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

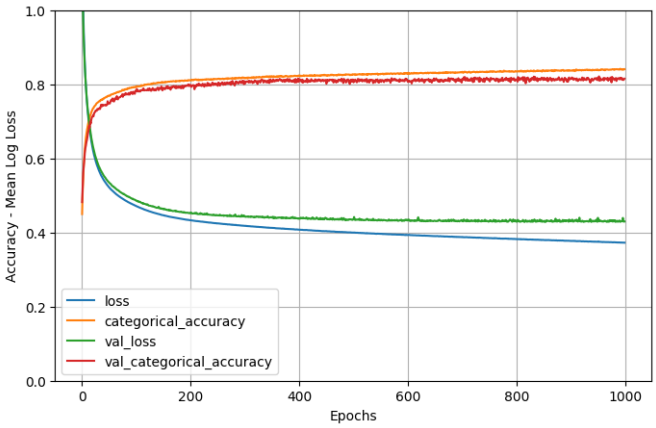

Figure 1: Entrenamiento durante el Experimento 2 de la Arquitectura 1

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.

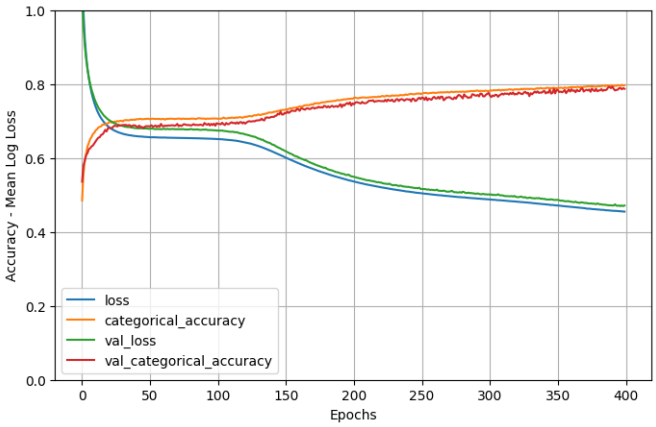

Figure 2: Entrenamiento durante el Experimento 3 de la Arquitectura 1

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

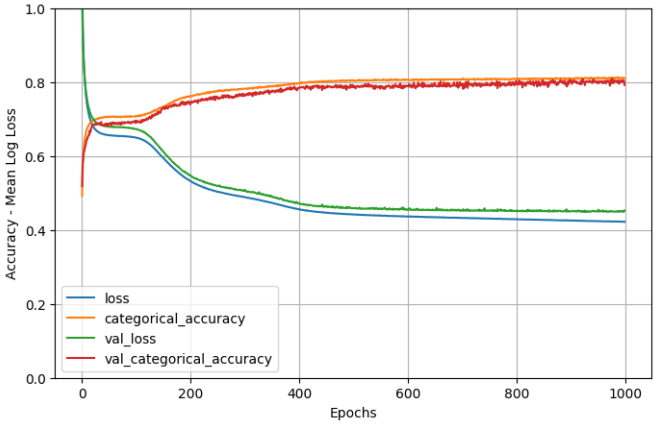

Figure 3: Entrenamiento durante el Experimento 4 de la Arquitectura 1

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.

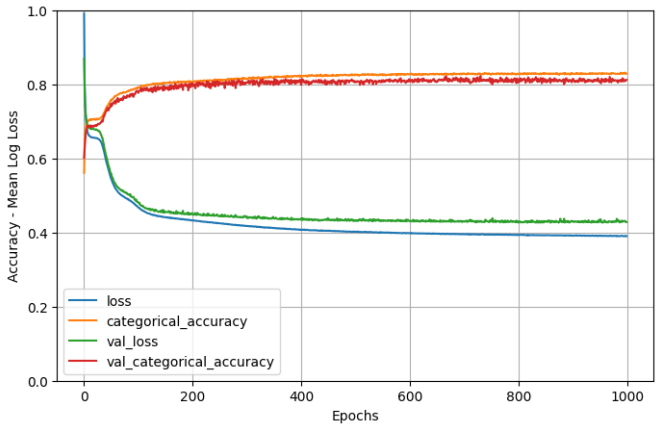

Figure 4: Entrenamiento durante el Experimento 5 de la Arquitectura 1

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.

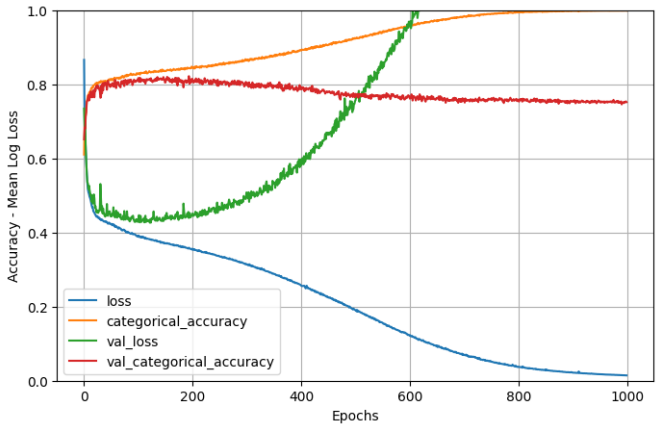

Figure 5: Entrenamiento durante el Experimento 1 de la Arquitectura 2

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	**l2 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.

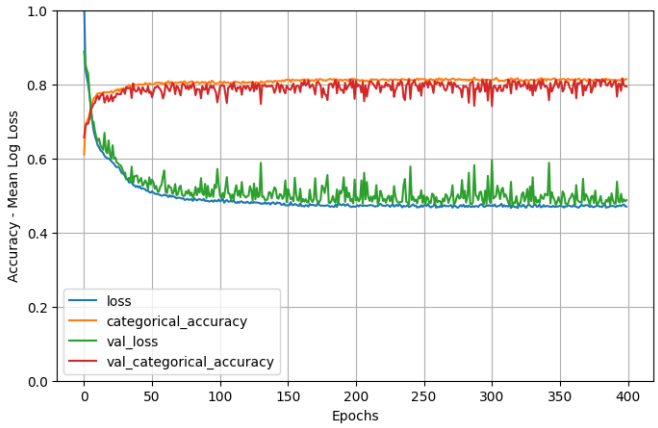

Figure 6: Entrenamiento durante el Experimento 2 de la Arquitectura 2

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**	l2 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.

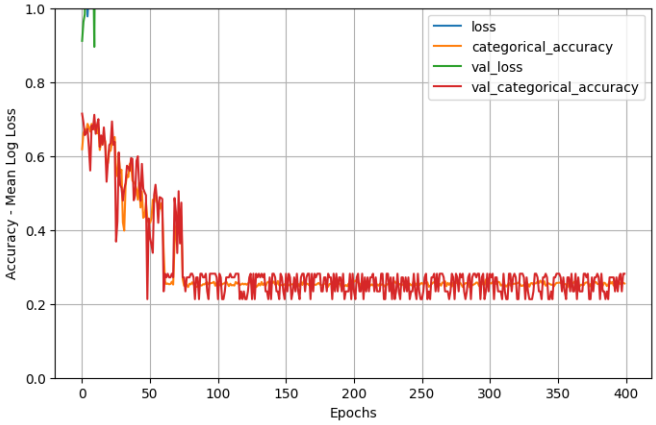

Figure 7: Entrenamiento durante el Experimento 4 de la Arquitectura 2

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

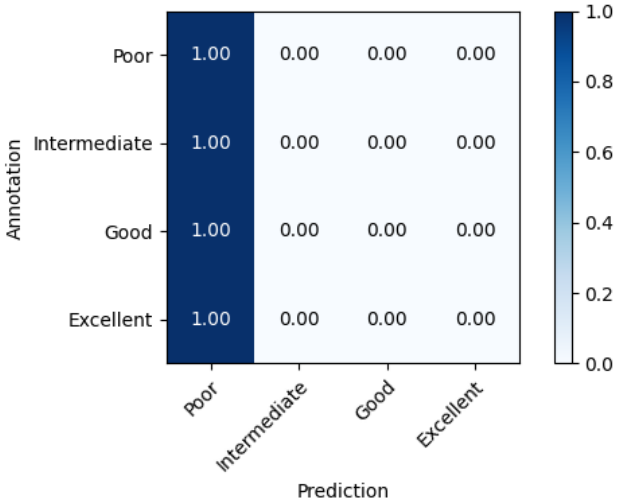

Figure 8: Matriz de confusión en el Experimento 3 de la Arquitectura 2

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 (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

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 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*.

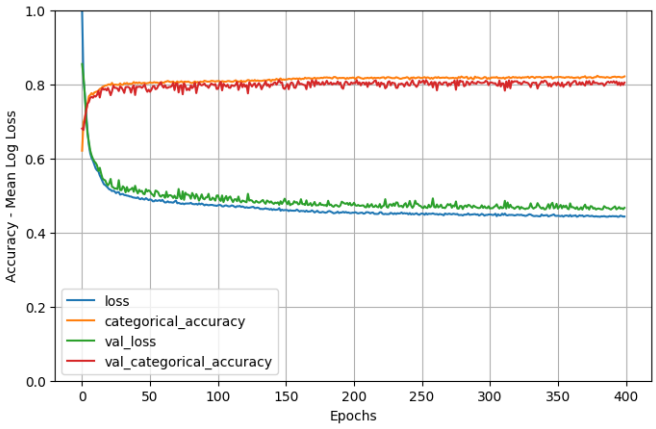

Figure 9: Entrenamiento durante el Experimento 4 de la Arquitectura 2

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

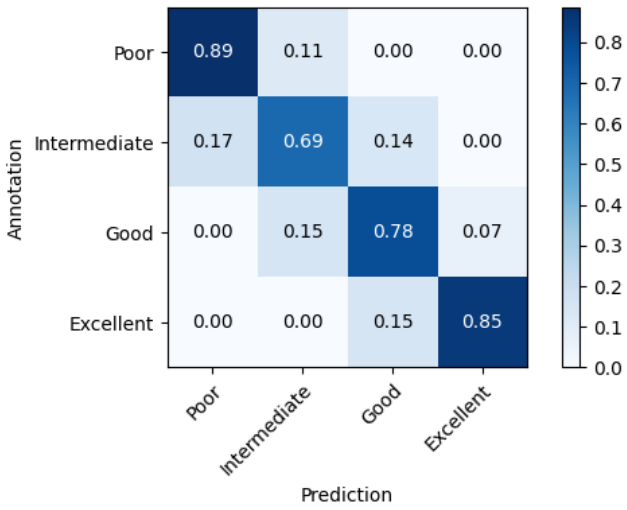

Figure 10: Matriz de confusión en el Experimento 4 de la Arquitectura 2

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	l2 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

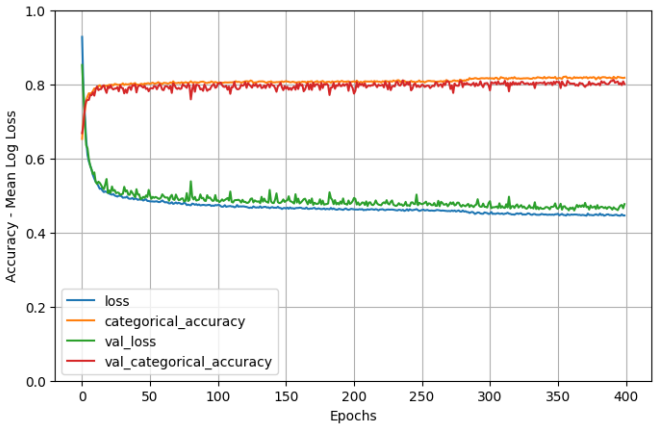

Figure 11: Entrenamiento durante el Experimento 5 de la Arquitectura 2

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	l2 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.

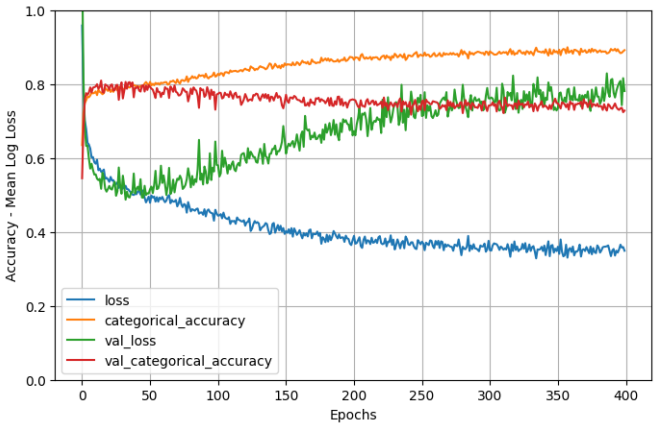

Figure 12: Entrenamiento durante el Experimento 1 de la Arquitectura 3

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*