Integrantes:

- Felipe De Jesús Liévano Pinilla – 201924905
- Juan Martín Vásquez Cristancho – 202113314

Grupo: Yisus Martín

Taller 6: Introducción a Redes Neuronales

1. Clasificación con redes neuronales - Datos iris

Punto 1. Selección de Especie

Para realizar el modelo base se seleccionó la especie de Setosa y se creó la variable binaria para su uso:

```
y = tf.keras.utils.to_categorical(iris.target, num_classes=3)
y_s = y[:,[0]]
```

Punto 2. Creación del modelo base

División de los datos

Lo primero fue dividir los datos en los conjuntos de prueba, validación y testeo:

```
X_train_full, X_test, y_train_full, y_test = train_test_split(
    iris.data, y_s, test_size=0.2, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_full, y_train_full, test_size=0.2, random_state=42)
```

Normalización

Se crea el normalizador de los datos:

```
std_scl = StandardScaler()
std_scl.fit(X_train)
```

Modelo base

Se crea el modelo base como una red neuronal con capa de entrada, capa oculta de 8 neuronas y una capa de salida de una única neurona, esta última con función de activación sigmoide;

```
tf.random.set_seed(42)
tf.keras.backend.clear_session()
base = tf.keras.Sequential()
base.add(tf.keras.layers.InputLayer(input_shape=(4,)))
base.add(tf.keras.layers.Dense(8, activation="relu"))
base.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

Finalmente, se especifica el optimizador, en este caso se usa sgd, y se especifica la función de error, para este caso se usa la entropía cruzada binaria:

```
base.compile(loss="binary_crossentropy",
             optimizer="sgd",
             metrics=["accuracy"])
```

Punto 3. Entrenamiento y Prueba del modelo base

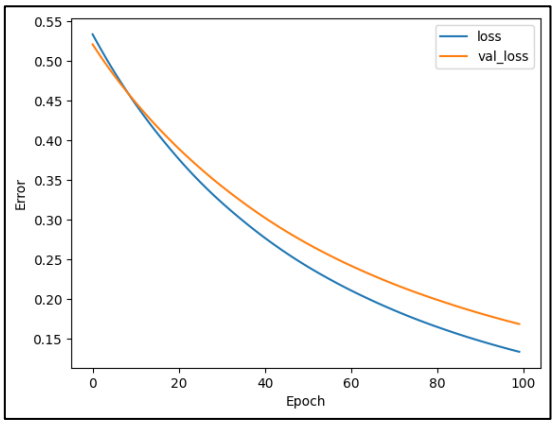
Entrenamiento

Se entrena al modelo con 100 épocas

```
history_base = base.fit(X_train, y_train, epochs=100,
                        validation_data=(X_valid, y_valid))
```

Grafica historial de pérdida de entrenamiento y validación

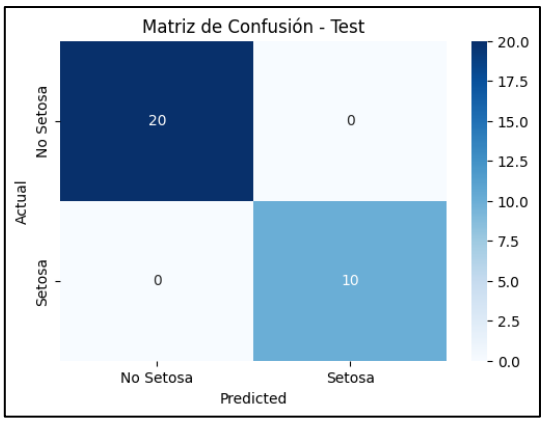
Una vez entrenado se grafica el historial de pérdida de entrenamiento y validación:



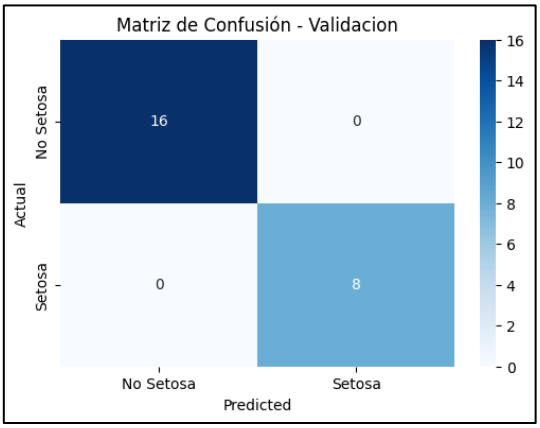
La gráfica muestra que la pérdida en entrenamiento y validación desciende de forma continua a lo largo de las épocas, lo cual evidencia que el modelo está aprendiendo de manera progresiva y mejorando su capacidad de clasificación. En concreto, la pérdida de entrenamiento disminuye más rápido debido a que el modelo se ajusta repetidamente a los mismos datos, mientras que la de validación también baja de forma estable, señal de que el modelo generaliza bien y no está sobreajustando en exceso. La brecha entre ambas curvas es razonable y no indica problemas importantes de sobreajuste, pues la curva de validación no se estanca ni repunta. Aunque es importante notar que la de validación esta por encima de la de entrenamiento.

Matriz de confusión y Métricas:

Una vez hecho esto se procede a graficar la matriz de confusión y calcular las métricas pertinentes para el modelo como lo es la accuracy, la precisión, el recall, el puntaje F1 y F2 y ROC AUC.




```
--- MÉTRICAS DEL MODELO (Validación) ---
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
F2 Score: 1.0000
ROC AUC Score: 1.0000
```



Las matrices de confusión muestran que el modelo clasifica correctamente todas las instancias, diferenciando sin errores entre “Setosa” y “No Setosa”. Esto implica que, en ambos conjuntos (prueba y validación), no se produce ninguna confusión al asignar las etiquetas, lo que evidencia que la estructura de la red (una capa oculta de 8 neuronas con activación ReLU) es adecuada para separar las clases en este problema específico.

Por otro lado, las métricas alcanzan valores perfectos (1.0 en exactitud, precisión, exhaustividad, F1 y AUC), lo que refuerza la observación de una separación total entre las clases. Aunque estos resultados son muy positivos, es recomendable validar que no exista sobreajuste mediante técnicas como la validación cruzada, especialmente si en el futuro se amplía la clasificación a otras especies o se incorporan más variables.

Punto 4. Nuevos modelos

Primer modelo

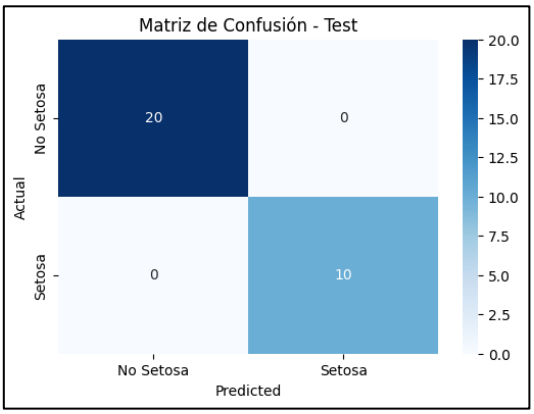
El primer modelo que se crea es una red neuronal con capa de entrada, tres capas ocultas con 10, 15 y 20 neuronas, además de la capa de salida con una única neurona,

al cual se le especifica la función de error (entropía cruzada binaria) y se usa como optimizador sgd:

```
tf.random.set_seed(42)
tf.keras.backend.clear_session()
m1 = tf.keras.Sequential()
m1.add(tf.keras.layers.InputLayer(input_shape=(4,)))
m1.add(tf.keras.layers.Dense(10, activation="relu"))
m1.add(tf.keras.layers.Dense(15, activation="relu"))
m1.add(tf.keras.layers.Dense(20, activation="relu"))
m1.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

```
m1.compile(loss="binary_crossentropy",
           optimizer="sgd",
           metrics=["accuracy"])
```

Una vez entrenado el modelo se grafica la matriz de confusión y calculan las métricas:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
F2 Score: 1.0000
ROC AUC Score: 1.0000
```

El nuevo modelo, con tres capas ocultas (10, 15 y 20 neuronas) y una salida de una sola neurona, clasifica perfectamente entre “Setosa” y “No Setosa”. La matriz de confusión muestra cero errores, y las métricas (recall, F1, F2, ROC AUC) alcanzan el valor máximo de 1.0.

Esta separación perfecta se atribuye, en gran medida, a la clara distinción de la clase Setosa frente a las demás. Aun así, se recomienda realizar validaciones adicionales, para descartar sobreajuste. Si el problema se amplía a más clases o variables, conviene revisar si la arquitectura de la red sigue siendo adecuada.

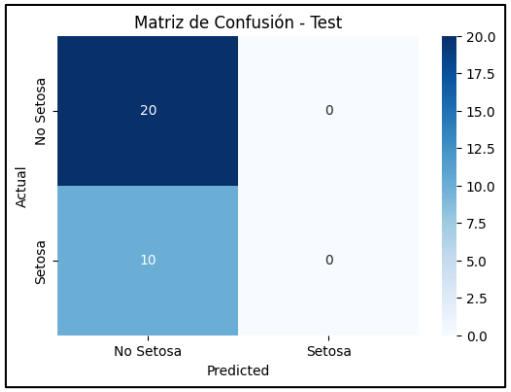
Segundo modelo

Ahora, se realiza un modelo que consta de una red neuronal con 5 capas ocultas de 5, 7, 21, 30 y 5 neuronas, además de la capa oculta de salida con una única neurona.

```python
m2 = tf.keras.Sequential()
m2.add(tf.keras.layers.InputLayer(input_shape=(4,)))
m2.add(tf.keras.layers.Dense(5, activation="relu"))
m2.add(tf.keras.layers.Dense(7, activation="relu"))
m2.add(tf.keras.layers.Dense(21, activation="relu"))
m2.add(tf.keras.layers.Dense(30, activation="relu"))
m2.add(tf.keras.layers.Dense(5, activation="relu"))
m2.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

```python
m2.compile(loss="binary_crossentropy",
           optimizer="sgd",
           metrics=["accuracy"])
```

Una vez entrenado se grafico la matriz de confusión y se calcularon las métricas pertinentes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6667
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
F2 Score: 0.0000
ROC AUC Score: 0.5000
```

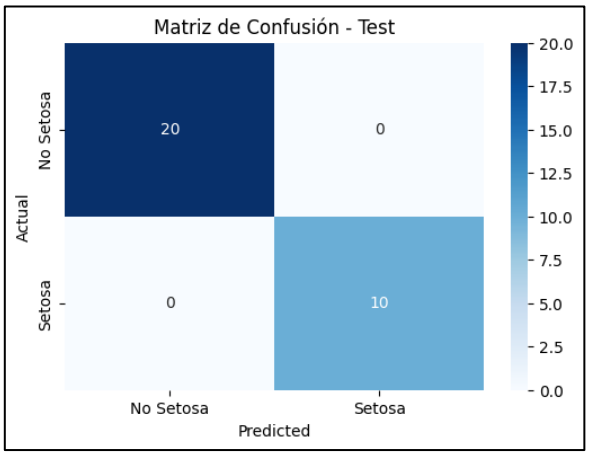
La matriz de confusión muestra que el modelo clasifica todas las instancias como “No Setosa”, sin reconocer ningún caso de “Setosa”. Esto explica la precisión, recall, F1 y F2 en 0, ya que el modelo nunca predice la clase positiva. La exactitud (66.67%) proviene únicamente de las instancias de “No Setosa” bien clasificadas, y el AUC de 0.5 indica un comportamiento equivalente a adivinar al azar. Para mejorar, conviene revisar la configuración de la red, la inicialización de pesos y la estrategia de optimización, así como verificar la distribución de clases y, de ser necesario, aplicar técnicas de balance o ajuste de hiperparámetros.

Tercer modelo

Por último, se crea un modelo como red neuronal con una capa de entrada, cuatro capas ocultas con 8, 20, 35 y 46 neuronas, además de la capa de salida con una única neurona.

```python
m3 = tf.keras.Sequential()
m3.add(tf.keras.layers.InputLayer(input_shape=(4,)))
m3.add(tf.keras.layers.Dense(8, activation="relu"))
m3.add(tf.keras.layers.Dense(20, activation="relu"))
m3.add(tf.keras.layers.Dense(35, activation="relu"))
m3.add(tf.keras.layers.Dense(46, activation="relu"))
m3.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

Una vez entrenado, se grafica la matriz de confusión y se calculan las métricas pertinentes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
F2 Score: 1.0000
ROC AUC Score: 1.0000
```

Este nuevo modelo, compuesto por cuatro capas ocultas (8, 20, 35 y 46 neuronas) y una única neurona de salida, exhibe un desempeño impecable en el conjunto de entrenamiento: todas las métricas (exactitud, precisión, recall, F1, F2 y AUC) alcanzan 1.0. La matriz de confusión corrobora que no hay errores al clasificar “Setosa” y “No Setosa”. Sin embargo, es fundamental comprobar si estos resultados se mantienen fuera del conjunto de entrenamiento, por ejemplo con un conjunto de prueba o validación cruzada, para asegurarse de que el modelo no se limite a memorizar los datos y sea capaz de generalizar correctamente.

Comparación:

Los tres modelos difieren principalmente en la complejidad de sus arquitecturas y, en consecuencia, en el rendimiento que muestran en el conjunto de entrenamiento. El primer modelo (tres capas ocultas de 10, 15 y 20 neuronas) y el tercero (cuatro capas ocultas de 8, 20, 35 y 46 neuronas) logran métricas perfectas (Accuracy, Precision, Recall, F1, F2, ROC AUC todas en 1.0), lo que indica que separan por completo la clase

“Setosa” de “No Setosa” en el entrenamiento. Por el contrario, el segundo modelo (cinco capas ocultas de 5, 7, 21, 30 y 5 neuronas) obtiene un desempeño muy limitado: 66.67% de exactitud y valores nulos en precisión, recall, F1 y F2, lo que sugiere que no está reconociendo la clase positiva y clasifica todo como “No Setosa”.

En términos de interpretación, los modelos con métricas perfectas podrían estar sobreajustando (memorizando el conjunto de entrenamiento), especialmente si la separación de la clase Setosa es sencilla o si los datos son escasos. El segundo modelo, por su parte, parece tener problemas de convergencia o configuración, pues su arquitectura más profunda no se traduce en un mejor desempeño. Por ello, conviene validar los tres modelos con un conjunto de prueba o mediante validación cruzada para verificar su capacidad de generalización y descartar que el resultado perfecto sea producto de un sobreajuste.

2. Clasificación con redes neuronales - Datos heart

Pregunta 1. Selección del Modelo base

Selección de variables

Para realizar el modelo base de decidió que se usarían las siguientes variables:

- categóricas numericas: 'sex', 'exang', 'fbs'
- categórico texto: 'thal'
- numérica:s 'age', 'trestbps', 'chol'

```
df.drop(columns=['cp', 'restecg','ca','thalach', 'oldpeak', 'slope'])
```

Datos de entrenamiento

Posterior a la selección de variables se separan los datos en conjunto de entrenamiento, prueba y validación.

```
train = df.sample(frac=0.8, random_state=100)
```

Normalización y manejo de variables

Se construye un normalizador para las variables además de realizarle el tratamiento correspondiente a las variables dependiendo de su naturaleza.

Creación del modelo

Se crea el modelo, para ello se inicia creando una capa de entrada, concatenado todas las variables codificadas:

```
all_feats = keras.layers.concatenate(feats_encoded)
```

Luego, se agrega una capa densa de 32 neuronas:

```
model_layers = keras.layers.Dense(32, activation='relu')(all_feats)
```

Finalmente se crea una capa de salida, concatenado todas las variables codificadas:

```
model_layers = keras.layers.Dense(1, activation='sigmoid')(model_layers)
```

Con todo esto se crea el modelo y se compila con su optimizador (en este caso ADAM) y su función de perdida (en este caso entropía cruzada binaria):

```
model = keras.Model(inputs, model_layers)
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Pregunta 2. Entrenamiento y resultados

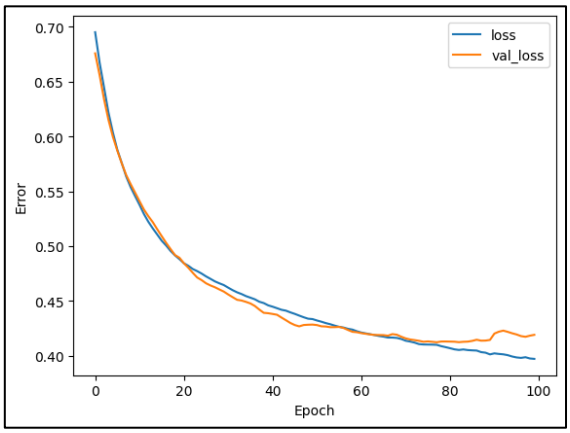
Entrenamiento

Se entrena el modelo con 100 épocas:

```
hostory = model.fit(train_ds, epochs=100, validation_data=val_ds)
```

Grafico del historial de perdidas

Una vez entrenado el modelo, se puede graficar el historial de perdidas:

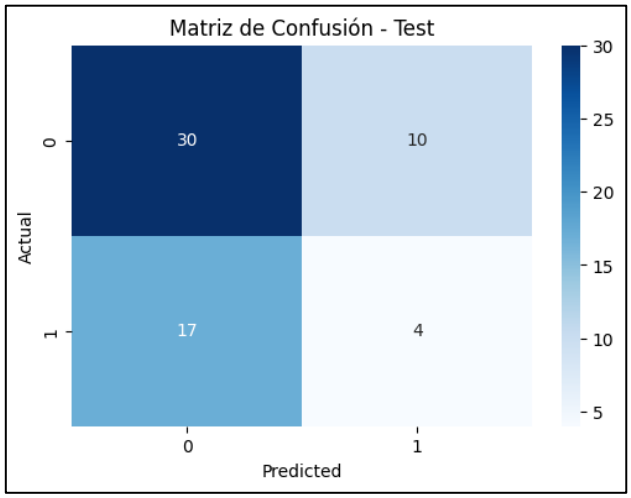


El gráfico refleja cómo la pérdida (loss) en entrenamiento y validación desciende de forma progresiva a lo largo de 100 épocas, lo que indica que el modelo está aprendiendo de manera adecuada. Ambas curvas se mantienen cercanas entre sí, lo cual sugiere que no hay un sobreajuste severo y que el modelo generaliza razonablemente bien. Hacia las últimas iteraciones, la curva de entrenamiento sigue bajando levemente, mientras que la de validación fluctúa un poco, una brecha normal que no representa un problema grave. En conjunto, el comportamiento de las curvas apunta a una convergencia estable,

aunque siempre cabe la posibilidad de afinar parámetros o aplicar regularización adicional para mejorar aún más el rendimiento.

Matriz de confusión y Métricas:

Finalmente, se puede graficar la matriz de confusión y calcular las métricas pertinentes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.5574
Precision: 0.2857
Recall: 0.1905
F1 Score: 0.2286
F2 Score: 0.2041
ROC AUC Score: 0.4702
```

La matriz de confusión muestra que, para la clase negativa (0), el modelo acierta en 30 de los 40 casos (TN = 30) y se equivoca en 10 (FP = 10), mientras que para la clase positiva (1) solo acierta en 4 de los 21 casos (TP = 4), dejando 17 ejemplos mal clasificados como negativos (FN = 17). Esta distribución se refleja en métricas de rendimiento bajas, pues la precisión (0.2857) y la sensibilidad (0.1905) evidencian la dificultad del modelo para identificar correctamente los positivos, mientras que el F1 Score (0.2286) y el F2 Score (0.2041) también confirman un desempeño limitado en la clasificación de la clase minoritaria. Además, la exactitud (0.5574) apenas supera la estrategia de predecir siempre la clase mayoritaria, y el ROC AUC (0.4702) sugiere que el poder de discriminación del modelo se aproxima al azar. En conjunto, estas cifras indican la necesidad de afinar los hiperparámetros, emplear técnicas de regularización o mejorar la ingeniería de características para equilibrar la detección de positivos y negativos.

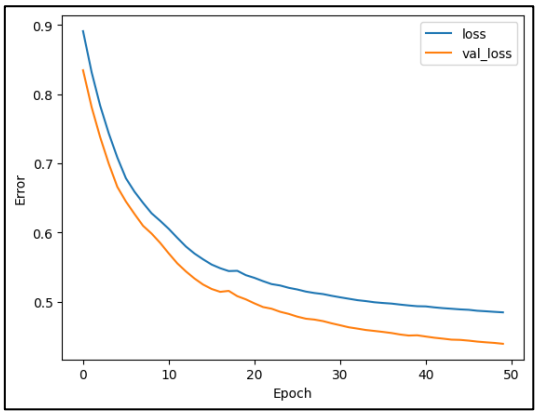
Pregunta 3. Optimizador

Primer modelo

La primera modificación que se realiza es usar como optimizador SGD:

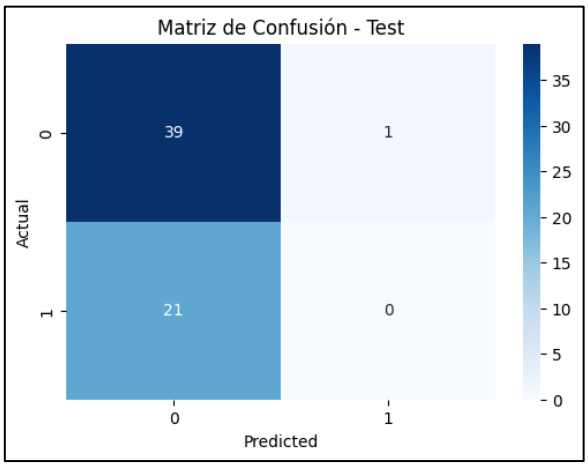
```
all_feats_opti_1 = keras.layers.concatenate(feats_encoded)
model_layers_opti_1 = keras.layers.Dense(32, activation='relu')(all_feats_opti_1)
model_layers_opti_1 = keras.layers.Dense(1, activation='sigmoid')(model_layers_opti_1)
model_opti_1 = keras.Model(inputs, model_layers_opti_1)

model_opti_1.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])
```

Este modelo se entrena con 50 épocas y se grafica su historial de perdidas:



La gráfica sugiere que, al usar SGD en lugar de Adam, el modelo experimenta un descenso más gradual tanto en la pérdida de entrenamiento como en la de validación. Aunque ambas curvas siguen una tendencia descendente a lo largo de las 50 épocas, se observa que la pérdida de entrenamiento (línea azul) va reduciéndose de manera progresiva, mientras que la de validación (línea naranja) se mantiene cercana y tiende a estabilizarse alrededor de valores en torno a 0.45–0.50. Este comportamiento es coherente con la naturaleza de SGD. Aun así, el resultado final no dista demasiado del que se logra con Adam, y en algunos casos el entrenamiento con SGD puede resultar más estable y menos propenso a “saltos” bruscos en la pérdida, aunque demande más tiempo para alcanzar un óptimo similar o ligeramente inferior.

Por último, se grafica la matriz de confusión y se calculan las métricas correspondientes:



```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6393
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
F2 Score: 0.0000
ROC AUC Score: 0.4875
```

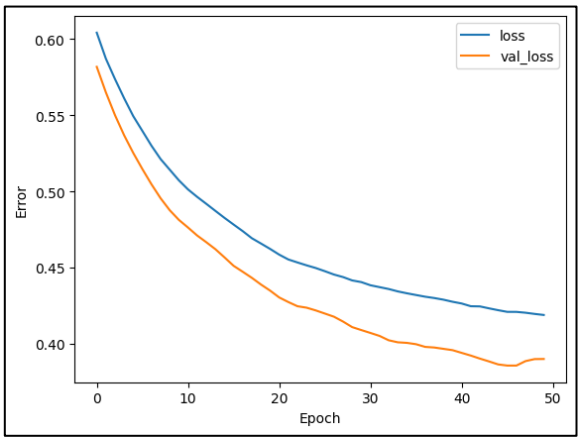
La matriz de confusión revela que el modelo, al usar SGD en lugar de Adam, acierta prácticamente todos los casos de la clase negativa (39 de 40), pero no logra detectar ningún caso de la clase positiva (0 verdaderos positivos y 21 falsos negativos). Esto se traduce en métricas de precisión, sensibilidad, F1 y F2 iguales a 0 para la clase positiva, y en un ROC AUC cercano al azar (0.4875). A pesar de que la exactitud global (0.6393) es mayor que la del modelo anterior, este valor se explica por la alta proporción de ejemplos negativos en el conjunto de prueba, de modo que el modelo se limita a predecir casi siempre la clase "0". En consecuencia, no hay una capacidad real de discriminación para la clase positiva, por lo que sería necesario ajustar los hiperparámetros, aplicar técnicas de regularización o balancear la clase minoritaria para mejorar la detección de positivos.

Segundo Modelo

Ahora, para el segundo modelo, se utiliza como optimizador adamw:

```python
all_feats_opti_2 = keras.layers.concatenate(feats_encoded)
model_layers_opti_2 = keras.layers.Dense(32, activation='relu')(all_feats_opti_2)
model_layers_opti_2 = keras.layers.Dense(1, activation='sigmoid')(model_layers_opti_2)
model_opti_2 = keras.Model(inputs, model_layers_opti_2)

model_opti_2.compile(optimizer='adamw', loss='binary_crossentropy', metrics=['accuracy'])
```

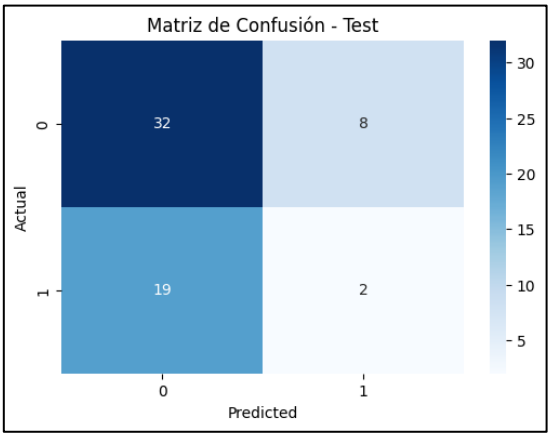
Este modelo se entrena con 50 épocas y se grafica su historial de perdidas:



La curva indica que, con AdamW como optimizador, tanto la pérdida de entrenamiento (línea azul) como la de validación (línea naranja) descienden de manera estable y sostenida a lo largo de las 50 épocas. Se observa que la pérdida de validación se mantiene por debajo de la de entrenamiento en la mayor parte del proceso, lo que sugiere una generalización razonable sin indicios marcados de sobreajuste. Además, la

tendencia descendente de ambas curvas apunta a que el modelo continúa aprendiendo de forma consistente, reflejando las ventajas que ofrece AdamW al desacoplar la tasa de aprendizaje y la regularización por peso (weight decay). En conjunto, la gráfica sugiere un proceso de entrenamiento efectivo y un mejor balance entre reducción de la pérdida y estabilidad en la validación, en comparación con las alternativas evaluadas anteriormente.

Finalmente, se grafica su matriz de confusión y calculan las métricas:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.5574
Precision: 0.2000
Recall: 0.0952
F1 Score: 0.1290
F2 Score: 0.1064
ROC AUC Score: 0.4476
```

La matriz de confusión muestra que el modelo acierta en 32 de los 40 casos negativos (verdaderos negativos) y en 2 de los 21 casos positivos (verdaderos positivos), dejando un total de 8 falsos positivos y 19 falsos negativos. Esto se refleja en una exactitud (Accuracy) de 0.5574, impulsada principalmente por la alta proporción de ejemplos negativos bien clasificados. Sin embargo, la sensibilidad (Recall) de 0.0952 indica que el modelo apenas reconoce el 9.52% de los casos positivos, mientras que la precisión (Precision) de 0.20 señala que, de todas las veces que se predice la clase positiva, solo el 20% es realmente positivo. En consecuencia, el F1 Score (0.1290) y el F2 Score (0.1064) se mantienen bajos, y el ROC AUC Score (0.4476) está por debajo del azar ideal (0.5), lo que en conjunto evidencia una capacidad de discriminación limitada para la clase positiva.

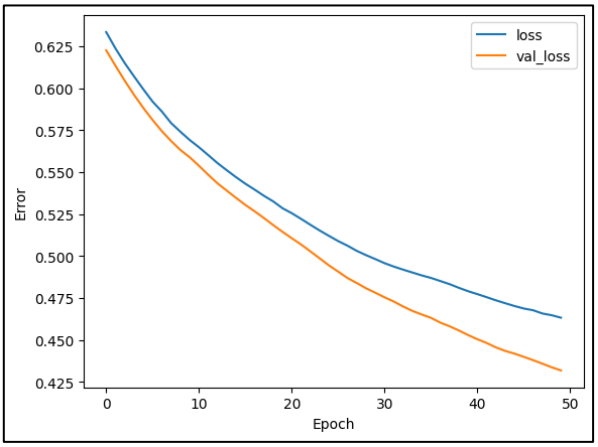
Pregunta 4. Tasa de aprendizaje

Primer modelo

Para el primer modelo se configura una taza de aprendizaje del 0,05%:

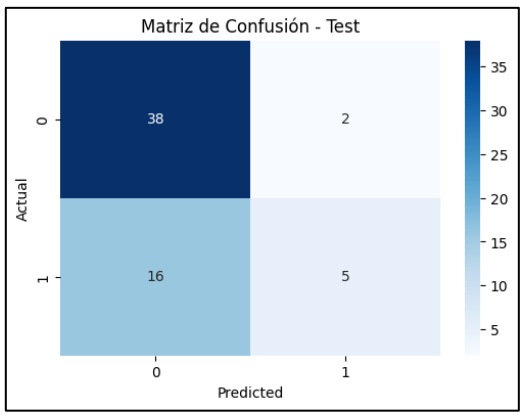
```python
all_feats_tasa_1 = keras.layers.concatenate(feats_encoded)
model_layers_tasa_1 = keras.layers.Dense(32, activation='relu')(all_feats_tasa_1)
model_layers_tasa_1 = keras.layers.Dense(1, activation='sigmoid')(model_layers_tasa_1)
model_tasa_1 = keras.Model(inputs, model_layers_tasa_1)

model_tasa_1.compile(optimizer=keras.optimizers.Adam(
    learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

El modelo se entrena con 50 épocas y se grafica el historial de perdidas:



La gráfica muestra que, con un optimizador Adam y una tasa de aprendizaje de 0.0005 (0.05%), tanto la pérdida de entrenamiento (curva azul) como la de validación (curva naranja) descienden de forma consistente a lo largo de las 50 épocas. Se aprecia una disminución gradual y estable, lo que sugiere que la tasa de aprendizaje no es excesivamente alta ni demasiado baja. Además, la curva de validación se mantiene por debajo de la de entrenamiento en la mayor parte del proceso, lo que indica una buena capacidad de generalización sin señales evidentes de sobreajuste. En conjunto, el modelo parece beneficiarse de una convergencia adecuada, al mismo tiempo que conserva un margen razonable entre ambas curvas, reflejando un entrenamiento equilibrado.

Finalmente, se grafica la matriz de confusión y se calculan las métricas correspondientes:



```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.7049
Precision: 0.7143
Recall: 0.2381
F1 Score: 0.3571
F2 Score: 0.2747
ROC AUC Score: 0.5940
```

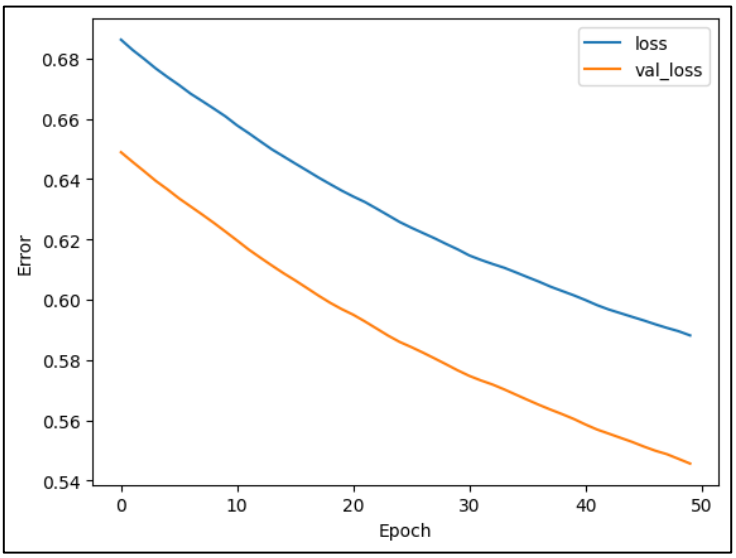
La matriz de confusión muestra que el modelo acierta en 38 de los 40 casos negativos (TN = 38) y en 5 de los 21 casos positivos (TP = 5), lo que se traduce en una exactitud (Accuracy) de 0.7049. Esta cifra está impulsada principalmente por el buen desempeño en la clase negativa, dado que la sensibilidad (Recall = 0.2381) es baja y refleja la dificultad del modelo para reconocer los casos positivos (16 falsos negativos). Sin embargo, la precisión (Precision = 0.7143) indica que, cuando el modelo predice la clase positiva, acierta en la mayoría de las ocasiones. Como consecuencia de esta baja sensibilidad, el F1 Score (0.3571) y el F2 Score (0.2747) son modestos, evidenciando un desempeño limitado en la identificación de la clase minoritaria. Finalmente, el ROC AUC (0.5940) muestra que el poder de discriminación del modelo supera levemente el azar, pero aún está lejos de un valor óptimo.

Segundo modelo

Para el segundo modelo, se utiliza una tasa del 0,01%:

```python
all_feats_tasa_2 = keras.layers.concatenate(feats_encoded)
model_layers_tasa_2 = keras.layers.Dense(32, activation='relu')(all_feats_tasa_2)
model_layers_tasa_2 = keras.layers.Dense(1, activation='sigmoid')(model_layers_tasa_2)
model_tasa_2 = keras.Model(inputs, model_layers_tasa_2)

model_tasa_2.compile(optimizer=keras.optimizers.Adam(
    learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

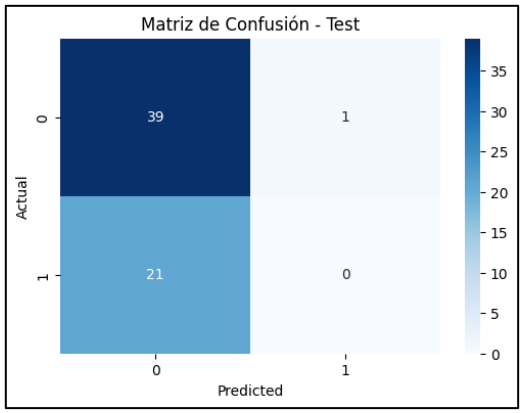
Entrenado también con 100 épocas. la gráfica de perdidas es la siguiente:



La gráfica muestra que, con un optimizador Adam y una tasa de aprendizaje de 0.0001 (0.01%), tanto la pérdida de entrenamiento (línea azul) como la de validación (línea

naranja) descienden de manera estable, pero a un ritmo más moderado que en configuraciones con una tasa de aprendizaje más alta. La curva de validación se mantiene por debajo de la de entrenamiento en la mayor parte de las 50 épocas, lo que sugiere que el modelo no está sobreajustando de forma notable y que generaliza razonablemente bien. Sin embargo, el ritmo de convergencia parece lento, lo cual es típico de tasas de aprendizaje muy bajas. En este escenario, es posible que el modelo requiera más épocas para alcanzar una pérdida más reducida, o bien un ajuste adicional de hiperparámetros (por ejemplo, aumentar la tasa de aprendizaje ligeramente o modificar la arquitectura) para lograr una mejora más acelerada.

Finalmente, su matriz de confusión y métricas es:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6393
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
F2 Score: 0.0000
ROC AUC Score: 0.4875
```

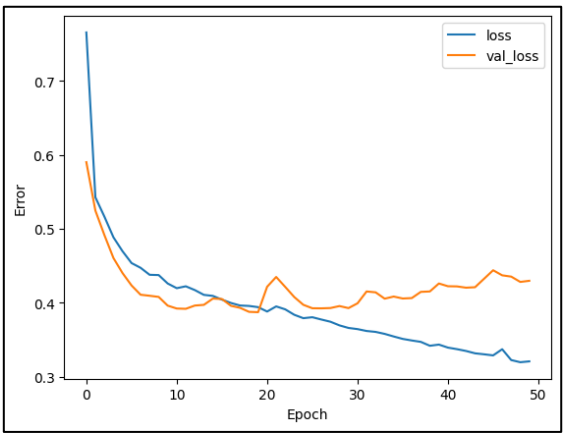
La matriz de confusión muestra que el modelo acierta en 39 de los 40 casos negativos, pero no identifica ninguno de los 21 casos positivos, resultando en precisión y sensibilidad nulas (0.0000). Esta situación deriva en un F1 Score, F2 Score y ROC AUC muy bajos (0.0000 y 0.4875, respectivamente), lo que evidencia la incapacidad del modelo para discriminar la clase positiva. A pesar de que la exactitud (0.6393) podría parecer aceptable, se debe principalmente a la alta proporción de ejemplos negativos, de modo que el modelo prácticamente siempre predice la clase “0”.

Tercer modelo

El último modelo corresponde a uno con tasa de aprendizaje del 0.7%:

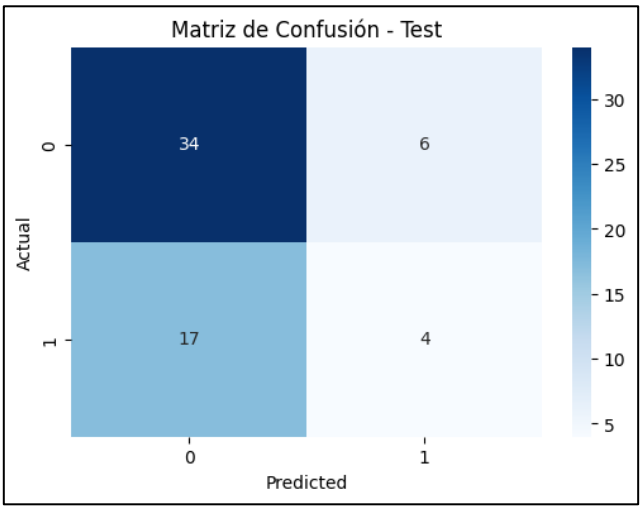
```
all_feats_tasa_3 = keras.layers.concatenate(feats_encoded)
model_layers_tasa_3 = keras.layers.Dense(32, activation='relu')(all_feats_tasa_3)
model_layers_tasa_3 = keras.layers.Dense(1, activation='sigmoid')(model_layers_tasa_3)
model_tasa_3 = keras.Model(inputs, model_layers_tasa_3)

model_tasa_3.compile(optimizer=keras.optimizers.Adam(
    learning_rate=0.007), loss='binary_crossentropy', metrics=['accuracy'])
```

Entrenado con 50 épocas su grafica es la siguiente:



La gráfica evidencia que, con una tasa de aprendizaje del 0.007 (0.7%), la pérdida de entrenamiento (línea azul) desciende con rapidez en las primeras épocas, pero la curva de validación (línea naranja) presenta fluctuaciones más pronunciadas, especialmente hacia la mitad del proceso (alrededor de la época 20). Este comportamiento sugiere que el modelo está aprendiendo con rapidez, pero a costa de cierta inestabilidad en la generalización, como se refleja en los picos de la pérdida de validación. Aunque la pérdida de entrenamiento continúa descendiendo y alcanza valores alrededor de 0.3, la de validación se estabiliza por encima de 0.45, lo cual indica un riesgo de sobreajuste o, al menos, una menor capacidad de adaptación a los datos de validación

Finalmente, su matriz de confusión y métricas son las siguientes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6230
Precision: 0.4000
Recall: 0.1905
F1 Score: 0.2581
F2 Score: 0.2128
ROC AUC Score: 0.5202
```

La matriz de confusión revela que el modelo acierta en 34 de los 40 casos negativos y en 4 de los 21 casos positivos, lo que se traduce en una exactitud (Accuracy) de 0.6230. Sin embargo, la sensibilidad (Recall) de 0.1905 indica que apenas se identifican correctamente el 19.05% de los positivos, mientras que la precisión (Precision) de 0.4000 refleja que, de todas las veces que se predice la clase "1", solo el 40% corresponde realmente a casos positivos. Como consecuencia, el F1 Score (0.2581) y el F2 Score (0.2128) permanecen en valores bajos, y el ROC AUC Score (0.5202) sugiere una capacidad de discriminación apenas superior al azar. En conjunto, aunque la exactitud global parezca aceptable, el modelo continúa mostrando limitaciones importantes para identificar la clase minoritaria.

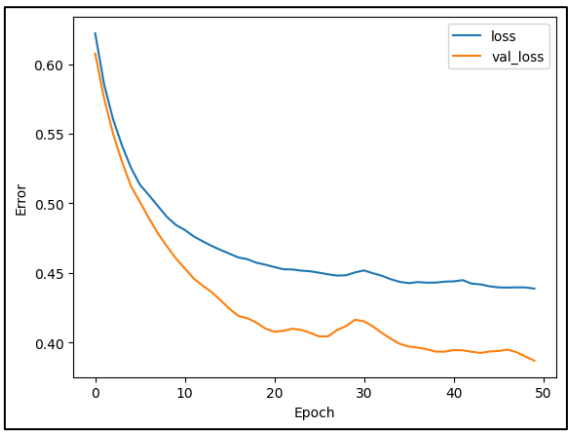
Pregunta 5. Función de activación

Primer modelo

Para el primer modelo se utiliza como función de activación la función celu:

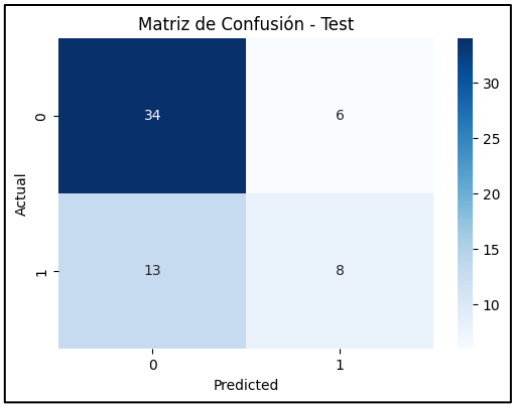
```
all_feats_fun_1 = keras.layers.concatenate(feats_encoded)
model_layers_fun_1 = keras.layers.Dense(32, activation='celu')(all_feats_fun_1)
model_layers_fun_1 = keras.layers.Dense(1, activation='sigmoid')(model_layers_fun_1)
model_fun_1 = keras.Model(inputs, model_layers_fun_1)

model_fun_1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Entrenando el modelo con 50 épocas se procede a graficar el historial de perdidas:



La gráfica muestra que, con Adam como optimizador y CELU como activación en la capa oculta, tanto la pérdida de entrenamiento como la de validación descienden de forma estable. La validación (curva naranja) se mantiene por debajo de la de entrenamiento (curva azul) en varias etapas, lo que indica buena generalización y ausencia de sobreajuste marcado. En conjunto, el uso de CELU ofrece una convergencia equilibrada y un rendimiento consistente en validación.

La matriz de confusión y métricas del modelo son las siguientes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6885
Precision: 0.5714
Recall: 0.3810
F1 Score: 0.4571
F2 Score: 0.4082
ROC AUC Score: 0.6155
```

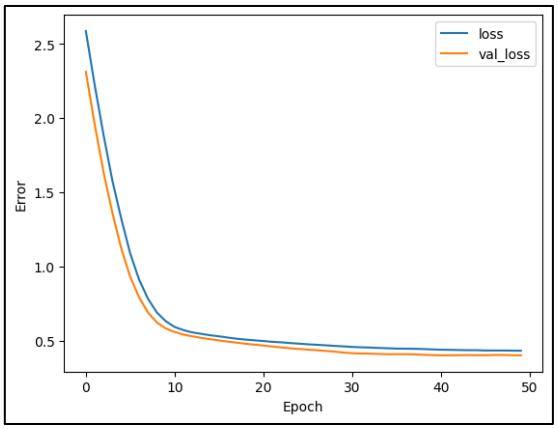
La matriz de confusión muestra que el modelo identifica 34 de los 40 negativos y 8 de los 21 positivos, lo que se traduce en una precisión (0.5714) razonable pero una sensibilidad (0.3810) aún limitada. La exactitud (0.6885) refleja principalmente el buen desempeño en la clase negativa, mientras que el F1 Score (0.4571) y el F2 Score (0.4082) indican un equilibrio mejorado entre precisión y recall respecto a intentos anteriores. Por último, el ROC AUC (0.6155) revela una capacidad de discriminación superior al azar, aunque todavía con margen para refinar la detección de la clase positiva.

Segundo modelo

Para el segundo modelo se utiliza como función de activación la función exponencial:

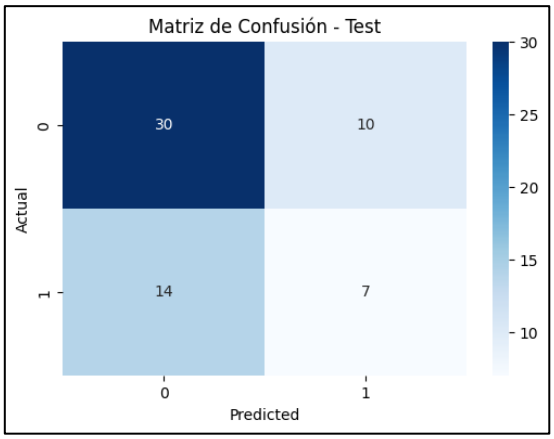
```
all_feats_fun_2 = keras.layers.concatenate(feats_encoded)
model_layers_fun_2 = keras.layers.Dense(32, activation='exponential')(all_feats_fun_2)
model_layers_fun_2 = keras.layers.Dense(1, activation='sigmoid')(model_layers_fun_2)
model_fun_2 = keras.Model(inputs, model_layers_fun_2)

model_fun_2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Entrenando el modelo con 50 épocas se procede a graficar el historial de perdidas:



La gráfica muestra que, con la función exponencial como activación en la capa oculta, tanto la pérdida de entrenamiento (línea azul) como la de validación (línea naranja) descienden rápidamente al principio y se estabilizan alrededor de 0.5. Este comportamiento sugiere un aprendizaje estable sin indicios claros de sobreajuste, pues ambas curvas se mantienen relativamente cercanas. La marcada caída inicial podría deberse a la naturaleza de la función exponencial, que tiende a amplificar valores si las ponderaciones son grandes al inicio, pero finalmente el modelo logra converger de forma razonable y consistente.

La matriz de confusión y métricas del modelo son las siguientes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6066
Precision: 0.4118
Recall: 0.3333
F1 Score: 0.3684
F2 Score: 0.3465
ROC AUC Score: 0.5417
```

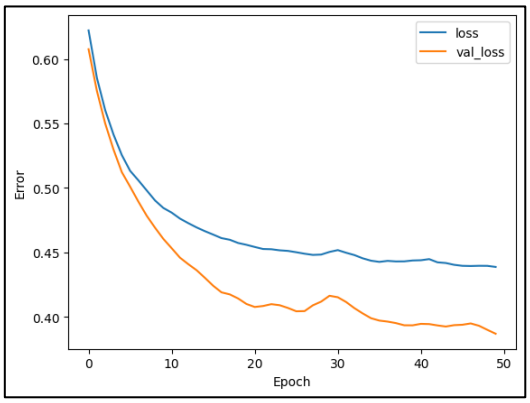
La matriz de confusión indica que el modelo clasifica correctamente 30 de los 40 negativos, pero solo 7 de los 21 positivos, lo cual explica la exactitud de 0.6066, impulsada sobre todo por la clase negativa. La precisión (0.4118) y la sensibilidad (0.3333) reflejan un desequilibrio en la detección de la clase positiva, con 10 falsos positivos y 14 falsos negativos. En consecuencia, el F1 Score (0.3684) y el F2 Score (0.3465) se mantienen bajos, mientras que el ROC AUC (0.5417) indica un poder de discriminación ligeramente superior al azar, pero aún con amplio margen de mejora en la identificación de positivos.

Tercer modelo

Para el tercer modelo se utiliza como función de activación la función hard tanh:

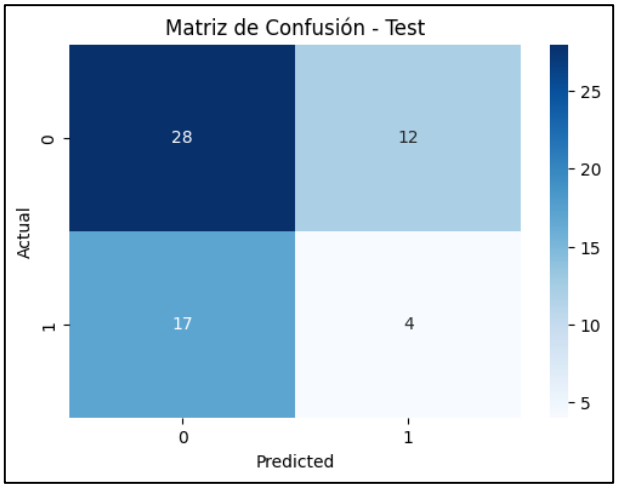
```
all_feats_fun_3 = keras.layers.concatenate(feats_encoded)
model_layers_fun_3 = keras.layers.Dense(32, activation='hard_tanh')(all_feats_fun_3)
model_layers_fun_3 = keras.layers.Dense(1, activation='sigmoid')(model_layers_fun_3)
model_fun_3 = keras.Model(inputs, model_layers_fun_3)

model_fun_3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Entrenando el modelo con 50 épocas se procede a graficar el historial de perdidas:



La gráfica muestra que, con la función de activación HardTanh, la pérdida de entrenamiento (azul) y la de validación (naranja) descienden de forma consistente. La curva de validación se sitúa por debajo de la de entrenamiento durante buena parte del proceso, lo que sugiere que el modelo generaliza sin signos evidentes de sobreajuste. Hacia las últimas épocas, ambas curvas parecen estabilizarse, con la validación en torno a 0.40 y la de entrenamiento ligeramente por encima de 0.45, lo que indica un aprendizaje razonablemente estable.

La matriz de confusión y métricas del modelo son las siguientes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.5246
Precision: 0.2500
Recall: 0.1905
F1 Score: 0.2162
F2 Score: 0.2000
ROC AUC Score: 0.4452
```

La matriz de confusión indica que el modelo identifica 28 de los 40 casos negativos, pero solo 4 de los 21 positivos, reflejando una precisión (0.25) y una sensibilidad (0.1905) bajas. Como consecuencia, el F1 Score (0.2162) y el F2 Score (0.20) también se mantienen en valores reducidos, y el ROC AUC (0.4452) está por debajo de 0.5, lo cual sugiere que el modelo no discrimina de forma fiable entre las dos clases. Aunque la exactitud (0.5246) supera levemente el 50%, se debe principalmente al mayor número de negativos correctos, sin un verdadero equilibrio en la detección de la clase positiva.

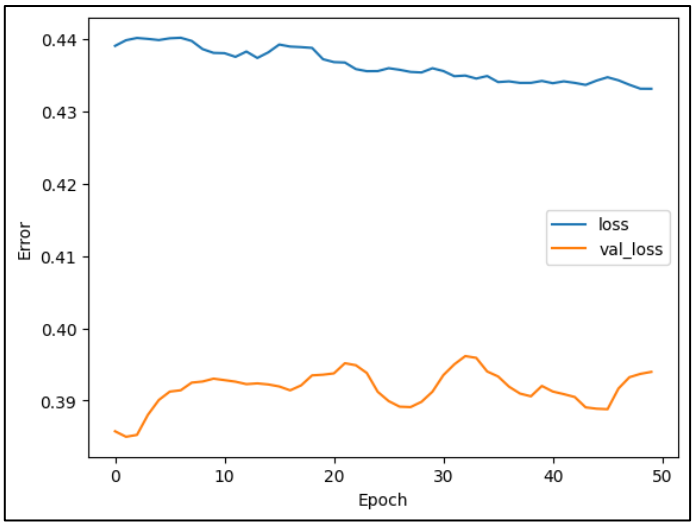
Pregunta 6. Modelos finales

Primer modelo

Para el primer modelo se utiliza como función de activación la función celu, una tasa de aprendizaje de 0.7% y como optimizador adamw:

```
all_feats_fin_1 = keras.layers.concatenate(feats_encoded)
model_layers_fin_1 = keras.layers.Dense(32, activation='celu')(all_feats_fin_1)
model_layers_fin_1 = keras.layers.Dense(1, activation='sigmoid')(model_layers_fin_1)
model_fin_1 = keras.Model(inputs, model_layers_fun_1)

model_fin_1.compile(optimizer=keras.optimizers.AdamW(
    learning_rate=0.007), loss='binary_crossentropy', metrics=['accuracy'])
```

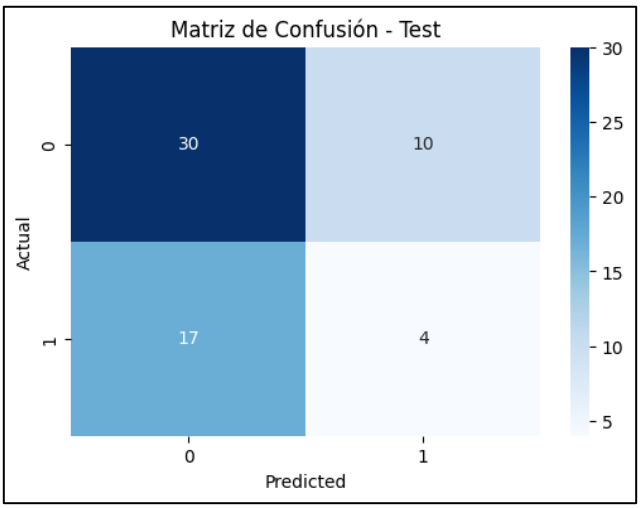
Entrenando el modelo con 50 épocas se procede a graficar el historial de perdidas:



La pérdida de entrenamiento se mantiene cercana a 0.44 con ligeras variaciones, mientras la de validación ronda 0.39 y presenta fluctuaciones igualmente moderadas, lo que sugiere que, con AdamW, una tasa de aprendizaje de 0.7% y CELU como activación, el modelo se encuentra en un punto de convergencia parcial sin indicios claros de sobreajuste.

No obstante, la falta de un descenso más pronunciado en la pérdida de entrenamiento podría indicar que la tasa de aprendizaje es algo elevada, o que se requieren ajustes adicionales en la arquitectura o en los hiperparámetros para lograr una optimización más profunda y un mejor rendimiento global.

La matriz de confusión y métricas del modelo son las siguientes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.5574
Precision: 0.2857
Recall: 0.1905
F1 Score: 0.2286
F2 Score: 0.2041
ROC AUC Score: 0.4702
```

La matriz de confusión indica que el modelo clasifica correctamente 30 de los 40 casos negativos, pero solo 4 de los 21 positivos, lo que provoca una sensibilidad (0.1905) y una precisión (0.2857) bajas para la clase minoritaria.

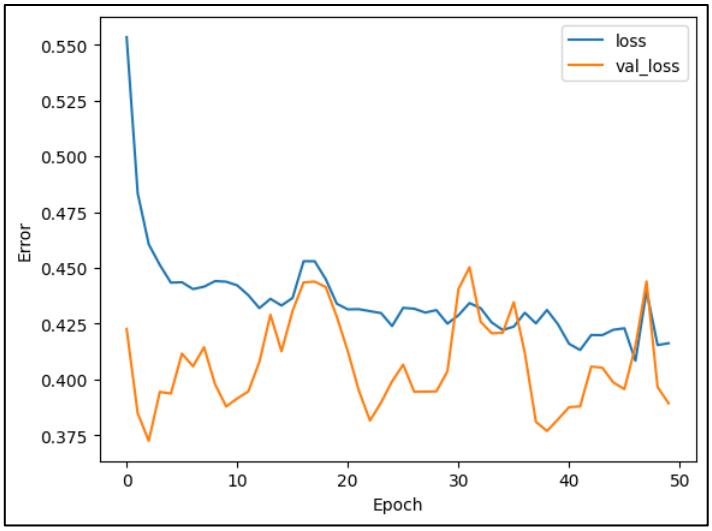
En consecuencia, el F1 Score (0.2286) y el F2 Score (0.2041) son igualmente reducidos, y aunque la exactitud (0.5574) supera apenas el escenario de predecir siempre la clase mayoritaria, el ROC AUC (0.4702) revela un poder de discriminación por debajo del deseable.

Segundo modelo

Para el último modelo se utiliza como como función de activación la función relu, una tasa de aprendizaje de 1% y como optimizador adam:

```
all_feats_fin_2 = keras.layers.concatenate(feats_encoded)
model_layers_fin_2 = keras.layers.Dense(32, activation='celu')(all_feats_fin_2)
model_layers_fin_2 = keras.layers.Dense(1, activation='sigmoid')(model_layers_fin_2)
model_fin_2 = keras.Model(inputs, model_layers_fin_2)

model_fin_2.compile(optimizer=keras.optimizers.Adam(
    learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

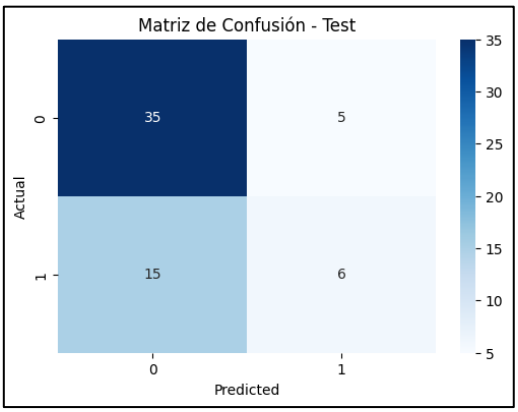
Entrenando el modelo con 50 épocas se procede a graficar el historial de perdidas:



La gráfica evidencia fluctuaciones considerables tanto en la pérdida de entrenamiento (línea azul) como en la de validación (línea naranja), lo que sugiere una convergencia inestable al combinar ReLU, Adam y una tasa de aprendizaje de 1%.

Aunque ambas curvas descienden en términos generales, los picos y valles indican que el modelo podría beneficiarse de reducir la tasa de aprendizaje o incorporar más regularización para suavizar la convergencia y alcanzar un rendimiento más consistente.

La matriz de confusión y métricas del modelo son las siguientes:




```
--- MÉTRICAS DEL MODELO (Entrenamiento) ---
Accuracy: 0.6721
Precision: 0.5455
Recall: 0.2857
F1 Score: 0.3750
F2 Score: 0.3158
ROC AUC Score: 0.5804
```

La matriz de confusión revela que el modelo identifica 35 de los 40 casos negativos y 6 de los 21 positivos, con 5 falsos positivos y 15 falsos negativos. Como resultado, la exactitud (0.6721) supera ligeramente el 67%, y la precisión (0.5455) es razonable, pero el recall (0.2857) sigue siendo limitado. En consecuencia, el F1 Score

(0.3750) y el F2 Score (0.3158) se mantienen moderados, mientras que el ROC AUC (0.5804) muestra un poder de discriminación algo superior al azar, aunque con margen de mejora para la clase positiva.