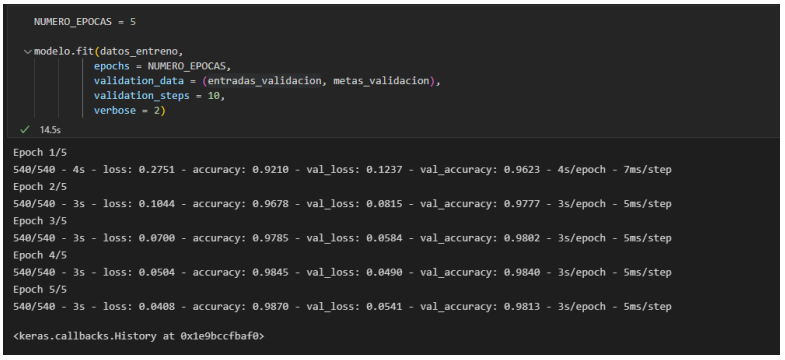
Jose Ponce 19092
Gabriel Quiroz

Laboratorio 3

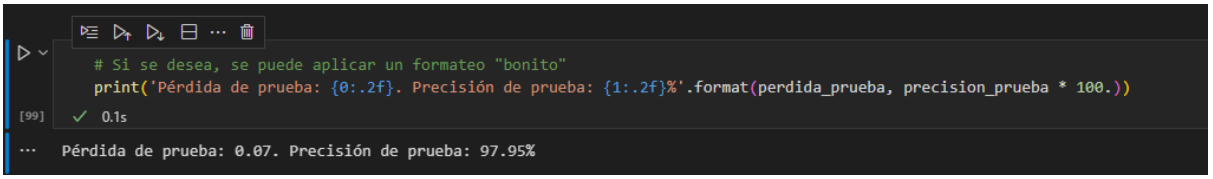
1. El ancho (tamaño de la capa escondida) del algoritmo. Intenten con un tamaño de 200. ¿Cómo cambia la precisión de validación del modelo? ¿Cuánto tiempo se tardó el algoritmo en entrenar? ¿Puede encontrar un tamaño de capa escondida que funcione mejor?

Con un tamaño de 200, la precisión de validación del modelo aumentó a 97.79%

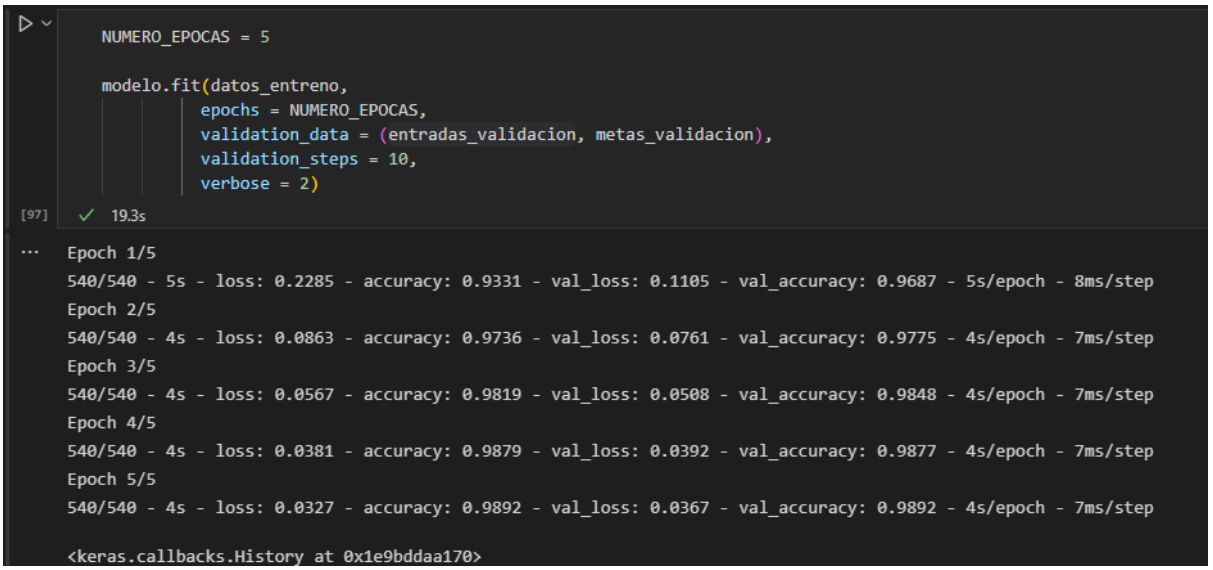
Se tardó en entrenar 14.5s



Con un tamaño de capa escondida de 400, se obtiene una validación del modelo del 97.95%



Tardandose en entrenar 19.3s



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 de los pesos y los sesgos.

Agregar una capa más al algoritmo provoca que la precisión de validación disminuya a 97.83%

```
# 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.))
```

[104] ✓ 0.1s

```
Pérdida de prueba: 0.08. Precisión de prueba: 97.83%
```

El tiempo que tarda en ejecutar aumenta a 22.8s

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

2] ✓ 22.8s

```
Epoch 1/5
540/540 - 5s - loss: 0.2232 - accuracy: 0.9331 - val_loss: 0.0919 - val_accuracy: 0.9695 - 5s/epoch - 10ms/step
Epoch 2/5
540/540 - 4s - loss: 0.0901 - accuracy: 0.9719 - val_loss: 0.0694 - val_accuracy: 0.9803 - 4s/epoch - 8ms/step
Epoch 3/5
540/540 - 4s - loss: 0.0622 - accuracy: 0.9804 - val_loss: 0.0567 - val_accuracy: 0.9830 - 4s/epoch - 8ms/step
Epoch 4/5
540/540 - 4s - loss: 0.0460 - accuracy: 0.9851 - val_loss: 0.0508 - val_accuracy: 0.9847 - 4s/epoch - 8ms/step
Epoch 5/5
540/540 - 4s - loss: 0.0387 - accuracy: 0.9881 - val_loss: 0.0411 - val_accuracy: 0.9885 - 4s/epoch - 8ms/step

<keras.callbacks.History at 0x1e9c00ef970>
```

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

Con un tamaño de capa escondida de 550 y las 5 capas escondidas, la precisión de validación aumenta a 98.04%

```
# 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.))
```

[153] ✓ 0.1s

```
Pérdida de prueba: 0.08. Precisión de prueba: 98.04%
```

El tiempo de ejecución es de 45.1, aunque hay que tener en cuenta que la precisión aumentó, las capas escondidas aumentaron y el tamaño de las capas escondidas es mayor.

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

[151] ✓ 45.1s

```
Epoch 1/5
540/540 - 10s - loss: 0.2352 - accuracy: 0.9286 - val_loss: 0.1194 - val_accuracy: 0.9637 - 10s/epoch - 18ms/step
Epoch 2/5
540/540 - 9s - loss: 0.1045 - accuracy: 0.9686 - val_loss: 0.1066 - val_accuracy: 0.9708 - 9s/epoch - 16ms/step
Epoch 3/5
540/540 - 9s - loss: 0.0742 - accuracy: 0.9783 - val_loss: 0.0637 - val_accuracy: 0.9818 - 9s/epoch - 16ms/step
Epoch 4/5
540/540 - 9s - loss: 0.0572 - accuracy: 0.9829 - val_loss: 0.0521 - val_accuracy: 0.9848 - 9s/epoch - 16ms/step
Epoch 5/5
540/540 - 9s - loss: 0.0477 - accuracy: 0.9859 - val_loss: 0.0449 - val_accuracy: 0.9885 - 9s/epoch - 16ms/step

<keras.callbacks.History at 0x1e9cc41ed10>
```

4. Experimenten con las funciones de activación. Intenten aplicar una transformación sigmoidal a ambas capas. La activación sigmoidal se obtiene escribiendo “sigmoid”.

Con un tamaño de capa escondida de 550, dos capas escondidas se obtiene una validación del 96.69%

```
modelo = tf.keras.Sequential([

    tf.keras.layers.Flatten(input_shape=(28, 28, 1)), # capa entrada

    tf.keras.layers.Dense(tamanio_capa_escondida, activation='sigmoid'), # 1era capa escondida
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='sigmoid'), # 2nda capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 3ra capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 4ta capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 5ta capa escondida

    tf.keras.layers.Dense(tamanio_salida, activation='softmax') # capa salida
])
```

✓ 0.1s

```
# 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.))
```

[159] ✓ 0.1s

```
Pérdida de prueba: 0.11. Precisión de prueba: 96.69%
```

El tiempo de entrenamiento es de 28.9s

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

[157] ✓ 28.9s

```
Epoch 1/5
540/540 - 6s - loss: 0.4732 - accuracy: 0.8624 - val_loss: 0.2397 - val_accuracy: 0.9272 - 6s/epoch - 12ms/step
Epoch 2/5
540/540 - 6s - loss: 0.2116 - accuracy: 0.9362 - val_loss: 0.1628 - val_accuracy: 0.9525 - 6s/epoch - 10ms/step
Epoch 3/5
540/540 - 5s - loss: 0.1502 - accuracy: 0.9551 - val_loss: 0.1181 - val_accuracy: 0.9635 - 5s/epoch - 10ms/step
Epoch 4/5
540/540 - 6s - loss: 0.1123 - accuracy: 0.9663 - val_loss: 0.0960 - val_accuracy: 0.9717 - 6s/epoch - 11ms/step
Epoch 5/5
540/540 - 6s - loss: 0.0885 - accuracy: 0.9731 - val_loss: 0.0833 - val_accuracy: 0.9735 - 6s/epoch - 11ms/step

<keras.callbacks.History at 0x1e9cdbd2380>
```

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”.

Capa escondida de 550 y dos capas escondidas:

Con la activación tanh se obtiene una precisión de 97.96% con un tiempo de ejecución de 27.8s

```
modelo = tf.keras.Sequential([

    tf.keras.layers.Flatten(input_shape=(28, 28, 1)), # capa entrada

    tf.keras.layers.Dense(tamanio_capa_escondida, activation='tanh'), # 1era capa escondida
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='tanh'), # 2nda capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 3ra capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 4ta capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 5ta capa escondida

    tf.keras.layers.Dense(tamanio_salida, activation='softmax') # capa salida
])
```

✓ 0.1s

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

[167] ✓ 27.8s

```
Epoch 1/5
540/540 - 6s - loss: 0.0352 - accuracy: 0.9893 - val_loss: 0.0419 -
val_accuracy: 0.9865 - 6s/epoch - 11ms/step
Epoch 2/5
540/540 - 6s - loss: 0.0280 - accuracy: 0.9914 - val_loss: 0.0269 -
val_accuracy: 0.9923 - 6s/epoch - 10ms/step
Epoch 3/5
540/540 - 5s - loss: 0.0211 - accuracy: 0.9935 - val_loss: 0.0280 -
val_accuracy: 0.9908 - 5s/epoch - 10ms/step
Epoch 4/5
540/540 - 6s - loss: 0.0186 - accuracy: 0.9940 - val_loss: 0.0254 -
val_accuracy: 0.9932 - 6s/epoch - 10ms/step
Epoch 5/5
540/540 - 5s - loss: 0.0160 - accuracy: 0.9949 - val_loss: 0.0143 -
val_accuracy: 0.9970 - 5s/epoch - 10ms/step

<keras.callbacks.History at 0x1e9cd8dfdc0>
```

```
# 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.))
```

[169] ✓ 0.1s

```
Pérdida de prueba: 0.07. Precisión de prueba: 97.96%
```

Con la activación ReLu se obtiene una precisión de 98.06% con un tiempo de ejecución de 27.7s

```
modelo = tf.keras.Sequential([

    tf.keras.layers.Flatten(input_shape=(28, 28, 1)), # capa entrada

    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 1era capa escondida
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 2nda capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 3ra capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 4ta capa escondida
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 5ta capa escondida

    tf.keras.layers.Dense(tamanio_salida, activation='softmax') # capa salida
])
```

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

[172] ✓ 27.7s

```
Epoch 1/5
540/540 - 6s - loss: 0.2150 - accuracy: 0.9349 - val_loss: 0.0907 - val_accuracy: 0.9708 - 6s/epoch - 12ms/step
Epoch 2/5
540/540 - 5s - loss: 0.0808 - accuracy: 0.9744 - val_loss: 0.0657 - val_accuracy: 0.9800 - 5s/epoch - 10ms/step
Epoch 3/5
540/540 - 5s - loss: 0.0538 - accuracy: 0.9831 - val_loss: 0.0514 - val_accuracy: 0.9858 - 5s/epoch - 10ms/step
Epoch 4/5
540/540 - 6s - loss: 0.0398 - accuracy: 0.9872 - val_loss: 0.0475 - val_accuracy: 0.9837 - 6s/epoch - 10ms/step
Epoch 5/5
540/540 - 6s - loss: 0.0297 - accuracy: 0.9901 - val_loss: 0.0319 - val_accuracy: 0.9927 - 6s/epoch - 10ms/step

<keras.callbacks.History at 0x1e9d20bf910>
```

```
# 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.))
```

[174] ✓ 0.1s

```
Pérdida de prueba: 0.07. Precisión de prueba: 98.06%
```

6. Ajusten el tamaño de la tanda. Prueben con un tamaño de tanda de 10,000. ¿Cómo cambia el tiempo requerido? ¿Cómo cambia la precisión?
La precisión cambia considerablemente, bajando de 98.06% a 93.05% y el tiempo aumento 20s más.

```
TAMANIO_TANDA = 10000

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
```

[16] ✓ 0.7s

```
# 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.))
```

[24] ✓ 0.1s

```
Pérdida de prueba: 0.24. Precisión de prueba: 93.05%
```

7. Ajusten el tamaño de la tanda a 1. Eso corresponde al SGD. ¿Cómo cambian el tiempo y la precisión? ¿Es el resultado coherente con la teoría?

```
TAMANIO_TANDA = 1

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
```

[16] ✓ 0.1s

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

[22] ✓ 22m 1.3s

```
Epoch 1/5
54000/54000 - 274s - loss: 0.2894 - accuracy: 0.9252 - val_loss: 0.2136 - val_accuracy: 0.9503 - 274s/epoch - 5ms/step
Epoch 2/5
54000/54000 - 272s - loss: 0.1907 - accuracy: 0.9565 - val_loss: 0.1552 - val_accuracy: 0.9643 - 272s/epoch - 5ms/step
Epoch 3/5
54000/54000 - 255s - loss: 0.1709 - accuracy: 0.9622 - val_loss: 0.1600 - val_accuracy: 0.9658 - 255s/epoch - 5ms/step
Epoch 4/5
54000/54000 - 265s - loss: 0.1643 - accuracy: 0.9666 - val_loss: 0.1872 - val_accuracy: 0.9677 - 265s/epoch - 5ms/step
Epoch 5/5
54000/54000 - 255s - loss: 0.1532 - accuracy: 0.9695 - val_loss: 0.1880 - val_accuracy: 0.9665 - 255s/epoch - 5ms/step

<keras.callbacks.History at 0x27591bf86a0>
```

```
# 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.))
```

✓ 0.8s

```
Pérdida de prueba: 0.21. Precisión de prueba: 95.98%
```

El tiempo de entrenamiento aumentó muchísimo ya que se tardó 22 minutos, sin embargo la precisión de la prueba aumentó a 95.98%. Es coherente puesto ya que al solo tener una tanda, esta se entrena por muchísimo más tiempo dando lugar a que la precisión aumente o tenga más variaciones.

8. Ajusten la tasa de aprendizaje. Prueben con un valor de 0.0001. ¿Hace alguna diferencia?

```
# 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.))
```

[29] ✓ 0.2s

```
Pérdida de prueba: 0.23. Precisión de prueba: 97.94%
```

La precisión de la prueba aumentó considerablemente, llegando casi a 98%

9. Ajusten la tasa de aprendizaje a 0.02. ¿Hay alguna diferencia?

```
NUMERO_EPOCAS = 5

modelo.fit(datos_entreno,
           epochs = NUMERO_EPOCAS,
           validation_data = (entradas_validacion, metas_validacion),
           validation_steps = 10,
           verbose = 2)
```

✓ 23m 45.1s

```
Epoch 1/5
54000/54000 - 287s - loss: 2.3785 - accuracy: 0.1086 - val_loss: 2.3266 - val_accuracy: 0.1133 - 287s/epoch - 5ms/step
Epoch 2/5
54000/54000 - 289s - loss: 2.3167 - accuracy: 0.1003 - val_loss: 2.3091 - val_accuracy: 0.0953 - 289s/epoch - 5ms/step
Epoch 3/5
54000/54000 - 278s - loss: 2.3162 - accuracy: 0.1039 - val_loss: 2.3139 - val_accuracy: 0.0913 - 278s/epoch - 5ms/step
Epoch 4/5
54000/54000 - 287s - loss: 2.3162 - accuracy: 0.1021 - val_loss: 2.3093 - val_accuracy: 0.1133 - 287s/epoch - 5ms/step
Epoch 5/5
54000/54000 - 285s - loss: 2.3164 - accuracy: 0.1023 - val_loss: 2.3055 - val_accuracy: 0.0918 - 285s/epoch - 5ms/step
```

```
# 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.))
```

[34] ✓ 0.1s

```
Pérdida de prueba: 2.31. Precisión de prueba: 8.92%
```

Al realizar esto el tiempo de ejecución aumento y lo más alarmante es que la prueba de precisión disminuyó hasta un 8.92%

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

```
# 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.))
```

[41] ✓ 0.1s

```
Pérdida de prueba: 0.09. Precisión de prueba: 98.80%
```