UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3092 - Reporte de Laboratorio

Sección 21

Ing. Javier Josué Fong Guzmán

Laboratorio 2

Luis Pedro González Aldana

Rebecca Maria Smith Martínez

GUATEMALA, 28 de Julio de 2023

Descripción de la red

Las variables más relevantes que describen una película son su duración, director, rating promedio, índice de aprobación, costo de producción y su ganancia. La creación de una red sobre este dataset estará orientada a la predicción de la rentabilidad de la película.

Antes de poder usar este dataset para entrenar la red neuronal, deberá pasar por un proceso de preprocesamiento.

El objetivo de la red neuronal será predecir si una película será rentable (**clasificación**). La rentabilidad se calculará restando el costo de producción del presupuesto de la película. Si el resultado es mayor que 0, la película se clasificará como "rentable". Si es menor o igual a 0, la película no se clasificará como rentable.

Este enfoque simplifica el problema y nos permite predecir directamente el objetivo de interés, que es la rentabilidad.

Después de completar este preprocesamiento, el dataset estará listo para ser ingresado a la red neuronal. Se pondrá especial atención a las siguientes variables:

- Género
- Director
- Duración
- Rating promedio
- Inversión (costo de producción)

Es importante señalar que esta simplificación no nos permitirá capturar la "calidad" de una película de la misma manera que la medida original de "éxito". Sin embargo, proporcionará una medida más directa y fácilmente interpretable de la viabilidad financiera de una película, que puede ser particularmente útil para los productores y las personas que invierten en la producción de películas

Red neuronal 1

Tabla 1: Características de primera red neuronal

Capas internas*	4
Técnica de regularización	Normalización de lotes y validación (Implícita)
Tamaño de tanda	40
Optimizador	Adam

Épocas	200

*Omitiendo las capas de normalización que preceden las demás capas internas con procesamiento

Tabla 2: Característica de las capas de primera red neuronal

Capa	Número de neuronas	Función de activación
Entrada	64	relu
Interna 1	128	relu
Interna 2	128	relu
Interna 3	64	relu
Interna 4	64	relu
Salida	2	softmax

Tabla 3: Resultados de la primera red neuronal

Tiempo de ejecución (s)	54.9
Accuracy_validation	0.72
Loss_validation	0.5813
Accuracy_evualation	0.72
Loss_evaluation	0.5844

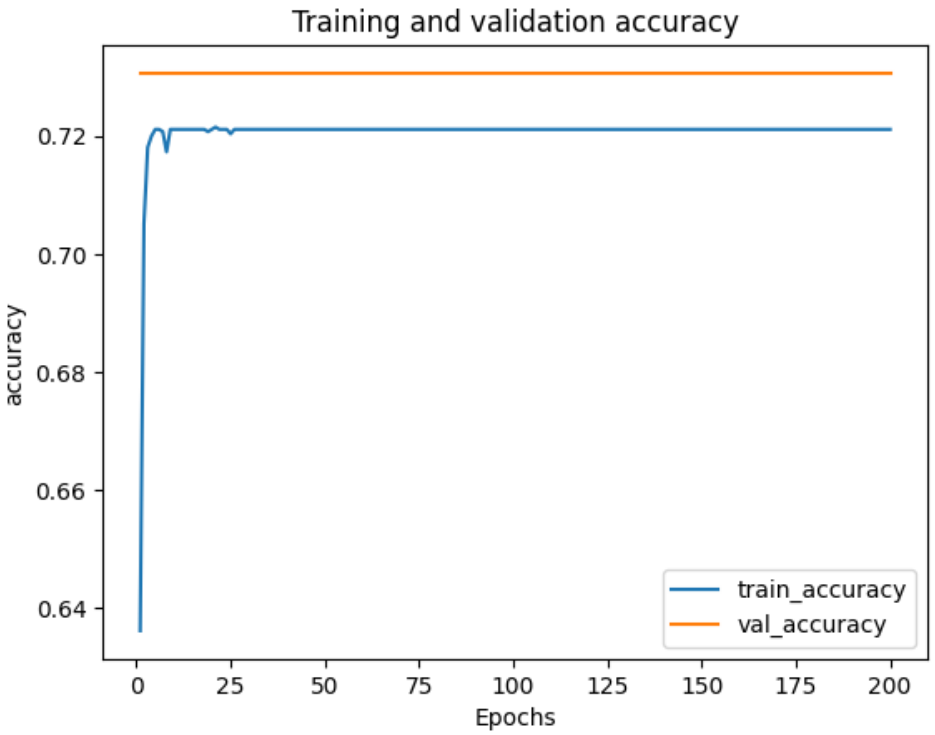


Figura 1: Comparación de accuracy de validación y entrenamiento a lo largo de las épocas de la red neuronal 1.

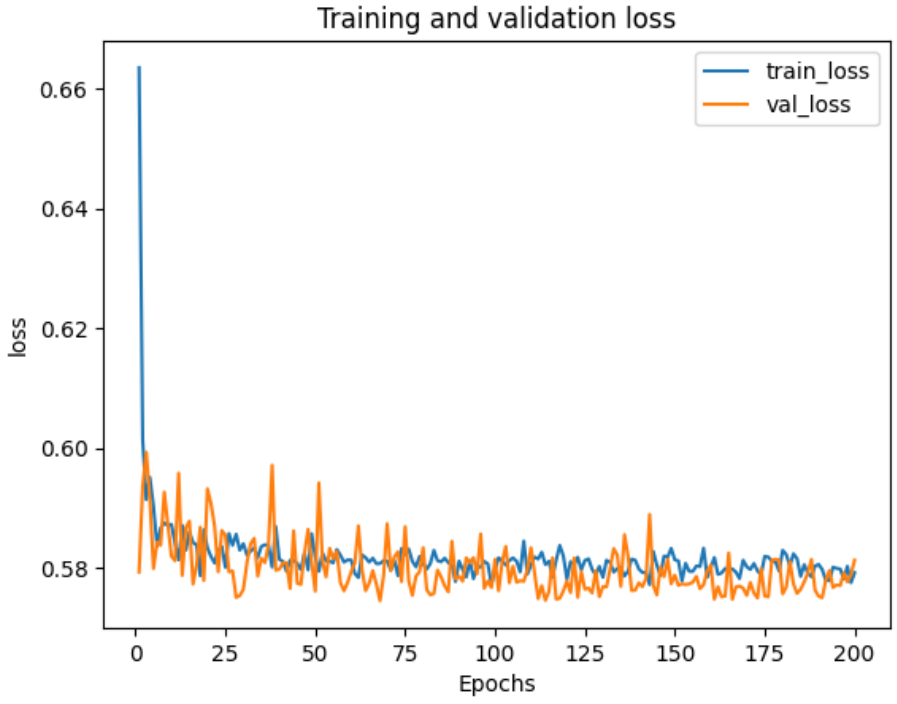


Figura 2: Comparación de loss de validación y entrenamiento a lo largo de las épocas de la red neuronal 1.

Las configuraciones de esta red se hicieron como punto de partida. Se utilizaron capas densas en función de aprender patrones complejos como lo puede ser el estar pendiente de tantos valores como lo del género de una película, que para este dataset resultó bastante extenso. El uso de Adam como

optimizador es considerado una buena práctica para la mayoría de redes dada su robustez para para los cambios de parámetros. La normalización de lotes, capa tras capa, proporciona regularización que puede reducir el sobreajuste. Finalmente, el uso de ReLu es particularmente útil y recomendado para los problemas asociados con el gradiente de descenso.

En cuanto a los resultados, tenemos un rendimiento medio que parece adaptarse igual de bien los datos de entrenamiento con los de evaluación. Su consiste 72% como valor de precisión parece un buen punto de partida que parece indicar que si existe una correlación importante entre los valores de entrada y la clasificación esperada. El tiempo de entrenamiento está dentro de los límites considerables por lo que es un red cuando menos considerablemente buena, no obstante, aún quedan aspectos de pérdida pendientes de discusión que la podrían hacer no viable . Aún con todo ello está un poco lejos de ser una predicción confiable, sobre todo con un tema tan sensible como es la rentabilidad de una película.

En la figura 1 se puede observar la curva de entrenamiento y validación. Esta gráfica hace que podamos ver de manera matemática como se representa el incremento de aprendizaje en el tiempo. La exactitud de entreno en la figura 1 nos demuestra que el modelo comienza aprendiendo poco y tiene un gran pico poco después. El modelo fue entrenado muy bien y el resultado lo miramos en la exactitud de validación el cual muestra que el modelo entrenado tuvo un buen resultado.

En la figura 2 tenemos una gráfica llamada “Overfit” la cual nos indica que el modelo aprendió bastante bien el data set. El modelo no podrá dar estimaciones exactas de la data que agregamos y que no es parte del dataset original, si agregamos datos, la validación decrece a cierto punto y luego volverá a crecer.

Red neuronal 2

Tabla 4: Características de segunda red neuronal

Capas internas*	5
Técnica de regularización	Dropout
Tamaño de tanda	40
Optimizador	Adam
Épocas	200

*Omitiendo las capas de dropout que preceden las demás capas internas con procesamiento

Tabla 5: Característica de las capas de segunda red neuronal

Capa	Número de neuronas	Función de activación
Entrada	128	LeakyReLU
Interna 1	256	LeakyReLU
Interna 2	256	LeakyReLU
Interna 3	128	tanh
Interna 4	128	tanh
Salida	2	softmax

Tabla 6: Resultados de la primera segunda neuronal

Tiempo de ejecución (s)	67.8
Accuracy_validation	0.7386
Loss_validation	0.5772
Accuracy_evualation	0.718
Loss_evaluation	0.5952

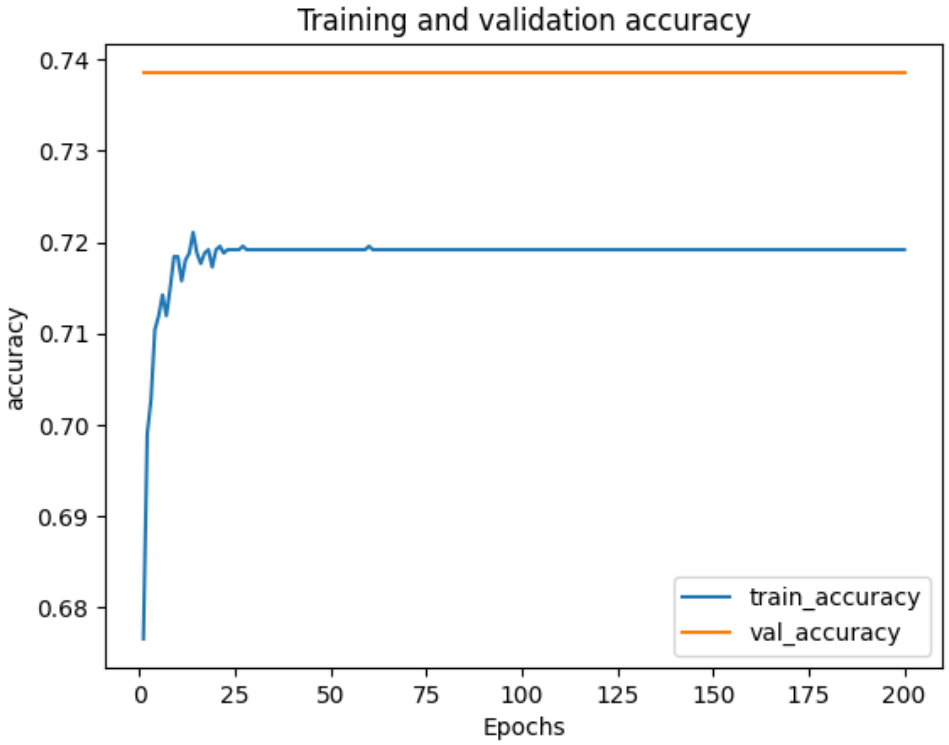
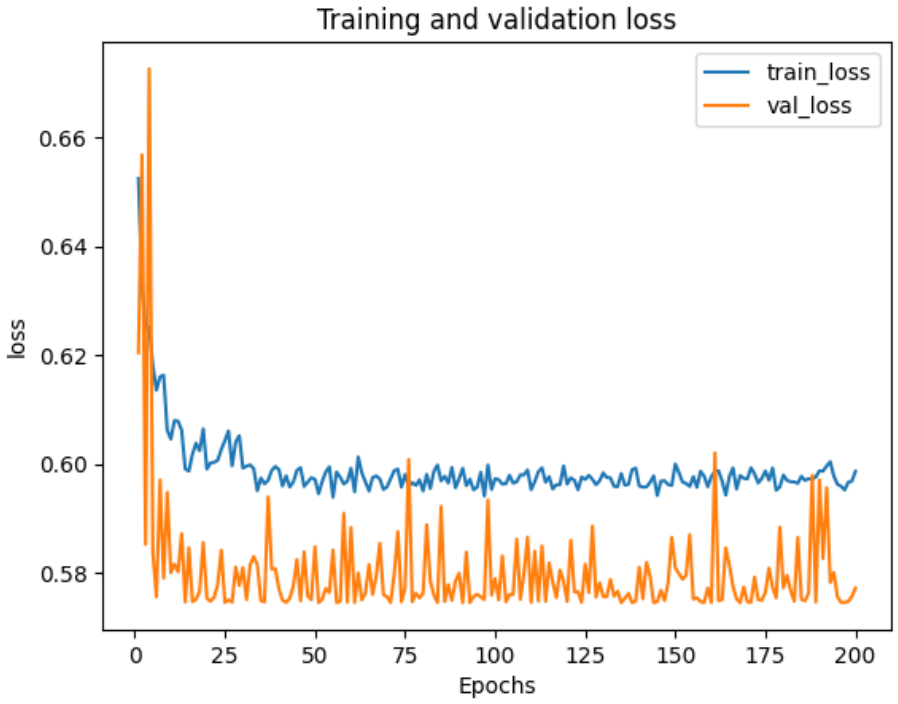


Figura 3: Comparación de accuracy de validación y entrenamiento a lo largo de las épocas de la red neuronal 2.



Comparación de loss de validación y entrenamiento a lo largo de las épocas de la red neuronal 2.

Se añadió la función de regularización de ***Dropout*** lo que puede ayudar a prevenir el sobreajuste. ***LeakyReLU*** es una variante de ReLU que permite pequeños valores negativos cuando la entrada es menor que cero, lo que puede ayudar a evitar las neuronas "muertas" (neuronas que siempre se activan

a cero). La función de activación ***tanh*** puede ofrecer mejores resultados para ciertos problemas, ya que devuelve valores entre -1 y 1 y por lo tanto puede manejar patrones negativos y positivos en los datos. Finalmente, se aumentó la profundidad de la red para permitirle aprender patrones más complejos.

En cuanto a los resultados, esta alternativa ha mejorado ligeramente el rendimiento de su predecesora. Sin embargo, el tiempo de ejecución ha aumentado lo que es natural dado el aumento en la complejidad del modelo (aumento de capas y uso de Dropout)

El Accuracy de validación ha aumentado de 0.72 a 0.7386, lo que indica que el modelo es capaz de predecir correctamente parte mayor de los ejemplos en el conjunto de validación, la pérdida (loss) de validación también ha disminuido, de 0.5813 a 0.5772, lo que generalmente indica que el modelo ha mejorado su rendimiento.; sugiriendo que los cambios realizados en la red han permitido que el modelo aprenda una representación más eficaz de los datos. Sin embargo, la Accuracy en la evaluación ha disminuido ligeramente de 0.72 a 0.718, y la pérdida de evaluación ha aumentado de 0.5844 a 0.5952. Un posible indicio sobre ajuste del modelo, ya que su rendimiento en el conjunto de validación es mayor que en el conjunto de evaluación.

En la figura 3 la exactitud de entrenamiento es casi igual a la exactitud de validación. Vemos que el modelo aprendió un poco más lento pero aún así la exactitud de validación es bastante exacta. Al compararla con la figura 1, vemos que son similares pero vemos que el modelo aún así aprendió un poco más lento comparado a la primera red.

En la figura 4 vemos un “Overfit” muy diferente al de la figura 2 pero aún así afecta el funcionamiento si queremos agregar más datos. En la figura 4, la pérdida en el entrenamiento es mayor pero la pérdida en la validación es bastante similar a la figura 2.

Red neuronal 3

Tabla 7: Características de tercera red neuronal

Capas internas*	1
Técnica de regularización	L2
Tamaño de tanda	20
Optimizador	Adam
Épocas	200

Tabla 8: Característica de las capas de tercera red neuronal

Capa	Número de neuronas	Función de activación
Entrada	32	relu
Interna 1	64	L2
Salida	2	softmax

Tabla 9: Resultados de la primera tercera neuronal

Tiempo de ejecución (s)	58.6
Accuracy_validation	0.7275
Loss_validation	0.7171
Accuracy_evualation	0.718
Loss_evaluation	0.7064

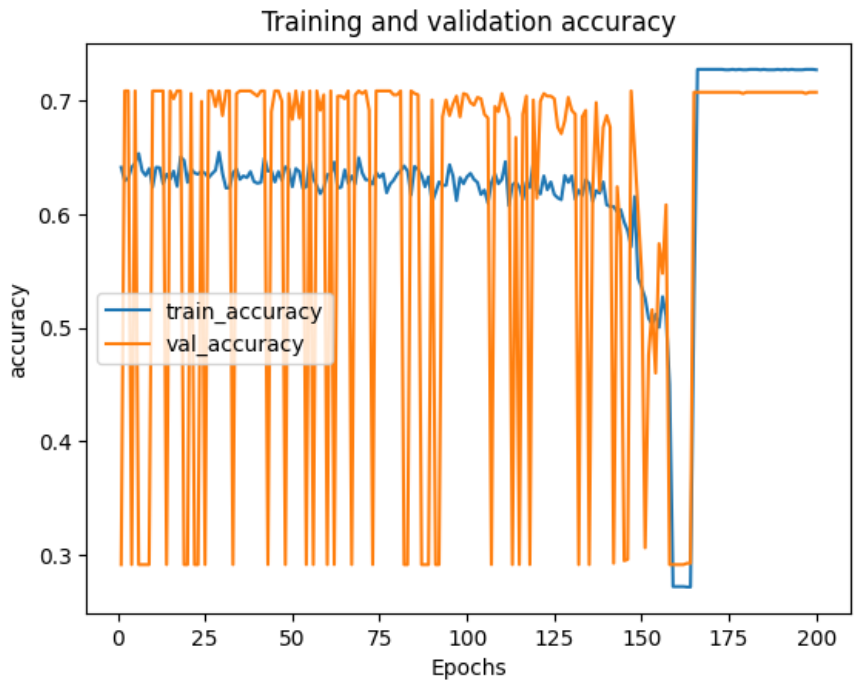


Figura 5: Comparación de accuracy de validación y entrenamiento a lo largo de las épocas de la red neuronal 3.

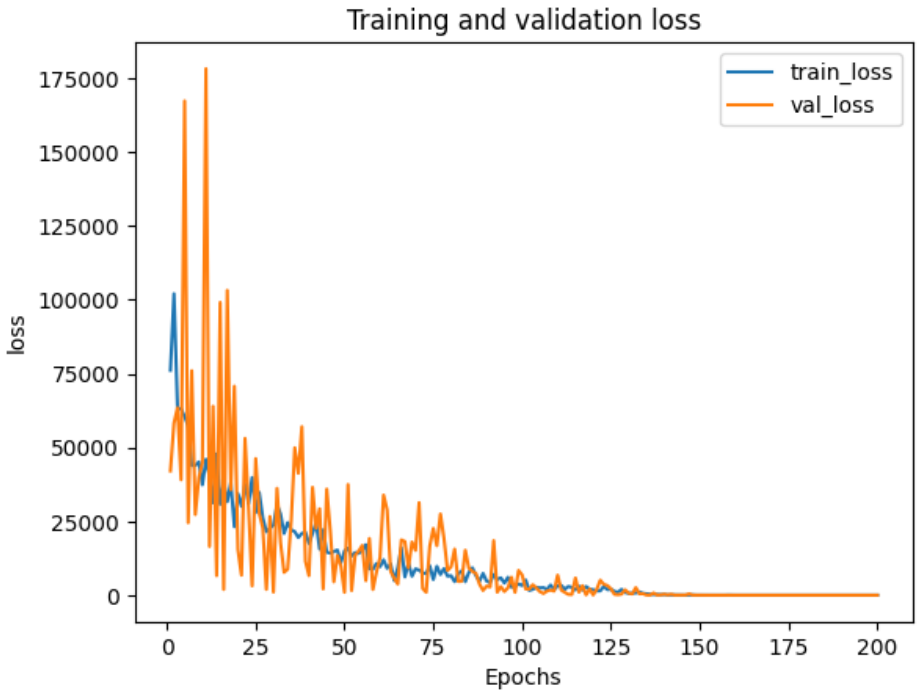


Figura 6: Comparación de loss de validación y entrenamiento a lo largo de las épocas de la red neuronal 3.

El uso de una regularización L2 castiga los pesos grandes, lo que puede ayudar a prevenir el sobreajuste. Se redujo el número de capas para prevenir este mismo problema de sobreajuste. Finalmente, un tamaño de lote más pequeño puede hacer que el entrenamiento sea más ruidoso y podría ayudar a evitar los mínimos locales. Un enfoque similar a la segunda red neuronal, orientada a reducir sobreajuste pero con técnicas distintas.

Se mejoró el tiempo de ejecución respecto al modelo anterior, derivado de tener menos capas y neuronas, y debido al cambio de la función de activación. En cuanto al accuracy de validación, este modelo presenta un valor de 0.7275, que es un poco más alto que el del modelo original (0.72) pero ligeramente más bajo que el de la primera red neuronal alternativa (0.7386). Un indicio que los cambios en esta red no fueron tan buenos como los de la segunda red neuronal.

La pérdida en la validación es 0.7171 es significativamente más alta que en los dos modelos anteriores indica la dificultad que tiene este en minimizar el error en sus predicciones en el conjunto de validación. En términos de accuracy en la evaluación, este modelo iguala a la primera red neuronal alternativa con un valor de 0.718. Esto es ligeramente inferior al del modelo original, pero la diferencia es pequeña.

Aunque no obtuvo tantas mejoras como las que se esperaban, los resultados siguen siendo razonablemente buenos y muestran que el modelo es capaz de predecir correctamente aproximadamente el 72% de los casos.

En la figura 5 vemos un aprendizaje variado. Comparado con la figura 1 y 3, este aprendizaje no tiene una curva si no una gran variación en su exactitud de variación mientras que el entreno es menos variado.. Al llegar al final vemos que la exactitud y validación se nivelan sin antes tener una exactitud bastante baja. El modelo tuvo mucha dificultad de aprendizaje pero lo logró.

En la figura 6 vemos que la gráfica tiene un “Good fit” el cuál nos indica que el modelo aprendió bien y tuvo resultados exitosos. Al compararla con las figuras 2 y 4 vemos la diferencia de pérdida de datos. En la figura 6 hubo pérdida pero llega a cero poco antes de finalizar.

Veredicto sobre la mejor red neuronal

Para evaluar cuál de las tres redes es la mejor, debemos considerar estos tres aspectos:

Accuracy

La primera red alternativa tiene la mayor precisión tanto en los datos de validación como en los de evaluación (0.7386 y 0.718 respectivamente). Esto indica que, de las tres redes, esta es la que mejor predice correctamente las clases en estos conjuntos de datos.

Loss

Aquí también la primera red alternativa gana, con los valores más bajos tanto en los datos de validación como en los de evaluación (0.5772 y 0.5952 respectivamente).

Tiempo de ejecución

La red original es la más rápida, seguida de la tercera red alternativa y luego la primera red alternativa. Sin embargo, las diferencias en el tiempo de ejecución son relativamente pequeñas, de manera que se dará más peso a los otros dos aspectos.

Considerando estos tres aspectos, **la segunda red neuronal** es la más efectiva de las tres en términos de rendimiento general. Aunque toma un poco más de tiempo para entrenar, ofrece la mayor precisión y la menor pérdida en los datos de validación y evaluación.

Anexos

Configuraciones y resultados en código de primera red neuronal (bn_movies_1.ipynb)

Capas de red

```
batched_model = Sequential([
    Dense(64, input_shape=(26,), activation="relu"),
    BatchNormalization(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dense(2, activation='softmax')
])
```

[16] ✓ 0.2s Python

Optimizador, función de pérdida y métricas

```
batched_model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

✓ 0.0s Python

Épocas, tamaño de batch

```
batched_history =  batched_model.fit(
    X_train,
    y_train,
    epochs=200,
    validation_split=0.25,
    batch_size=40,
    verbose=2
)
```

✓ 54.9s Python

Resultados

```
(function) verbose: Any
batched_model.evaluate(X_test, y_test, verbose=2)
```

✓ 0.1s Python

```
28/28 - 0s - loss: 0.5844 - accuracy: 0.7203 - 66ms/epoch - 2ms/step

[0.5844459533691406, 0.720319628715515]
```

```
Epoch 3/200
66/66 - 0s - loss: 0.5914 - accuracy: 0.7180 - val_loss: 0.5993 - val_accuracy: 0.7306 - 244ms/epoch - 4ms/step
Epoch 4/200
66/66 - 0s - loss: 0.5951 - accuracy: 0.7199 - val_loss: 0.5915 - val_accuracy: 0.7306 - 251ms/epoch - 4ms/step
Epoch 5/200
66/66 - 0s - loss: 0.5903 - accuracy: 0.7211 - val_loss: 0.5799 - val_accuracy: 0.7306 - 253ms/epoch - 4ms/step
Epoch 6/200
66/66 - 0s - loss: 0.5836 - accuracy: 0.7211 - val_loss: 0.5843 - val_accuracy: 0.7306 - 265ms/epoch - 4ms/step
Epoch 7/200
66/66 - 0s - loss: 0.5867 - accuracy: 0.7207 - val_loss: 0.5838 - val_accuracy: 0.7306 - 274ms/epoch - 4ms/step
Epoch 8/200
66/66 - 0s - loss: 0.5875 - accuracy: 0.7173 - val_loss: 0.5927 - val_accuracy: 0.7306 - 254ms/epoch - 4ms/step
Epoch 9/200
66/66 - 0s - loss: 0.5871 - accuracy: 0.7211 - val_loss: 0.5872 - val_accuracy: 0.7306 - 257ms/epoch - 4ms/step
Epoch 10/200
66/66 - 0s - loss: 0.5873 - accuracy: 0.7211 - val_loss: 0.5820 - val_accuracy: 0.7306 - 271ms/epoch - 4ms/step
Epoch 11/200
66/66 - 0s - loss: 0.5841 - accuracy: 0.7211 - val_loss: 0.5812 - val_accuracy: 0.7306 - 253ms/epoch - 4ms/step
Epoch 12/200
66/66 - 0s - loss: 0.5814 - accuracy: 0.7211 - val_loss: 0.5958 - val_accuracy: 0.7306 - 244ms/epoch - 4ms/step
Epoch 13/200
...
Epoch 199/200
66/66 - 0s - loss: 0.5775 - accuracy: 0.7211 - val_loss: 0.5789 - val_accuracy: 0.7306 - 271ms/epoch - 4ms/step
Epoch 200/200
66/66 - 0s - loss: 0.5792 - accuracy: 0.7211 - val_loss: 0.5813 - val_accuracy: 0.7306 - 279ms/epoch - 4ms/step
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Configuraciones y resultados en código de segunda red neuronal (bn_movies_2.ipynb)

Capas de red

```python
batched_model = Sequential([
    Dense(128, input_shape=(26,), activation="LeakyReLU"),
    Dropout(0.2),
    Dense(256, activation='LeakyReLU'),
    Dropout(0.2),
    Dense(256, activation='LeakyReLU'),
    Dropout(0.2),
    Dense(128, activation='tanh'),
    Dropout(0.2),
    Dense(128, activation='tanh'),
    Dense(2, activation='softmax')
])
```

✓ 0.1s Python

Optimizador y otros

```python
batched_model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

✓ 0.0s Python

Épocas, tamaño de batch

```python
batched_history =  batched_model.fit(
    X_train,
    y_train,
    epochs=200,
    validation_split=0.25,
    batch_size=40,
    verbose=2
)
```

[20] ✓ 1m 7.8s Python

Resultados

```
Epoch 3/200
66/66 - 0s - loss: 0.6247 - accuracy: 0.7028 - val_loss: 0.5852 - val_accuracy: 0.7386 - 338ms/epoch - 5ms/step
Epoch 4/200
66/66 - 0s - loss: 0.6252 - accuracy: 0.7104 - val_loss: 0.6726 - val_accuracy: 0.7386 - 319ms/epoch - 5ms/step
Epoch 5/200
66/66 - 0s - loss: 0.6181 - accuracy: 0.7119 - val_loss: 0.5838 - val_accuracy: 0.7386 - 325ms/epoch - 5ms/step
Epoch 6/200
66/66 - 0s - loss: 0.6135 - accuracy: 0.7142 - val_loss: 0.5756 - val_accuracy: 0.7386 - 309ms/epoch - 5ms/step
Epoch 7/200
66/66 - 0s - loss: 0.6160 - accuracy: 0.7119 - val_loss: 0.5971 - val_accuracy: 0.7386 - 331ms/epoch - 5ms/step
Epoch 8/200
66/66 - 0s - loss: 0.6163 - accuracy: 0.7150 - val_loss: 0.5790 - val_accuracy: 0.7386 - 319ms/epoch - 5ms/step
Epoch 9/200
66/66 - 0s - loss: 0.6062 - accuracy: 0.7184 - val_loss: 0.5948 - val_accuracy: 0.7386 - 326ms/epoch - 5ms/step
Epoch 10/200
66/66 - 0s - loss: 0.6046 - accuracy: 0.7184 - val_loss: 0.5800 - val_accuracy: 0.7386 - 310ms/epoch - 5ms/step
Epoch 11/200
66/66 - 0s - loss: 0.6080 - accuracy: 0.7158 - val_loss: 0.5816 - val_accuracy: 0.7386 - 306ms/epoch - 5ms/step
Epoch 12/200
66/66 - 0s - loss: 0.6079 - accuracy: 0.7180 - val_loss: 0.5802 - val_accuracy: 0.7386 - 305ms/epoch - 5ms/step
Epoch 13/200
...
Epoch 199/200
66/66 - 0s - loss: 0.5968 - accuracy: 0.7192 - val_loss: 0.5757 - val_accuracy: 0.7386 - 344ms/epoch - 5ms/step
Epoch 200/200
66/66 - 0s - loss: 0.5986 - accuracy: 0.7192 - val_loss: 0.5772 - val_accuracy: 0.7386 - 335ms/epoch - 5ms/step
```

```
batched_model.evaluate(X_test, y_test, verbose=2)
```

[24] ✓ 0.1s Python

```
28/28 - 0s - loss: 0.5952 - accuracy: 0.7180 - 73ms/epoch - 3ms/step

[0.5951686501502991, 0.7180365324020386]
```

Configuraciones y resultados en código de tercera red neuronal (bn_movies_3.ipynb)

Capas de red

```
batched_model = Sequential([
    Dense(32, input_shape=(26,), activation="relu"),
    Dense(64, activation="relu", kernel_regularizer='l2'),
    Dense(2, activation='softmax')
])
```

✓ 0.1s Python

Optimizador y otros

```
batched_model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

✓ 0.0s Python

Épocas, tamaño de batch

```python
batched_history =  batched_model.fit(
    X_train,
    y_train,
    epochs=200,
    validation_split=0.25,
    batch_size=20,
    verbose=2
)
```

✓ 58.6s Python

Resultados

```
132/132 - 0s - loss: 63127.0547 - accuracy: 0.6416 - val_loss: 39056.4570 - val_accuracy: 0.2911 - 274ms/epoch - 2ms/step
Epoch 5/200
132/132 - 0s - loss: 60442.5391 - accuracy: 0.6454 - val_loss: 167328.0000 - val_accuracy: 0.7089 - 268ms/epoch - 2ms/step
Epoch 6/200
132/132 - 0s - loss: 57349.8203 - accuracy: 0.6537 - val_loss: 24525.8203 - val_accuracy: 0.2911 - 269ms/epoch - 2ms/step
Epoch 7/200
132/132 - 0s - loss: 43911.0039 - accuracy: 0.6385 - val_loss: 75996.5859 - val_accuracy: 0.2911 - 267ms/epoch - 2ms/step
Epoch 8/200
132/132 - 0s - loss: 43903.9844 - accuracy: 0.6343 - val_loss: 27302.6758 - val_accuracy: 0.2911 - 268ms/epoch - 2ms/step
Epoch 9/200
132/132 - 0s - loss: 45116.4219 - accuracy: 0.6404 - val_loss: 39813.3086 - val_accuracy: 0.2911 - 264ms/epoch - 2ms/step
Epoch 10/200
132/132 - 0s - loss: 37364.7656 - accuracy: 0.6240 - val_loss: 44872.5312 - val_accuracy: 0.7089 - 263ms/epoch - 2ms/step
Epoch 11/200
132/132 - 0s - loss: 46017.8867 - accuracy: 0.6416 - val_loss: 178244.7500 - val_accuracy: 0.7089 - 264ms/epoch - 2ms/step
Epoch 12/200
132/132 - 0s - loss: 40171.7852 - accuracy: 0.6412 - val_loss: 16497.5840 - val_accuracy: 0.7089 - 281ms/epoch - 2ms/step
Epoch 13/200
132/132 - 0s - loss: 31289.8164 - accuracy: 0.6271 - val_loss: 63945.2344 - val_accuracy: 0.7089 - 268ms/epoch - 2ms/step
...
Epoch 199/200
132/132 - 0s - loss: 0.6992 - accuracy: 0.7279 - val_loss: 0.7188 - val_accuracy: 0.7078 - 267ms/epoch - 2ms/step
Epoch 200/200
132/132 - 0s - loss: 0.6986 - accuracy: 0.7275 - val_loss: 0.7171 - val_accuracy: 0.7078 - 256ms/epoch - 2ms/step
```

```python
batched_model.evaluate(X_test, y_test, verbose=2)
```

✓ 0.1s Python

```
28/28 - 0s - loss: 0.7064 - accuracy: 0.7180 - 59ms/epoch - 2ms/step

[0.7064318656921387, 0.7180365324020386]
```