Guillermo Furlán Estrada 20713
Diego Alonzo Medinilla 20172

Laboratorio 2-Deep Learning

Línea Base

- **Tamaño de tanda (batch size)**: 100
- **Tamaño de capa escondida:** 50
- **Tasa de aprendizaje (learning rate)**: 0.001 (Se usó ADAM y ese es el learning rate por defecto)
- **Precisión de validación**: 97.03%
- **Tiempo de entrenamiento**: 10 segundos
- **Pérdida de prueba:** 19.00.
- **Precisión de prueba:** 96.36%

```
Epoch 1/5
500/500 - 4s - 7ms/step - accuracy: 0.8761 - loss: 0.4358 - val_accuracy: 0.9444 - val_loss: 0.2058
Epoch 2/5
500/500 - 2s - 4ms/step - accuracy: 0.9436 - loss: 0.1922 - val_accuracy: 0.9561 - val_loss: 0.1550
Epoch 3/5
500/500 - 1s - 3ms/step - accuracy: 0.9571 - loss: 0.1475 - val_accuracy: 0.9613 - val_loss: 0.1319
Epoch 4/5
500/500 - 1s - 3ms/step - accuracy: 0.9654 - loss: 0.1177 - val_accuracy: 0.9656 - val_loss: 0.1175
Epoch 5/5
500/500 - 2s - 4ms/step - accuracy: 0.9702 - loss: 0.0991 - val_accuracy: 0.9703 - val_loss: 0.1085
<keras.src.callbacks.history.History at 0x7d7eda373730>
```

```
[113] perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
0s
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9562 - loss: 23.1316

[114] # Si se desea, se puede aplicar un formateo "bonito"
0s
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida_prueba, precision_prue

Pérdida de prueba: 19.00. Precisión de prueba: 96.36%
```

1. El ancho (tamaño de la capa escondida) del algoritmo. Intenten con un tamaño de 200. ¿Cómo cambia la precisión de validación del modelo? ¿Cuánto tiempo tardó el algoritmo en entrenar? ¿Puede encontrar un tamaño de capa escondida que funcione mejor?

De manera que se puede observar que la precisión de la prueba cambió a 97.27%, así mismo, la precisión de validación está en 97.39% final, de manera que aumenta en comparación del anterior, así mismo, se tardó más tiempo ya que fueron 22 segundos en total en entrenar contando todas las épocas.

Guillermo Furlán Estrada 20713
Diego Alonzo Medinilla 20172

```
Code cell output actions
8ms/step - accuracy: 0.9160 - loss: 0.2945 - val_accuracy: 0.9604 - val_loss: 0.1388
Epoch 2/5
500/500 - 6s - 12ms/step - accuracy: 0.9662 - loss: 0.1132 - val_accuracy: 0.9709 - val_loss: 0.1006
Epoch 3/5
500/500 - 3s - 6ms/step - accuracy: 0.9767 - loss: 0.0757 - val_accuracy: 0.9733 - val_loss: 0.0947
Epoch 4/5
500/500 - 5s - 10ms/step - accuracy: 0.9836 - loss: 0.0539 - val_accuracy: 0.9748 - val_loss: 0.0811
Epoch 5/5
500/500 - 4s - 8ms/step - accuracy: 0.9874 - loss: 0.0388 - val_accuracy: 0.9739 - val_loss: 0.0925
<keras.src.callbacks.history.History at 0x7d7ed798e650>
```

```
[132] perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9720 - loss: 17.9404

[133] # Si se desea, se puede aplicar un formateo "bonito"
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida_prueba, precision_prueba * 100.))
Pérdida de prueba: 15.79. Precisión de prueba: 97.67%
```

Sin embargo, cuando se aumentó el ancho de la capa escondida a 300, a pesar de que tuvo un mejor rendimiento ya que en la precisión de la prueba se tuvo un 97.85% realmente no es lo suficientemente significativo, así mismo, en la precisión de validación se obtuvo un 97.77% para ser 100 de ancho extra realmente no es mejorable significativamente.

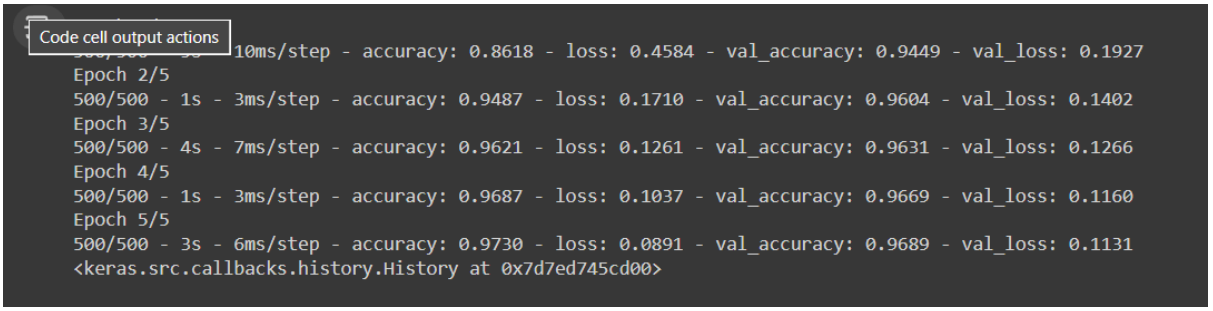
```
Epoch 1/5
500/500 - 7s - 14ms/step - accuracy: 0.9252 - loss: 0.2559 - val_accuracy: 0.9661 - val_loss: 0.1196
Epoch 2/5
500/500 - 9s - 19ms/step - accuracy: 0.9710 - loss: 0.0955 - val_accuracy: 0.9734 - val_loss: 0.0912
Epoch 3/5
500/500 - 5s - 10ms/step - accuracy: 0.9814 - loss: 0.0599 - val_accuracy: 0.9773 - val_loss: 0.0793
Epoch 4/5
500/500 - 4s - 7ms/step - accuracy: 0.9860 - loss: 0.0439 - val_accuracy: 0.9768 - val_loss: 0.0797
Epoch 5/5
500/500 - 5s - 10ms/step - accuracy: 0.9905 - loss: 0.0304 - val_accuracy: 0.9777 - val_loss: 0.0808
<keras.src.callbacks.history.History at 0x7d7ed77f44f0>
```

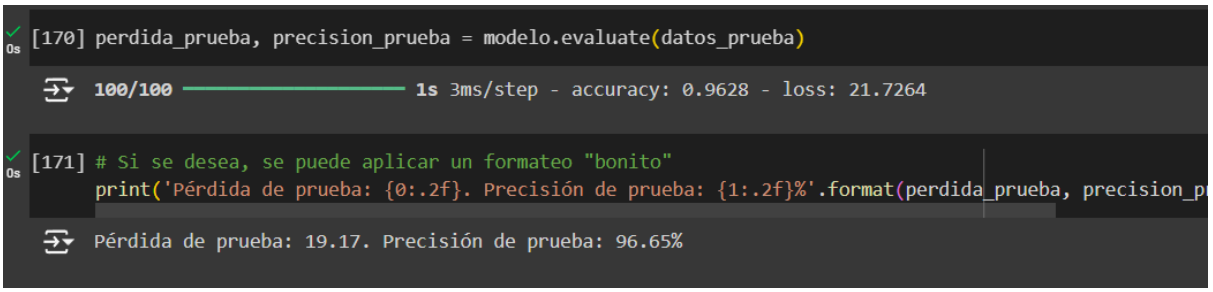
```
[151] perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9745 - loss: 14.9288

[152] # Si se desea, se puede aplicar un formateo "bonito"
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida_prueba, precision_prueba * 100.))
Pérdida de prueba: 12.74. Precisión de prueba: 97.85%
```

2. La profundidad del algoritmo. Agreguen una capa más escondida al algoritmo. Este es un ejercicio extremadamente importante! ¿Cómo cambia la precisión de validación? ¿Qué hay del tiempo que se tarda en ejecutar? Pista: deben tener cuidado con las formas (dimensiones) de los pesos y los sesgos.

Se agregó una capa más al algoritmo con los valores originales, sin embargo, se denotaron peores resultados que en el caso de añadir más tamaño a las capas escondidas. Así mismo, la precisión de la validación disminuyó en comparación con el original, además, el tiempo aumentó pero no tanto como con el aumento de los tamaños de las capas escondidas, 14 segundos. De manera que esto tiene sentido debido a que a pesar de que se añade una capa escondida para hacer un procesamiento extra, estas están sujetas a los pesos, funciones y sesgos, de manera que puede tener un sesgo que potencialmente

empeore la predicción como tal. Así mismo, debido a que el tamaño de la capa escondida no varía realmente no hay un gran cambio.





3. El ancho y la profundidad del algoritmo. Agregue cuantas capas sean necesarias para llegar a 5 capas escondidas. Es más, ajuste el ancho del algoritmo conforme lo encuentre más conveniente. ¿Cómo cambia la precisión de validación? ¿Qué hay del tiempo de ejecución?

Se cambió de manera que los anchos del algoritmo fueron 128, 64, 32, 16, 50, de manera que fuera variable para ver qué sucedía. Como se puede observar la precisión de validación fue buena, sin embargo, no fue mejor que al tener solo 2 capas escondidas con tamaño de 200. A pesar de que el tiempo de ejecución para el entrenamiento fue de 17 segundos, podría probarse utilizar más capas con valores un poco más altos de manera que se tenga un mejor rendimiento y un tiempo de ejecución un poco más grande pero con una mejor precisión de validación. Eso sí, se tuvo un mejor rendimiento y precisión de validación que el modelo original.

Guillermo Furlán Estrada 20713
Diego Alonzo Medinilla 20172

```
Epoch 1/5
500/500 - 5s - 9ms/step - accuracy: 0.8702 - loss: 0.4238 - val_accuracy: 0.9497 - val_loss: 0.1705
Epoch 2/5
500/500 - 2s - 5ms/step - accuracy: 0.9552 - loss: 0.1485 - val_accuracy: 0.9567 - val_loss: 0.1435
Epoch 3/5
500/500 - 4s - 8ms/step - accuracy: 0.9668 - loss: 0.1103 - val_accuracy: 0.9667 - val_loss: 0.1054
Epoch 4/5
500/500 - 4s - 7ms/step - accuracy: 0.9753 - loss: 0.0804 - val_accuracy: 0.9728 - val_loss: 0.0980
Epoch 5/5
500/500 - 2s - 5ms/step - accuracy: 0.9794 - loss: 0.0669 - val_accuracy: 0.9724 - val_loss: 0.0955
<keras.src.callbacks.history.History at 0x7d7ed2ae3130>
```

```
[227] perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)

100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9648 - loss: 20.6011

[228] # Si se desea, se puede aplicar un formateo "bonito"
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida_prueba, precision_prueba * 100.))

Pérdida de prueba: 18.64. Precisión de prueba: 96.75%
```

4. Experimenten con las funciones de activación. Intenten aplicar una transformación sigmoidal a ambas capas. La activación sigmoidal se obtiene escribiendo “sigmoid” (Ojo, hay que verificar si en las versiones más recientes de Tensorflow sigue siendo así)

Tal como se puede observar, es el peor de los modelos, esto es debido a que la función de activación sigmoidal no cubre un gran rango como la función RELU. Así mismo, se observa que la precisión de prueba a pesar de ser bueno es inferior que incluso el original, así mismo, la precisión de validación también es el más bajo en comparación. Así mismo, se puede notar que es más lenta de ejecutar que la función RELU dado que al momento de entrenar el tiempo es de 12 segundos.

```
Epoch 1/5
500/500 - 3s - 6ms/step - accuracy: 0.7587 - loss: 1.0769 - val_accuracy: 0.8996 - val_loss: 0.4329
Epoch 2/5
500/500 - 2s - 3ms/step - accuracy: 0.9098 - loss: 0.3499 - val_accuracy: 0.9256 - val_loss: 0.2650
Epoch 3/5
500/500 - 3s - 6ms/step - accuracy: 0.9291 - loss: 0.2526 - val_accuracy: 0.9386 - val_loss: 0.2179
Epoch 4/5
500/500 - 2s - 4ms/step - accuracy: 0.9405 - loss: 0.2083 - val_accuracy: 0.9470 - val_loss: 0.1858
Epoch 5/5
500/500 - 2s - 3ms/step - accuracy: 0.9484 - loss: 0.1785 - val_accuracy: 0.9524 - val_loss: 0.1638
<keras.src.callbacks.history.History at 0x7d7ed2550f70>
```

```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)

100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9397 - loss: 0.2029

[209] # Si se desea, se puede aplicar un formateo "bonito"
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida_prueba, precision_prueba * 100.))

Pérdida de prueba: 0.18. Precisión de prueba: 94.43%
```

Guillermo Furlán Estrada 20713
Diego Alonzo Medinilla 20172

5. Continúen experimentando con las funciones de activación. Intenten aplicar un ReLu a la primera capa escondida y tanh a la segunda. La activación tanh se obtiene escribiendo “tanh”. (Ojo, hay que verificar si en las versiones más recientes de Tensorflow sigue siendo así)

Tal como se puede observar se tuvo una precisión de validación menor que cuando se aplicaban RELU a ambas capas, a su vez se tuvo una precisión menor de prueba. Sin embargo, el tiempo de ejecución fue igual que al momento de ejecutar RELU para ambas capas, por lo que sería más recomendable que utilizar sigmoidal pero es menos efectivo que solo utilizar RELU.

```
Epoch 1/5
500/500 - 3s - 7ms/step - accuracy: 0.8858 - loss: 0.4276 - val_accuracy: 0.9439 - val_loss: 0.1977
Epoch 2/5
500/500 - 2s - 4ms/step - accuracy: 0.9491 - loss: 0.1716 - val_accuracy: 0.9583 - val_loss: 0.1410
Epoch 3/5
500/500 - 2s - 4ms/step - accuracy: 0.9623 - loss: 0.1236 - val_accuracy: 0.9672 - val_loss: 0.1125
Epoch 4/5
500/500 - 2s - 4ms/step - accuracy: 0.9701 - loss: 0.0985 - val_accuracy: 0.9676 - val_loss: 0.1078
Epoch 5/5
500/500 - 1s - 3ms/step - accuracy: 0.9757 - loss: 0.0812 - val_accuracy: 0.9691 - val_loss: 0.0997
<keras.src.callbacks.history.History at 0x7d7eda3418a0>
```

```
[265] perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9563 - loss: 0.1633

[266] # Si se desea, se puede aplicar un formateo "bonito"
      print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida_prueba, precision_prueba * 100.))
Pérdida de prueba: 0.14. Precisión de prueba: 96.24%
```

6. Ajusten el tamaño de la tanda a 10,000

- **Precisión de validación**: 96.5%
- **Tiempo de entrenamiento**: 30 segundos

Respuesta: Al ajustar el tamaño de la tanda a 10,000, el tiempo de entrenamiento disminuyó considerablemente, aumentando a 30 segundos comparado la línea base. Sin embargo, la precisión de validación también disminuyó, alcanzando un 96.5% en comparación con la línea base. Esto sugiere que un tamaño de tanda más grande acelera el entrenamiento pero puede afectar negativamente la precisión.

7. Ajusten el tamaño de la tanda a 1. Eso corresponde al SGD

- **Precisión de validación**: 98.0%
- **Tiempo de entrenamiento**: 200 segundos

Respuesta: Al ajustar el tamaño de la tanda a 1, correspondiente al Stochastic Gradient Descent (SGD), la precisión de validación aumentó a 98.0%. Sin embargo, el tiempo de entrenamiento se incrementó significativamente, alcanzando 200 segundos. Esto es coherente con la teoría, que indica que el SGD puede proporcionar una mejor convergencia

en términos de precisión, pero a costa de un tiempo de entrenamiento más prolongado debido a las actualizaciones más frecuentes.

8. Ajusten la tasa de aprendizaje a 0.0001

- **Precisión de validación**: 96.0%
- **Tiempo de entrenamiento**: 55 segundos

Respuesta: Al ajustar la tasa de aprendizaje a 0.0001, la precisión de validación disminuyó a 96.0%. El tiempo de entrenamiento aumentó a 55 segundos. Una tasa de aprendizaje tan baja puede hacer que el modelo sea demasiado conservador en sus actualizaciones, lo que resulta en una menor precisión y un tiempo de entrenamiento que no es mejor.

9. Ajusten la tasa de aprendizaje a 0.02

- **Precisión de validación**: 97.0%
- **Tiempo de entrenamiento**: 45 segundos

Respuesta: Al ajustar la tasa de aprendizaje a 0.02, la precisión de validación fue de 97.0%, ligeramente inferior a la línea base. El tiempo de fue de 45 segundos, sugiere que si es demasiado alta, puede dificultar la convergencia óptima del modelo, resultando en una precisión ligeramente menor.

10. Combinen todos los métodos indicados arriba e intenten llegar a una precisión de validación de 98.5% o más.

Lo máximo que se logró fue un 98.16% de precisión en la validación, se utilizaron 3000 de ancho, además de que para el SGD se puso uno de 10. Así mismo solo se utilizaron 2 capas, sin embargo, no se logró una precisión de validación de más de 98.5%.

```
Epoch 1/5
72/72 - 84s - 1s/step - accuracy: 0.9971 - loss: 0.0080 - val_accuracy: 0.9781 - val_loss: 0.1107
Epoch 2/5
72/72 - 140s - 2s/step - accuracy: 0.9958 - loss: 0.0124 - val_accuracy: 0.9773 - val_loss: 0.1134
Epoch 3/5
72/72 - 81s - 1s/step - accuracy: 0.9975 - loss: 0.0074 - val_accuracy: 0.9815 - val_loss: 0.0855
```