Arquitectura 5

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

## 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.

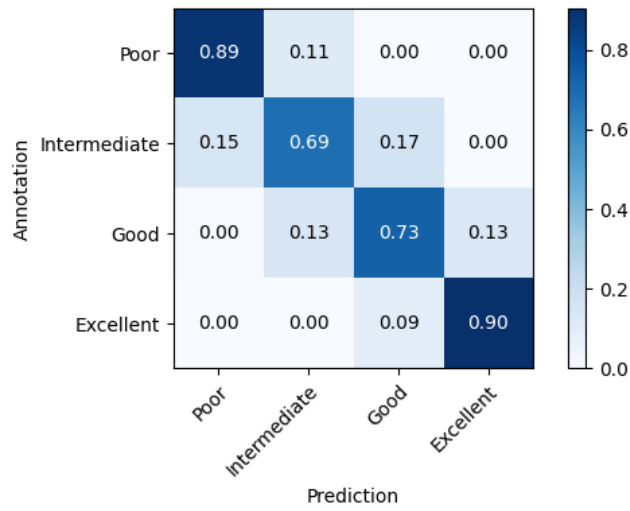


Figure 27: Matriz de confusión en el Experimento y de la Arquitectura x

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer
400	0.01	256	ReLU	C.C.	ADAM	11 0.0001	He Normal

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

Y obtenemos los siguientes resultados:

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

Table 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.

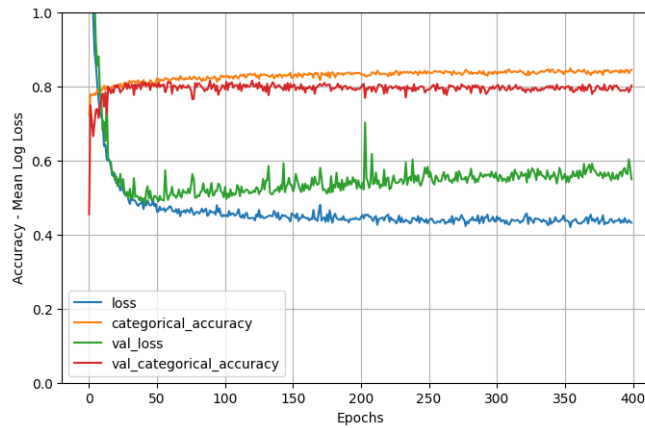


Figure 28: Entrenamiento durante el Experimento 2 de la Arquitectura 5

### 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 5.2 introducimos una técnica que hasta ahora no habíamos tenido en cuenta, el *dropout*.

Epochs	Learning rate	Batch size	Activation	Loss	Optimizer	Regularization	Initializer	Dropout
400	0.01	256	ReLU	C.C.	ADAM	11 0.0001	He Normal	<b>0.1</b>

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

Y obtenemos los siguientes resultados:

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

Table 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.

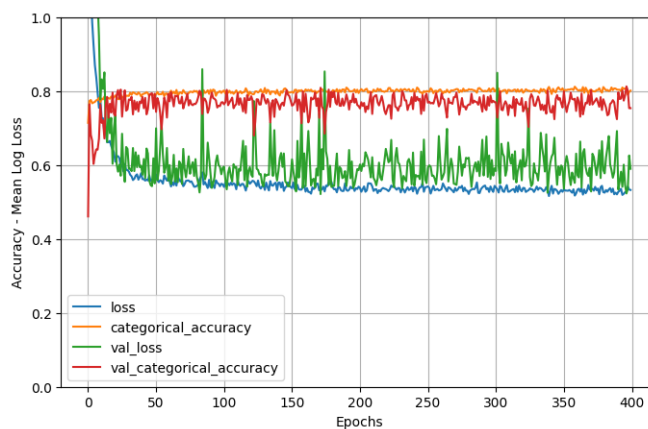


Figure 29: Entrenamiento durante el Experimento 3 de la Arquitectura 5

### 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	<b>eLU</b>	C.C.	ADAM	11 0.0001	He Normal	0.1

Table 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

Table 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.

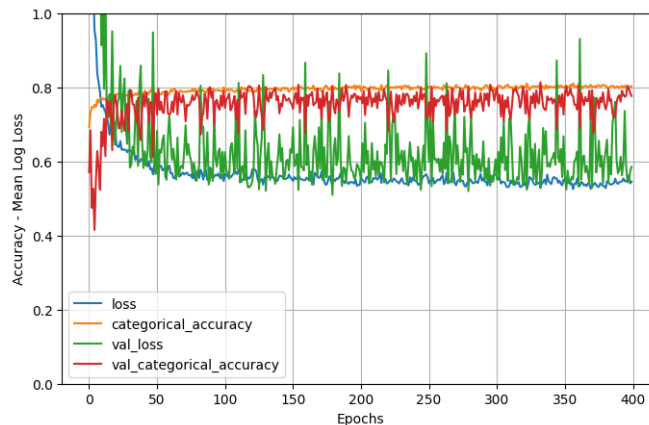


Figure 30: Entrenamiento durante el Experimento 4 de la Arquitectura 5

## 5.5 Conclusiones Arquitectura 5

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

## 6 Arquitectura 6: Solucionando el overfitting en 256-124-64-32

Probaremos una arquitectura de complejidad media para buscar un 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

### 6.1 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	Initializer
400	0.001	64	ReLU	C.C.	ADAM	No	0	He Normal

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

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
97.05	73.33	-2.05	23.72	462

Table 54: Resultados del Experimento 1 de la Arquitectura 6



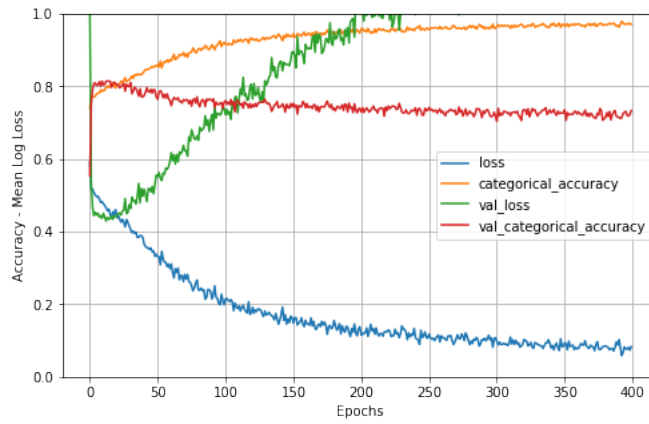


Figure 31: Entrenamiento durante el Experimento 1 de la Arquitectura 6

## 6.2 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:

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

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

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

Train accuracy (%)	Validation accuracy (%)	Bias (%)	Variance (%)	Training time (s)
84.52	77.42	10.48	7.1	503

Table 56: Resultados del Experimento 2 de la Arquitectura 6

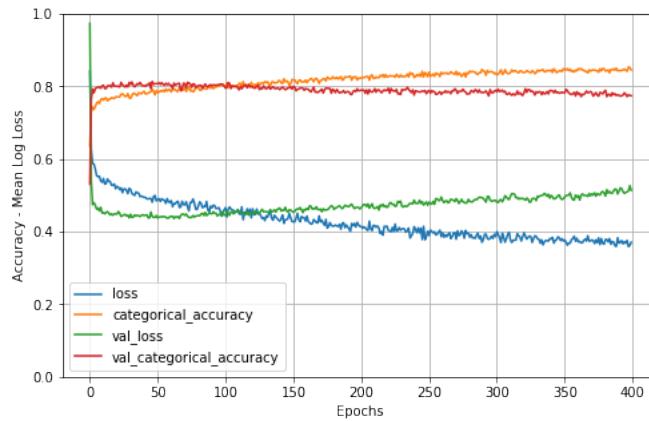


Figure 32: Entrenamiento durante el Experimento 2 de la Arquitectura 6

empeoramiento del Bias.

### 6.3 Experimento 3: Aumentar Epochs

## 7 Procesamiento de datos

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

### 7.1 Data imputation

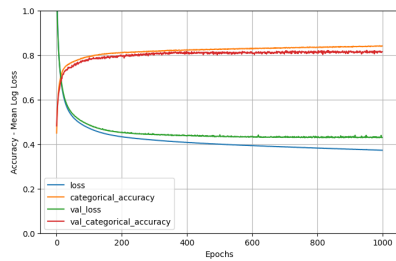
Usando el notebook base de *PreparingFootballPlayerDataset* lo modificamos para seleccionar distintas *features* e imputar datos mediante **regresión lineal** para el campo de *Release Clause* e *imputación de medias* para 48 jugadores con una puntuación global de 82.

### 7.2 Comparación de Arquitectura 1 con y sin data imputation

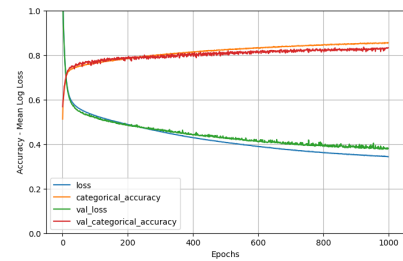
Usaremos la configuración del experimento 1.2 y comparamos resultados:

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

Table 57: Comparación de resultados del Experimento 2 de la Arquitectura 1



(a) Entrenamiento sin *data imputation*



(b) Entrenamiento con *data imputation*

Figure 33: Comparación del entrenamiento

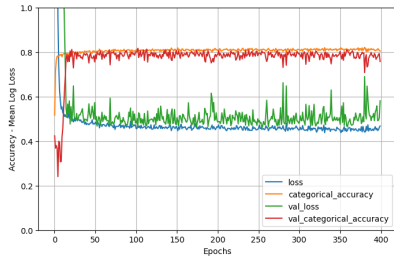
La figura 33 nos muestra la diferencia entre usar distintos datos en el entrenamiento. Con *data imputation* obtenemos un entrenamiento en el que la red aprende de manera más constante, además obtenemos mejores valores en todas las métricas.

### 7.3 Comparación de Arquitectura 1 con y sin data imputation

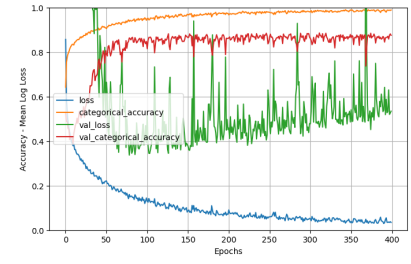
Usaremos la configuración del experimento 3.4 y comparamos resultados:

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

Table 58: Comparación de resultados del Experimento 4 de la Arquitectura 3



(a) Entrenamiento sin *data imputation*



(b) Entrenamiento con *data imputation*

Figure 34: Comparación del entrenamiento

La figura 34 nos muestra la 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.