Is it going to rain tomorrow in Australia?

Hands-On Project Unit 1, Aprendizaje Automático II

8 de marzo de 2025

GRUPO 9

Eduardo Álvarez Rey (eduardo.alvarez.rey@alumnos.upm.es)
Juan Bretones Moya (juan.bretones@alumnos.upm.es)
Álvaro Huisman (alvaro.huisman@alumnos.upm.es)
Alba Lozano Guixà (alba.lozano@alumnos.upm.es)

ÍNDICE

INTRODUCCIÓN

En este proyecto se busca abordar la construcción de una red neuronal profunda capaz de resolver un problema de clasificación binaria. Este se basa en predecir si mañana lloverá o no en distintas regiones de Australia en base a la variable objetivo *RainTomorrow.* El dataset sobre el que se va a trabajar recoge el histórico de precipitaciones a lo largo de diez años y se puede encontrar en el siguiente enlace:

https://www.kaggle.com/datasets/arunavakrchakraborty/australia-weather-data.

El dataset en crudo presentaba un total de 99.516 observaciones y 21 variables predictoras, además de la variable objetivo, tras ejecutar el NoteBook *PreparingAustraliaWeatherDataset.ipnyb* que contenía el preprocesamiento y limpieza del dataset nos quedamos con 39.574 instancias y el mismo número de variables.

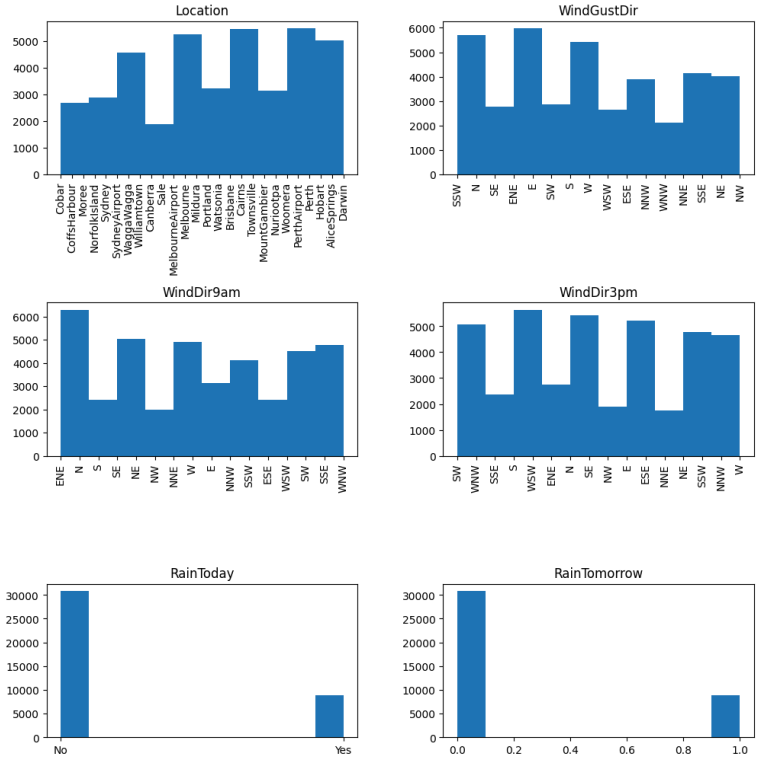
Para el desarrollo del proyecto, se empleó el notebook *PreparingAustraliaWeatherDataset.ipynb*, cuyo objetivo principal es limpiar y preparar el conjunto de datos meteorológicos de Australia. Este dataset, originalmente compuesto por más de 99.000 observaciones recopiladas a lo largo de una década en distintas localidades del país, incluye una amplia gama de variables climáticas como temperaturas, humedad, presión atmosférica, dirección y velocidad del viento, entre otras. El propósito de este proceso fue transformar los datos crudos en un formato estructurado y adecuado para el entrenamiento de una red neuronal que permita predecir la probabilidad de lluvia al día siguiente, representada por la variable objetivo *RainTomorrow*.

Durante la preparación, se eliminaron aquellos atributos considerados irrelevantes o que presentaban un elevado porcentaje de valores faltantes. Por ejemplo, variables como *Evaporation* y *Sunshine* fueron descartadas al presentar más del 30% de datos nulos. Para el resto de las variables, se aplicaron técnicas de imputación: en el caso de los atributos numéricos se utilizó la media, mientras que para las variables categóricas se optó por la moda. Posteriormente, se codificaron todas las variables categóricas, incluyendo las direcciones del viento y la variable *RainToday*, convirtiéndolas a formato numérico. Con el fin de facilitar el proceso de entrenamiento y mejorar la eficiencia del modelo, se normalizaron también los atributos numéricos, estandarizando su escala.

Como resultado de todo este procedimiento, se generaron dos archivos esenciales: *AustraliaWeatherPreparedCleanAttributes.csv*, que contiene los atributos ya procesados y listos para servir como entrada del modelo, y *AustraliaWeatherOutput.csv*, que recoge la variable objetivo codificada como 0 (no lloverá) o 1 (sí lloverá). Estos archivos conforman el conjunto de datos limpio y final sobre el cual se desarrollará y evaluará la red neuronal propuesta en este trabajo.

Además del procesamiento de los datos, durante esta etapa se generaron diversas gráficas y análisis estadísticos que ofrecieron una visión más clara sobre la composición del conjunto de datos. Las gráficas de barras, por ejemplo, muestran la distribución de frecuencias de algunas variables categóricas.

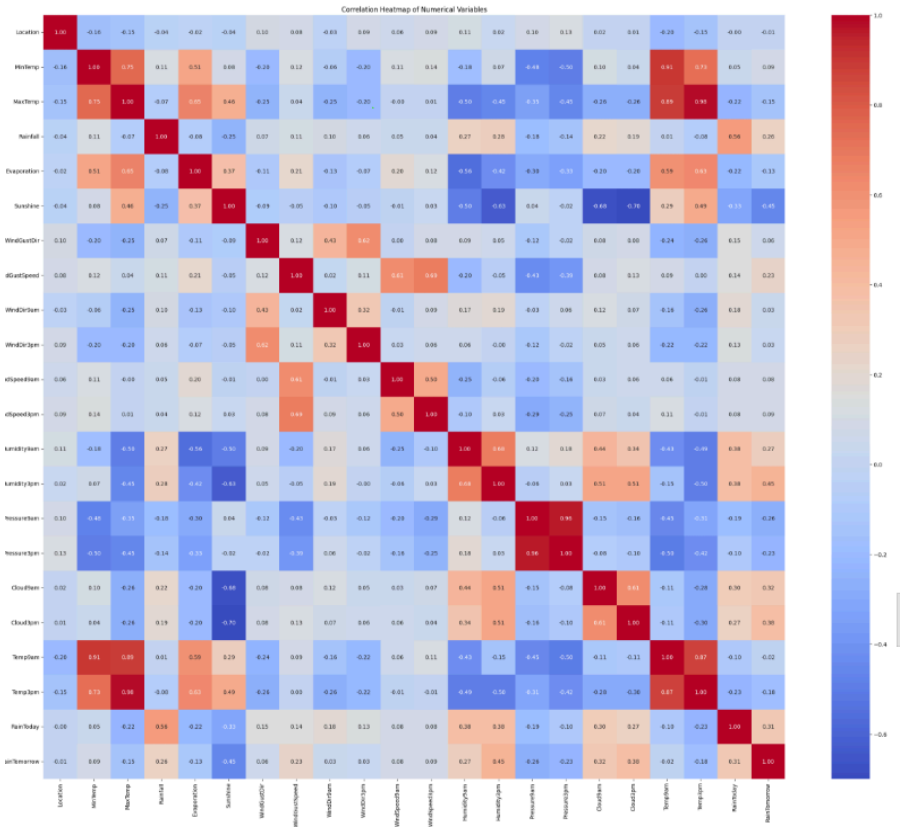
Se observa que la variable ***location*** tiene una distribución desigual entre las distintas ciudades australianas, lo que podría reflejar un sesgo geográfico en la recolección de datos. Las variables relacionadas con la dirección del viento — ***wind gust dir***, ***wind dir 9am*** y ***wind dir 3pm*** — también muestran distribuciones heterogéneas, lo cual indica que algunas direcciones cardinales se presentan con mayor frecuencia que otras.



Por otro lado, las gráficas correspondientes a ***rain today*** y ***rain tomorrow*** revelan un desbalance claro en la proporción de días con lluvia frente a días sin lluvia. La mayoría de los registros pertenecen a días sin lluvia, un aspecto importante a tener en cuenta al diseñar y entrenar el modelo, ya que este desequilibrio puede afectar su capacidad para detectar correctamente los casos minoritarios (días con lluvia) si no se aplican estrategias de balanceo adecuadas.

Asimismo, se generó una matriz de correlación entre las variables numéricas con el objetivo de identificar relaciones lineales relevantes. En ella se aprecia una fuerte correlación entre variables relacionadas, como *MinTemp* y *MaxTemp*, o *Temp9am* y *Temp3pm*, lo cual resulta esperable dado que reflejan distintos momentos del día. También se observa una alta correlación entre *Pressure9am* y *Pressure3pm*, lo que podría indicar cierta redundancia en la información.

En cuanto a la variable objetivo, *RainTomorrow* muestra una correlación positiva moderada con *RainToday*, y negativa con *Sunshine* y *Pressure*, lo que sugiere que días con menor insolación o baja presión tienden a anticipar lluvia al día siguiente. Este análisis es valioso para orientar la selección de atributos en el modelo predictivo.



En conjunto, estos análisis visuales y estadísticos no solo permitieron detectar patrones y posibles sesgos en los datos, sino que también resultaron fundamentales para orientar las decisiones durante la fase de limpieza y preparación. Gracias a ello, se logró establecer una base sólida sobre la cual desarrollar el posterior modelo de aprendizaje profundo.

PROCESO DE DISEÑO

Todas los resultados presentados a continuación han sido obtenidos desde un .ipnyb en Google Collab, empleando Python 3 y GPU T4 como soporte para la ejecución.

Modelo inicial

(398,29 segundos)

Para obtener una primera aproximación a la resolución del problema, optamos por construir una **red neuronal no profunda**, con una única capa oculta. Esto se debe a que sabemos que cualquier problema puede ser resuelto con una única capa oculta con la función de activación adecuada, gracias al **Teorema de Aproximación Universal**. Así mismo, como el problema es de clasificación binario, se decidió emplear la **función sigmoide** como función de activación.

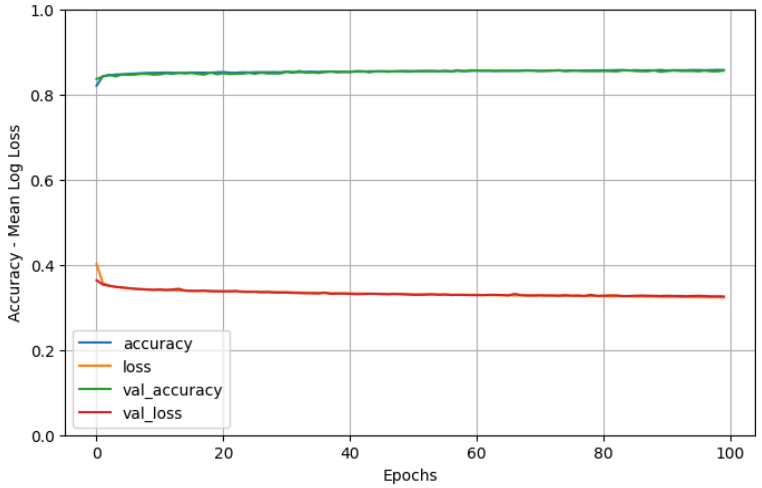
En cuanto a la arquitectura escogida, comenzamos estableciendo que el número de épocas sería 100, la tasa de aprendizaje 0,01 y una única capa con 20 neuronas. Esta decisión se basa en que pese a que cualquier problema puede ser resuelto con una única capa, el tiempo de cómputo y los resultados obtenidos son muy mejorables por redes profundas.

```
n_epochs = 100
lr = 0.01
n_neurons_per_hlayer = [20]
```

```
for neurons in n_neurons_per_hlayer:
    x = Dense(units=neurons, activation= 'sigmoid')(x)


# Capa de salida (Binaria → `sigmoid`)
output_layer = keras.layers.Dense(units=1, activation='sigmoid')(x)
```

Observando los resultados obtenidos, vemos como el *accuracy* en entrenamiento es algo menor que el *accuracy* en validación, ambos estando alrededor del 85%. Es por esto que consideramos esta primera solución **un buen comienzo**, sin embargo hay que buscar estrategias que permitan **reducir el tiempo** de ejecución y **mejoren las prestaciones** de la red.



	accuracy	loss
99	0.857513	0.323441

val_accuracy	val_loss
0.855951	0.325444

Métrica	Valor
Bias	0,0925
Varianza	0,0016

En cuanto a los valores de *bias* y varianza el modelo muestra un buen equilibrio; una varianza casi nula y un bias moderado. Este modelo parece tener una buena capacidad para generalizar. Aunque se busca mejorar la eficiencia del modelo es un buen punto de partida.

Modelo inicial profundo

(254 segundos)

Con fin de reducir el tiempo de ejecución y ser capaces de extraer relaciones más complejas entre los datos, lo que permite obtener mayor *accuracy* en la resolución del problema, se dejó atrás la red no profunda y se establecieron los siguientes hiperparámetros y arquitectura:

```
n_epochs = 500
lr = 0.001
batch_size = 512
n_neurons_per_hlayer = [ 500, 250, 150, 75, 25]
```

Se estableció un **número relativamente alto de épocas** (500) con fin de permitir un entrenamiento inicial profundo de la red. El valor de *learning rate* fue establecido siguiendo un criterio de equilibrio entre **estabilidad y convergencia rápida** (0.001). El *batch size* se estableció en 512, un tamaño grande que **estabiliza el descenso del gradiente** y permite un uso eficiente de la GPU.

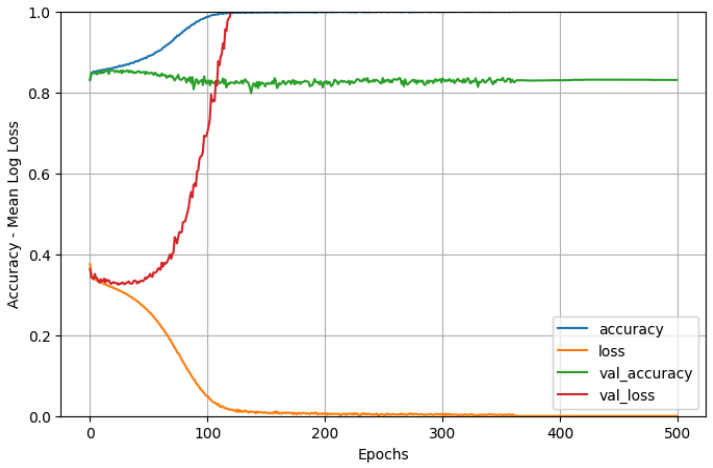
En cuanto a la arquitectura seleccionada, se estableció una **disminución progresiva en el número de neuronas por capa** para favorecer la extracción de características evitando el aumento excesivo de la complejidad del modelo.

En lo que a las capas ocultas se refiere, decidimos emplear la **función tanh**, ya que no anula los valores negativos y nuestros datos de entradas presentaban multitud de ellos. En este primer modelo profundo se decidió no emplear **ninguna técnica de regularización**.

El **criterio de optimización decretado** para esta red de neuronas inicial es **RMSProp** (Root Mean Square Propagation) que se usó buscando una mayor estabilidad en el entrenamiento.

Los resultados que se obtuvieron tras el entrenamiento con este modelo son los siguientes:

	accuracy	loss	val_accuracy	val_loss
499	1.0	0.000006	0.83068	1.894698



En vista a los resultados obtenidos, observamos que este modelo presenta un alto rendimiento en el entrenamiento pero muy bajo ante el conjunto de validación. La varianza obtenida por tanto, es muy elevada a lo largo de todo el proceso.

Por estos valores podemos afirmar que se está **memorizando el conjunto de entrenamiento,** es decir, estamos ante un claro caso de ***overfitting***.

Métrica	Valor
Bias	-0,05
Varianza	0,16932

En este caso, las métricas corroboran la presencia de *overfitting.* Se presenta una varianza considerablemente alta evidenciando que el modelo memoriza el conjunto de entrenamiento. El *bias* negativo puede deberse a este comportamiento de memorización del modelo.

El ***overfitting*** generado pudo ser causado por el uso de la función *tanh* en las capas ocultas pues esta función de activación presenta el problema de los ***Vanishing Gradients,*** los gradientes que se propagan hacia las primeras capas son extremadamente pequeños, dificultando el aprendizaje correcto en esas capas. Además, el criterio de optimización usado no evita el *overfitting*. Por ello, en vista de los resultados y sus posibles causas, se plantearon las siguientes soluciones en busca de paliar el *overfitting*:

1. **Añadir datos al conjunto de entrenamiento** (lo descartamos ya que no podemos modificar los datos en crudo o aumentar el porcentaje de datos de entrenamiento).
2. Añadir un **parámetro de regularización** para reducir la complejidad del modelo y reducir la sensibilidad a datos específicos.
3. Emplear ***Early Stopping*** para detener el entrenamiento en el momento en el que el error de validación comience a aumentar mientras el de entrenamiento sigue disminuyendo.

Modelo con regularización L2

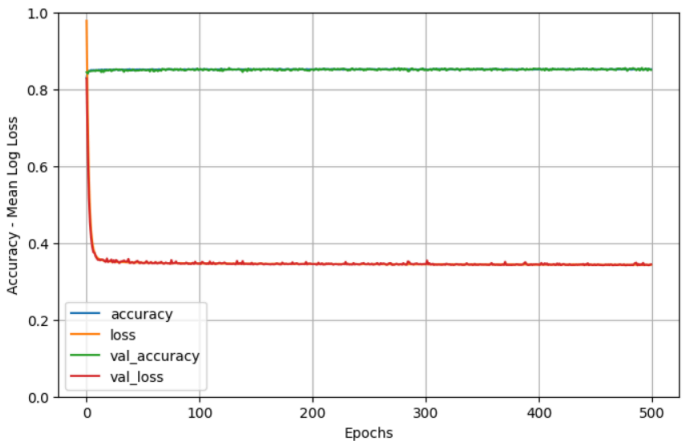
(175 segundos)

Continuando con las funciones, criterio de optimización e hiperparámetros escogidos en el primer experimento se decidió introducir un **parámetro de regularización** buscando obtener un modelo que **generalice mejor datos nuevos**. Se escogió la regularización **L2** con un valor de 0.001 penalizando así valores grandes en los pesos de la red neuronal y evitando un ajuste excesivo a los datos de entrenamiento.

```
# Regularizador L2 muy pequeño
l2_regularizer = l2(0.001)
```

Los resultados obtenidos fueron los siguientes,

	accuracy	loss	val_accuracy	val_loss
499	0.851859	0.343204	0.850139	0.344048



Observamos cómo esta configuración **no presenta *overfitting***, no se observa apenas diferencia entre el *accuracy* en entrenamiento y el *accuracy* en validación, es decir, la **varianza del modelo es mínima**. Por lo tanto, el diagnóstico anterior era correcto. Sin embargo, nuestro modelo sigue teniendo un **tiempo de ejecución elevado**, con fin de disminuirlo continuamos con otro experimento.

Métrica	**Valor**
Bias	0,0982
Varianza	0,0017

Este modelo hace evidente la mejora con respecto al anterior. El *overfitting* se elimina casi por completo y se mantiene un valor de *bias* aceptable. Se puede decir que la implementación de L2 fue efectiva para contener la complejidad manteniendo el rendimiento.

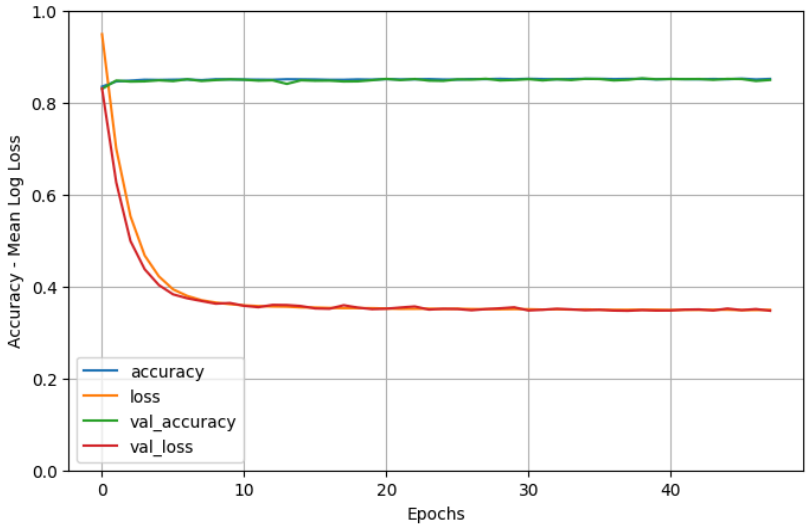
Modelo con regularización L2 y *Early Stopping*

(21 segundos)

La configuración de los hiperparámetros es la misma que en el modelo anterior, añadiendo ***Early Stopping*** para reducir el tiempo de ejecución.

```
early_stopping = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
```

Observamos de nuevo como el *overfitting* ya no está presente. Por otro lado, se obtienen resultados muy parecidos a los anteriores, con un **tiempo de ejecución mucho menor**. Es por esto que consideramos que la **inclusión del *Early Stopping*** en la configuración de nuestras redes neuronales **optimiza la eficiencia en el entrenamiento**.



	accuracy	loss
47	0.851353	0.349183

val_accuracy	val_loss
0.849128	0.347164

Métrica	**Valor**
Bias	0,0987
Varianza	0,0022

Los resultados en estas métricas son parecidos a los del modelo anterior aunque con un tiempo de ejecución considerablemente menor. Esta reducción seguramente sea gracias a la implementación de *Early Stopping.*

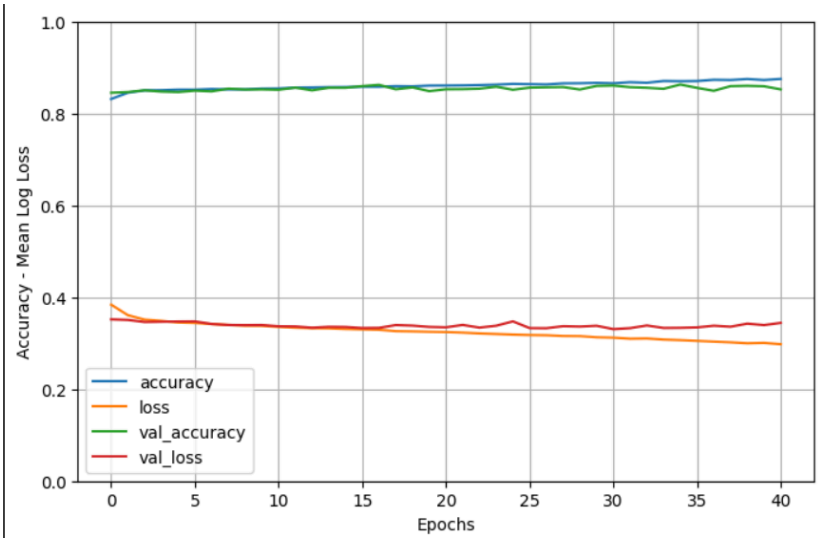
Modelo con regularización L2 más moderada y *Early Stopping*

(18 segundos)

De nuevo la configuración de la red neuronal es idéntica a los casos anteriores, la única diferencia es el **decremento en el valor de L2** (de 0.001 a 0.00001, i.e., cuatro veces más pequeño). Así se podría dar el caso de un mejor rendimiento ya que un valor menor de L2 puede **permitir que los pesos de la red crezcan más**.

```
# Regularizador L2 muy pequeño
l2_regularizer = l2(0.00001)
```

Observamos como efectivamente se da una **mejora en el rendimiento de la red con el decremento del valor de L2**, tanto en entrenamiento como en validación. En entrenamiento mejora más de 2% y se da una leve mejora en validación. Cabe destacar, cómo de nuevo podemos observar una r**educción en el tiempo de ejecución** gracias a la inclusión del *Early Stopping* en la configuración de la red.



	accuracy	loss
40	0.875201	0.297834

val_accuracy	val_loss
0.852413	0.344287

Métrica	Valor
Bias	0,0748
Varianza	0,0228

Con respecto al modelo anterior se aprecia una reducción del *bias* y un ligero aumento de la varianza. Estos cambios son causados por la reducción de la fuerza de regularización del L2, que permitió mayor capacidad de aprendizaje pero también un incremento tenue del riesgo de *overfitting*.

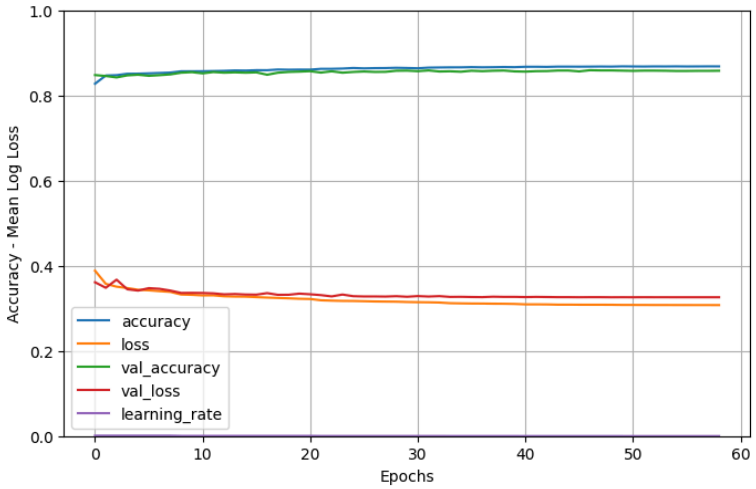
Modelo con regularización L2 más moderada, *Early Stopping* y *Lr Schedule*

(25 segundos)

Vamos a seguir buscando mejoras en el rendimiento con la misma configuración que la red anterior, **añadiendo el Lr Schedule que ajusta dinámicamente la tasa de aprendizaje** durante el proceso de entrenamiento. De esta manera se pretende mejorar la convergencia y el rendimiento del modelo. Se ha implementado la política ***Reduce on Plateau*** que disminuye la tasa de aprendizaje cuando el rendimiento deja de mejorar.

```
lr_schedule = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)
```

Observamos un **empeoramiento en el *accuracy* en entrenamiento** y una **leve mejora en el *accuracy* en validación**, con un **mayor tiempo de ejecución** que el modelo anterior pese a la inclusión del *Early Stopping*, por lo que se seguirán buscando posibles configuraciones en busca del consenso idóneo entre rendimiento y tiempo de ejecución.



	accuracy	loss
58	0.868252	0.307636

val_accuracy	val_loss	learning_rate
0.857973	0.326047	0.000002

Métrica	Valor
Bias	0,0818
Varianza	0,0103

Al añadir el *Lr Schedule* se consiguió una sutil mejora en validación pero también aumentó el tiempo de ejecución. La varianza baja un poco y el *bias* sube en relación al modelo precio aunque no de una manera significativa. No hay mejora contundente.

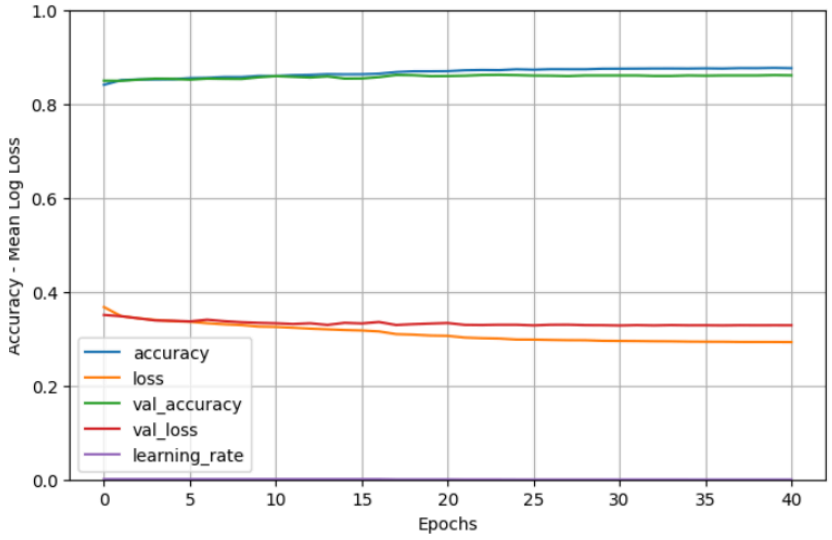
Modelo anterior cambiando el optimizador

(20 segundos)

El siguiente cambio que se decidió implementar en la red fue **cambiar el RMSProp por una variante de Adam**, optimizador que combina RMSProp con Momentum. La variante escogida ha sido **Nadam** que incorpora NAG, permitiendo menos oscilaciones y una convergencia más rápida y estable. Por otro lado, se ha empleado el Learning Rate 0.001.

```
# Compilar modelo
model.compile(optimizer=keras.optimizers.Nadam(learning_rate=lr),
```

En vista a los resultados obtenidos, se observa una **mejora en el tiempo de ejecución** (5 segundos menos que el modelo anterior) y una leve mejora en el *accuracy* en validación. Sin embargo, el *accuracy* en entrenamiento empeora ligeramente, por lo tanto se consideró que esta solución sigue sin ser la idónea.



	accuracy	loss	val_accuracy	val_loss	learning_rate
40	0.876023	0.292784	0.860753	0.328599	0.000008

Métrica	Valor
Bias	0,0740
Varianza	0,0153

La implementación con Nadam provocó una ligera mejora en el balance entre varianza y *bias.* Son métricas mejores en un modelo más estable.

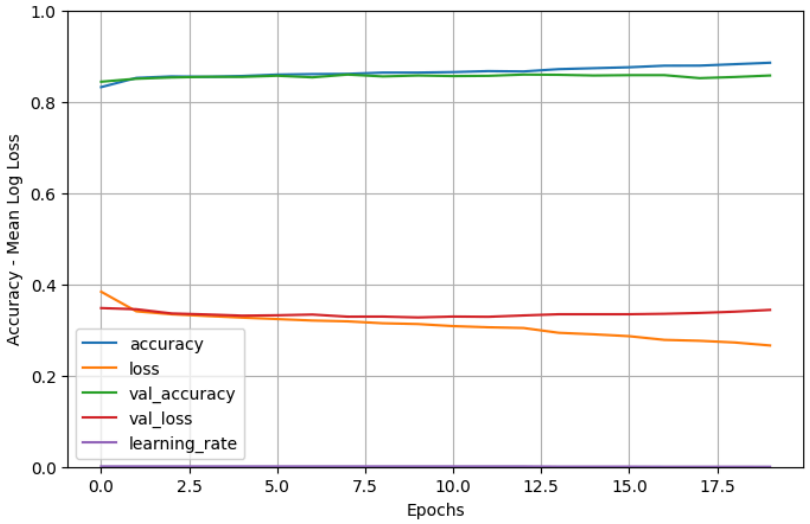
Modelo con función de activación RELU

(13,76 segundos)

En este modelo de red neuronal, la configuración de los hiperparámetros es la misma que en los modelos anteriores, sin embargo la **función empleada en las capas ocultas es RELU en vez de tanh**, con la idea de intentar solventar el problema del *Vanishing Gradients*.

```
for neurons in n_neurons_per_hlayer:
    x = Dense(units=neurons, kernel_regularizer = l2_regularizer)(x)  # Sin activación aquí
    x = Activation(relu)(x) # Activación RELU
```

A la vista de los resultados obtenidos, se aprecia que vuelve a estar presente cierto *overfitting.* De esta manera, se plantearon de nuevo las soluciones propuestas en el primer experimento.



	accuracy	loss	val_accuracy	val_loss	learning_rate
19	0.885435	0.265932	0.857468	0.343869	0.000125

No debe pasarse por alto que este no es un caso de *overfitting* tan claro como el del primer modelo. La razón de esta mejora se debe al uso de ***Early Stopping***, ya que hace que el entrenamiento del modelo se detenga cuando no hay mejora en el conjunto de validación.

Métrica	Valor
Bias	0,0646
Varianza	0,0280

Cambiando *tanh* por ReLu aumentó la varianza pero el *bias* se redujo considerablemente. Esta tendencia revela cierto riesgo de *overfitting*, algo que puede estar causado porque la nueva función de activación permite un aprendizaje más agresivo. Por ello se plantea una regularización más severa.

Modelo con RELU y un valor de L2 mayor

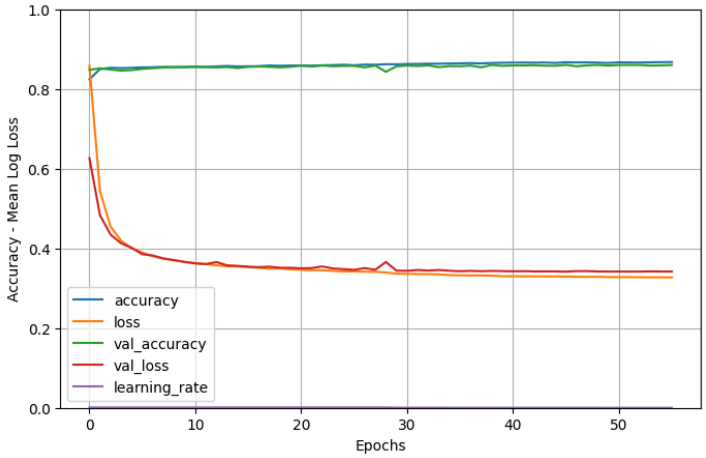
(26,86 segundos)

En este experimento el objetivo que se planteó fue la reducción rápida del *overfitting* que apareció en el modelo anterior. Para ello se asumió el riesgo de una disminución considerablemente la capacidad de aprendizaje de la red y se añadió un **coeficiente L2 relativamente alto** (de 0,01).

```
# Regularizador L2 mas grande
l2_regularizer = l2(0.001)
```

	accuracy	loss	val_accuracy	val_loss	learning_rate
55	0.867463	0.327184	0.859995	0.341958	0.000016

Al analizar los resultados se detecta de nuevo un **peor desempeño general de la red**, ya que empeora en tiempo de ejecución y *accuracy* en entrenamiento, mientras que se da una leve mejora en el *accuracy* en validación. Por ello, se decidió implementar otra técnica de regularización.



Métrica	Valor
Bias	0,0826
Varianza	0,0075

Incrementando la regularización L2 se consiguió una reducción considerable de la varianza aunque también se aumentó el *bias.* Aunque esto sugiere una ligera pérdida del modelo en la capacidad del aprendizaje es una configuración conservadora que puede ser interesante si se prioriza evitar un posible *overfitting*.

Modelo con RELU y *DropOut*

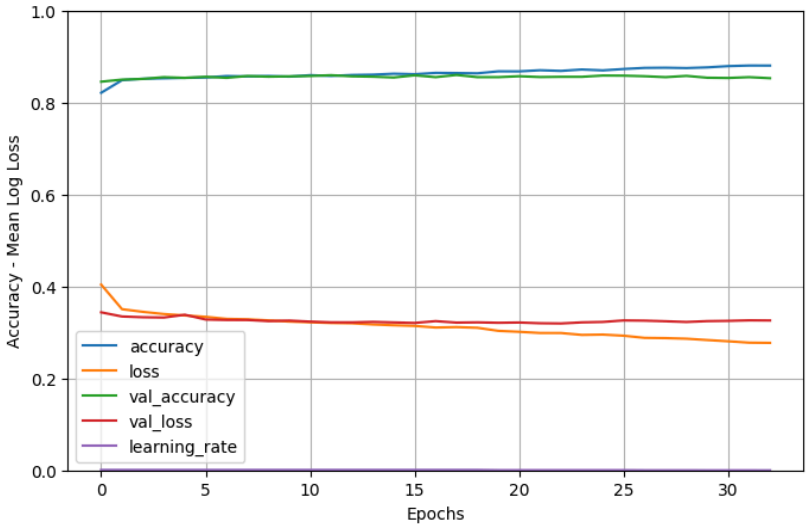
(20,33 segundos)

Entre los distintos métodos de regularización se optó por implementar ***DropOut*** con un coeficiente de 0,2. Esta técnica apaga aleatoriamente algunas neuronas durante el entrenamiento. En este caso, el 20% de las neuronas de cada capa de la red se irán apagando de manera aleatoria, de esta manera, se evita que el modelo dependa de un subconjunto restringido de neuronas exclusivamente .

```
# Regularizador dropout
dropout_rate = 0.2
```

```
for neurons in n_neurons_per_hlayer:
    x = Dense(units=neurons)(x)  # Sin activación aquí
    x = Activation(relu)(x) # Activación RELU
    x = Dropout(dropout_rate)(x)  # Dropout después de la activación
```

En vista a estos resultados, se aprecia una **mejora en el tiempo de ejecución y en el *accuracy* en entrenamiento** con respecto al modelo anterior. Sin embargo, el ***accuracy* en validación** disminuye lo que nos indica una **pérdida en la capacidad de generalización** de nuestra red, ya que ha aumentado la varianza.



	accuracy	loss	val_accuracy	val_loss	learning_rate
32	0.880382	0.277319	0.852919	0.326017	0.000063

Métrica	Valor
Bias	0,0697
Varianza	0,0274

La implementación con *DropOut* mantuvo bajo el *bias* pero provocó un aumento de la varianza. Aunque se redujo el tiempo de ejecución estas métricas indican cierta pérdida en la capacidad de generalización.

RESULTADOS FINALES

Tras el análisis de los resultados de los experimentos realizados se identifica que los que ofrecen **mejores datos** de *accuracy*, varianza y *bias* así como de estabilidad y tiempo de ejecución son **el modelo implementado con el optimizador Nadam** y **el que aplica ReLU y un regularizador L2 mayor**.

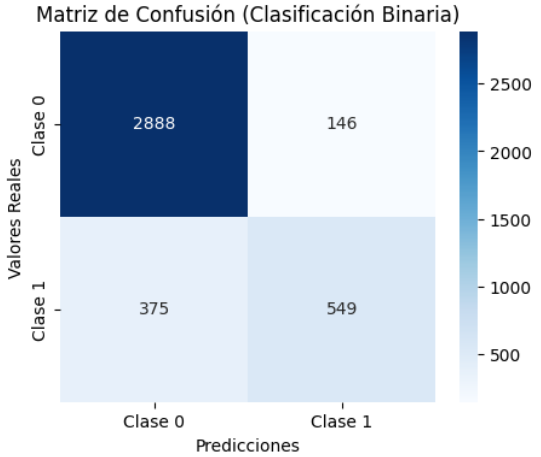
Modelo	Accuracy	Loss	Val_Accuracy	Val_Loss	Learning Rate	Tiempo (s)	Hiperparámetros
Modelo con optimizador Nadam	0.876023	0.292784	0.860753	0.328599	0.000008	20	Optimizer: **Nadam** Activation: **Tanh (ocultas), Sigmoid (salida)** Regularization: **Sin regularización**
Modelo con ReLU y mayor regularización L2	0.867463	0.327184	0.859995	0.341958	0.000016	26.86	Optimizer: **Nadam** Activation: **ReLU (ocultas), Sigmoid (salida)** Regularization: **L2 = 0.01**

Los dos modelos tienen **Batch Size 512** y número de épocas **variable por Early Stopping**

Entre estos dos modelos, se ha seleccionado aquel con **regularización L2** porque, aunque su *accuracy* en entrenamiento (0.867) es ligeramente menor que la del modelo sin regularización (0.876), sus valores de *accuracy* en validación (0.859, 0.860) son prácticamente iguales, lo que indica una mejor capacidad de generalización. Asimismo, el *Loss* (valor de la función pérdida en validación) es ligeramente mayor (0.3419 vs. 0.3286). Además, la regularización L2 contribuye a la reducción del riesgo de *overfitting*, asegurando que el modelo no dependa demasiado de los datos de entrenamiento.

Por otro lado, a pesar de tener un tiempo mayor de entrenamiento (26.86s vs. 20s), este incremento está justificado teniendo en cuenta que el modelo con L2 es más estable y presenta mejor capacidad de generalización. Además, el uso de la activación **ReLU** en lugar de **Tanh** puede favorecer una mejor eficiencia en redes profundas (evitando el problema de los *Vanishing Gradients)*, consolidando nuestra elección por el **modelo con ReLU y mayor regularización L2**.

Una vez decidido el mejor modelo se aplicó a los datos de test obteniendo las siguientes métricas de evaluación y matriz de confusión:



Métricas de Evaluación:

	precision	recall	f1-score	support
0	0.89	0.95	0.92	3034
1	0.79	0.59	0.68	924
accuracy			0.87	3958
macro avg	0.84	0.77	0.80	3958
weighted avg	0.86	0.87	0.86	3958

Al analizar la matriz de confusión se hace evidente el buen desempeño del modelo escogido en la predicción de la clase 0 (días sin lluvia), con **2888 verdaderos negativos (VN)** que evidencian la sólida capacidad de identificar correctamente días sin lluvia. Por otro lado se detectan **549 verdaderos positivos (VP)**, es decir, predicciones correctas de lluvia.

Sin embargo, también se registran **375 falsos negativos (FN)**, casos en los que se predijo la ausencia de lluvia aunque sí la hubo; y **146 falsos positivos (FP),** casos que se corresponden a días en los que no hubo precipitaciones pero el modelo detectó que sí.

Examinando las métricas de evaluación obtenidas por clase los resultados son:

- Para la clase 1 (días de lluvia), una *precision* del 79.12%, la mayoría de días lluviosos detectados fueron correctos; y un *recall* del 59.42% que evidencia que logra identificar poco más de la mitad de los días con lluvia dejando una considerable proporción de casos sin detectar.
- Para la clase 0 (días sin lluvia), se evaluan una *precision* del 88.51% y un *recall* del 95.18%, unas métricas que reflejan un buen desempeño tanto en la correcta detección como en la minimización de los FN.

Estas métricas evidencian que el modelo evita de manera sólida las falsas alarmas (alta *precision* en clase 0), pero es algo más conservador al detectar la lluvia, lo que podría ser un problema si se prioriza no omitir días lluviosos. Por tanto, según el contexto, podría resultar útil ajustar el umbral de decisión para mejorar el *recall* en la clase 1.

Conviene subrayar que este desequilibrio en el rendimiento entre clases puede deberse al desbalance del *dataset*. Al hacer comprobaciones se detectó que el número de observaciones que corresponden a días sin lluvia es significativamente mayor frente al número de los días en los que sí llovió. Esta desproporción puede provocar que el modelo favorezca a la clase mayoritaria (no lluvia), obstaculizando la correcta identificación de la minoritaria (lluvia).

CÓDIGO FUENTE

URL de acceso al NoteBook → AA2_Group9.ipynb

CONCLUSIONES

Durante el desarrollo de este proyecto comenzamos con un modelo simple, pensado para predecir si llovería o no al día siguiente en distintas regiones de Australia. Aunque al principio optamos por una red neuronal poco profunda y con una configuración básica, pronto nos dimos cuenta de que el tiempo de ejecución no era viable, esta primera aproximación no era suficiente aunque sí interesante como experimento. El siguiente modelo, creado con la esperanza de obtener mejores resultados y con una arquitectura ya profunda, presentaba claros signos de *overfitting*, lo que nos obligó a realizar cambios para mejorar su capacidad de generalización sin aumentar demasiado los recursos utilizados.

A partir de ahí, fuimos introduciendo mejoras de forma progresiva. Primero añadimos regularización para evitar que el modelo se ajustara demasiado a los datos de entrenamiento. Luego incorporamos técnicas como *Early Stopping* que nos permitió ahorrar tiempo de entrenamiento sin perder *accuracy*. Más adelante, hicimos ajustes en la tasa de aprendizaje y probamos con un optimizador distinto que ofreciera mayor estabilidad y velocidad en la convergencia. Todos estos cambios nos permitieron analizar de manera concreta los efectos que produce la implementación con distintos métodos, algo no únicamente necesario para el desarrollo del proyecto sino que también nos ayudó a entender la teoría aprendida en clase.

Además, cambiamos la función de activación por una opción que nos ayudara a evitar problemas comunes en redes profundas, como el problema de los *vanishing gradients*. Aunque esta modificación trajo consigo un nuevo caso de *overfitting*, logramos controlarlo con las estrategias ya implementadas. Finalmente, experimentamos con técnicas como *DropOut*, lo que ayudó a reducir aún más la dependencia del modelo hacia ciertas partes de su arquitectura.

Después de todas estas pruebas, llegamos a un modelo sólido, eficiente y equilibrado. Fue capaz de rendir bien tanto con los datos de entrenamiento como con los de validación, obteniendo además buenos resultados en el conjunto de prueba. Con un *accuracy* cercano al 87%. Consideramos por tanto, que se alcanzó el objetivo principal del proyecto: construir un modelo fiable, con buen rendimiento y optimizado en cuanto al uso de recursos.

Sin embargo, es interesante destacar que durante el análisis e interpretación de los datos se detectó un gran desbalance entre las clases, siendo la mayoritaria la clase correspondiente a los días sin precipitaciones. El desequilibrio pudo haber influido en la capacidad del modelo de detectar de manera correcta la clase minoritaria (días lluviosos) perjudicando el rendimiento del modelo. En vistas a futuro, se plantea introducir más datos reales o incluso técnicas de balanceo como SMOTE, que permita un entrenamiento más equilibrado y una generalización adecuada para todos los contextos.

Además, también podría ser interesante el uso de otros tipos de normalización en el preprocesado. El método de estandarización (*StandardScaler*) podría plantear un enfoque diferente que cambiase el rendimiento y velocidad de convergencia del modelo final.

En resumen, aunque este campo ofrece un gran espacio de oportunidades de mejora con la aplicación de distintas estrategias sobre los datos, desarrollar el proyecto nos ha permitido entender mejor cómo ajustar una red neuronal para resolver un problema concreto, combinando teoría con pruebas prácticas, y siempre buscando el equilibrio entre complejidad, eficiencia y capacidad de generalización.