Universidad del Valle de Guatemala
Data Science
Sección 20

Andrés de la Roca 20332
Jun Woo Lee 20

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?

```
tamanio_capa_escondida = 50
```

```
NUMERO_EPOCAS = 5

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

✓ 3.8s

```
Epoch 1/5
540/540 - 1s - loss: 0.4171 - accuracy: 0.8829 - val_loss: 0.2208 - val_accuracy: 0.9377 - 1s/epoch - 2ms/step
Epoch 2/5
540/540 - 1s - loss: 0.1881 - accuracy: 0.9454 - val_loss: 0.1605 - val_accuracy: 0.9545 - 683ms/epoch - 1ms/step
Epoch 3/5
540/540 - 1s - loss: 0.1452 - accuracy: 0.9572 - val_loss: 0.1405 - val_accuracy: 0.9605 - 679ms/epoch - 1ms/step
Epoch 4/5
540/540 - 1s - loss: 0.1198 - accuracy: 0.9645 - val_loss: 0.1152 - val_accuracy: 0.9652 - 690ms/epoch - 1ms/step
Epoch 5/5
540/540 - 1s - loss: 0.1018 - accuracy: 0.9698 - val_loss: 0.1015 - val_accuracy: 0.9713 - 683ms/epoch - 1ms/step
```

```
Pérdida de prueba: 0.11. Precisión de prueba: 96.50%
```

```
tamanio_capa_escondida = 200
```

```
NUMERO_EPOCAS = 5

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

✓ 6.6s

```
Epoch 1/5
540/540 - 2s - loss: 0.2776 - accuracy: 0.9198 - val_loss: 0.1445 - val_accuracy: 0.9580 - 2s/epoch - 3ms/step
Epoch 2/5
540/540 - 1s - loss: 0.1084 - accuracy: 0.9672 - val_loss: 0.0905 - val_accuracy: 0.9723 - 1s/epoch - 2ms/step
Epoch 3/5
540/540 - 1s - loss: 0.0732 - accuracy: 0.9771 - val_loss: 0.0634 - val_accuracy: 0.9808 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.0540 - accuracy: 0.9831 - val_loss: 0.0619 - val_accuracy: 0.9810 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.0412 - accuracy: 0.9868 - val_loss: 0.0434 - val_accuracy: 0.9877 - 1s/epoch - 2ms/step
```

```
Pérdida de prueba: 0.07. Precisión de prueba: 97.86%
```

```
tamanio_capa_escondida = 220
```

```
✓ 8.1s
Epoch 1/5
540/540 - 2s - loss: 0.2643 - accuracy: 0.9227 - val_loss: 0.1305 - val_accuracy: 0.9593 - 2s/epoch - 4ms/step
Epoch 2/5
540/540 - 2s - loss: 0.1031 - accuracy: 0.9686 - val_loss: 0.0902 - val_accuracy: 0.9730 - 2s/epoch - 3ms/step
Epoch 3/5
540/540 - 2s - loss: 0.0700 - accuracy: 0.9787 - val_loss: 0.0695 - val_accuracy: 0.9783 - 2s/epoch - 3ms/step
Epoch 4/5
540/540 - 2s - loss: 0.0507 - accuracy: 0.9843 - val_loss: 0.0533 - val_accuracy: 0.9865 - 2s/epoch - 3ms/step
Epoch 5/5
540/540 - 1s - loss: 0.0375 - accuracy: 0.9882 - val_loss: 0.0462 - val_accuracy: 0.9863 - 1s/epoch - 3ms/step
```

```
Pérdida de prueba: 0.07. Precisión de prueba: 97.87%
```

Al analizar varios resultados se ve que al aumentar el tamaño de la capa, se vuelve más exacto. Pero hay un límite a cuánto se puede aumentar ya que al tener un tamaño muy grande empieza a tener peor precisión. Adicionalmente mientras más grande más tarda de ejecutar. Adicionalmente después de varias pruebas se encontró que el tamaño de 220 es el que mejores resultados trajo.

2. La profundidad del algoritmo. Agreguen una capa escondida más 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.

```
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'), # 3nda 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)
```

✓ 7.3s

```
Epoch 1/5
540/540 - 2s - loss: 0.2592 - accuracy: 0.9240 - val_loss: 0.1081 - val_accuracy: 0.9655 - 2s/epoch - 3ms/step
Epoch 2/5
540/540 - 1s - loss: 0.0977 - accuracy: 0.9696 - val_loss: 0.0664 - val_accuracy: 0.9792 - 1s/epoch - 3ms/step
Epoch 3/5
540/540 - 1s - loss: 0.0652 - accuracy: 0.9795 - val_loss: 0.0602 - val_accuracy: 0.9800 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.0487 - accuracy: 0.9842 - val_loss: 0.0375 - val_accuracy: 0.9895 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.0395 - accuracy: 0.9878 - val_loss: 0.0361 - val_accuracy: 0.9883 - 1s/epoch - 2ms/step
```

```
Pérdida de prueba: 0.08. Precisión de prueba: 97.89%
```

Al agregarle una capa escondida, la precisión de validación tuvo un incremento mínimo de 97.87 a 97.89 y también como un segundo más de ejecución.

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?

```
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'), # 3nda 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)
```

✓ 10.1s

```
Epoch 1/5
540/540 - 3s - loss: 0.2615 - accuracy: 0.9210 - val_loss: 0.1224 - val_accuracy: 0.9607 - 3s/epoch - 5ms/step
Epoch 2/5
540/540 - 2s - loss: 0.1023 - accuracy: 0.9693 - val_loss: 0.0863 - val_accuracy: 0.9740 - 2s/epoch - 3ms/step
Epoch 3/5
540/540 - 2s - loss: 0.0755 - accuracy: 0.9761 - val_loss: 0.0751 - val_accuracy: 0.9780 - 2s/epoch - 3ms/step
Epoch 4/5
540/540 - 2s - loss: 0.0608 - accuracy: 0.9808 - val_loss: 0.0637 - val_accuracy: 0.9800 - 2s/epoch - 4ms/step
Epoch 5/5
540/540 - 2s - loss: 0.0519 - accuracy: 0.9841 - val_loss: 0.0548 - val_accuracy: 0.9840 - 2s/epoch - 4ms/step
```

```
Pérdida de prueba: 0.09. Precisión de prueba: 97.39%
```

Al tener 5 capas escondidas, se ve que la precisión no mejora pero empeora ya que la pérdida sigue aumentando y la exactitud aumento también. Adicionalmente el tiempo de ejecución subio a 10 segundos de 7. Y cómo se agregaron 3 capas se puede decir que cada capa agrega un segundo más de ejecución.

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

```
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'), # 3nda 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)
```

✓ 6.6s

```
Epoch 1/5
540/540 - 2s - loss: 0.5656 - accuracy: 0.8461 - val_loss: 0.2696 - val_accuracy: 0.9193 - 2s/epoch - 3ms/step
Epoch 2/5
540/540 - 1s - loss: 0.2237 - accuracy: 0.9337 - val_loss: 0.1877 - val_accuracy: 0.9438 - 1s/epoch - 2ms/step
Epoch 3/5
540/540 - 1s - loss: 0.1681 - accuracy: 0.9496 - val_loss: 0.1462 - val_accuracy: 0.9568 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.1305 - accuracy: 0.9621 - val_loss: 0.1241 - val_accuracy: 0.9632 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.1051 - accuracy: 0.9693 - val_loss: 0.0968 - val_accuracy: 0.9733 - 1s/epoch - 2ms/step
```

```
Pérdida de prueba: 0.10. Precisión de prueba: 96.68%
```

Al cambiar las funciones de activación a sigmoidal, se ve que el modelo funciona peor ya que la precisión por primera vez bajo a 96% y la perdida de prueba también llego a 0.10

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”.

```
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='tanh'), # 2nda capa escondida
    # tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 3nda 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)
```

✓ 6.4s

```
Epoch 1/5
540/540 - 2s - loss: 0.2472 - accuracy: 0.9274 - val_loss: 0.1266 - val_accuracy: 0.9623 - 2s/epoch - 3ms/step
Epoch 2/5
540/540 - 1s - loss: 0.0946 - accuracy: 0.9710 - val_loss: 0.0826 - val_accuracy: 0.9760 - 1s/epoch - 2ms/step
Epoch 3/5
540/540 - 1s - loss: 0.0617 - accuracy: 0.9804 - val_loss: 0.0765 - val_accuracy: 0.9763 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.0445 - accuracy: 0.9858 - val_loss: 0.0498 - val_accuracy: 0.9850 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.0360 - accuracy: 0.9885 - val_loss: 0.0369 - val_accuracy: 0.9902 - 1s/epoch - 2ms/step
```

```
Pérdida de prueba: 0.06. Precisión de prueba: 98.11%
```

Al cambiar la primera capa a ReLu y la segunda a tanh se ve que termina mejorando ya que obtuvimos el mejor resultado con precisión de 98.11 y pérdida de 0.06.

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?

```
TAMANIO_TANDA = 10000
```

```
NUMERO_EPOCAS = 5

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

✓ 2.8s

```
Epoch 1/5
6/6 - 1s - loss: 1.8689 - accuracy: 0.4919 - val_loss: 1.1722 - val_accuracy: 0.7808 - 975ms/epoch - 163ms/step
Epoch 2/5
6/6 - 0s - loss: 0.9283 - accuracy: 0.8135 - val_loss: 0.6315 - val_accuracy: 0.8462 - 482ms/epoch - 80ms/step
Epoch 3/5
6/6 - 0s - loss: 0.5469 - accuracy: 0.8573 - val_loss: 0.4433 - val_accuracy: 0.8775 - 444ms/epoch - 74ms/step
Epoch 4/5
6/6 - 0s - loss: 0.4104 - accuracy: 0.8834 - val_loss: 0.3629 - val_accuracy: 0.8948 - 449ms/epoch - 75ms/step
Epoch 5/5
6/6 - 0s - loss: 0.3473 - accuracy: 0.8976 - val_loss: 0.3164 - val_accuracy: 0.9068 - 425ms/epoch - 71ms/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.))
```

✓ 0.0s

```
Pérdida de prueba: 0.31. Precisión de prueba: 91.21%
```

Al ajustar el tamaño de la tanda a 10000, el tiempo de ejecución se ve que disminuyó un montón a 2.8 segundos cuando antes era de 6. Pero la precisión y pérdida estan mucho peor que antes. Ya que perdimos mucha precisión.

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
```

```
NUMERO_EPOCAS = 5

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

✓ 4m 20.8s

```
Epoch 1/5
54000/54000 - 53s - loss: 0.2486 - accuracy: 0.9289 - val_loss: 0.1413 - val_accuracy: 0.9635 - 53s/epoch - 984us/step
Epoch 2/5
54000/54000 - 54s - loss: 0.1462 - accuracy: 0.9595 - val_loss: 0.1014 - val_accuracy: 0.9695 - 54s/epoch - 994us/step
Epoch 3/5
54000/54000 - 51s - loss: 0.1225 - accuracy: 0.9669 - val_loss: 0.1206 - val_accuracy: 0.9705 - 51s/epoch - 951us/step
Epoch 4/5
54000/54000 - 51s - loss: 0.1136 - accuracy: 0.9697 - val_loss: 0.1003 - val_accuracy: 0.9735 - 51s/epoch - 948us/step
Epoch 5/5
54000/54000 - 51s - loss: 0.1113 - accuracy: 0.9710 - val_loss: 0.1189 - val_accuracy: 0.9712 - 51s/epoch - 953us/step
```

```
Pérdida de prueba: 0.16. Precisión de prueba: 96.15%
```

Al ajustar el tamaño de la tanda a 1, se ven los cambios que corresponde al SGD. En especial el tiempo de corrida que tomó mucho más que con 10000, con 4 minutos de entrenamiento. Además la precisión subió a 96.15

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

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
modelo.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
NUMERO_EPOCAS = 5

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

✓ 6.2s

```
Epoch 1/5
540/540 - 2s - loss: 0.6530 - accuracy: 0.8420 - val_loss: 0.2956 - val_accuracy: 0.9153 - 2s/epoch - 3ms/step
Epoch 2/5
540/540 - 1s - loss: 0.2588 - accuracy: 0.9268 - val_loss: 0.2086 - val_accuracy: 0.9395 - 1s/epoch - 2ms/step
Epoch 3/5
540/540 - 1s - loss: 0.1997 - accuracy: 0.9428 - val_loss: 0.1684 - val_accuracy: 0.9510 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.1612 - accuracy: 0.9531 - val_loss: 0.1425 - val_accuracy: 0.9585 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.1370 - accuracy: 0.9601 - val_loss: 0.1233 - val_accuracy: 0.9637 - 1s/epoch - 2ms/step
```

```
Pérdida de prueba: 0.13. Precisión de prueba: 95.99%
```

En esto se regresó al tamaño de tanda de 100 que nos daba los mejores resultados y se observó que los resultados empeoraron con un learning rate de 0.0001

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

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.02)
modelo.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
NUMERO_EPOCAS = 5

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

✓ 6.3s

```
Epoch 1/5
540/540 - 2s - loss: 0.3770 - accuracy: 0.8946 - val_loss: 0.2375 - val_accuracy: 0.9290 - 2s/epoch - 3ms/step
Epoch 2/5
540/540 - 1s - loss: 0.2929 - accuracy: 0.9198 - val_loss: 0.3309 - val_accuracy: 0.9088 - 1s/epoch - 2ms/step
Epoch 3/5
540/540 - 1s - loss: 0.3246 - accuracy: 0.9098 - val_loss: 0.3463 - val_accuracy: 0.9067 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.3365 - accuracy: 0.9086 - val_loss: 0.3300 - val_accuracy: 0.9122 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.3834 - accuracy: 0.8974 - val_loss: 0.3890 - val_accuracy: 0.9043 - 1s/epoch - 2ms/step
```

```
Pérdida de prueba: 0.41. Precisión de prueba: 90.09%
```

Al cambiar la tasa de aprendizaje a 0.02 empeoró mucho el modelo y nos dio nuestros peores resultados de 90% de precisión.

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

```
TAMANIO_TANDA = 75
```

```
tamanio_capa_escondida = 220
```

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

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

```
✓ 18m 37.9s

Epoch 1/10
60000/60000 [==============================] - 374s 6ms/step - loss: 0.1252 - accuracy: 0.9629 - val_loss: 0.0493 - val_accuracy: 0.9842
Epoch 2/10
60000/60000 [==============================] - 376s 6ms/step - loss: 0.0544 - accuracy: 0.9848 - val_loss: 0.0549 - val_accuracy: 0.9866
Epoch 3/10
60000/60000 [==============================] - 368s 6ms/step - loss: 0.0412 - accuracy: 0.9889 - val_loss: 0.0589 - val_accuracy: 0.9857
```

```
Pérdida de prueba: 0.07. Precisión de prueba: 98.14%
```

Se combinaron todos los ejercicios anteriores para llegar a una precisión de 98.14% y no se pudo lograr llegar a 98.5%. Ya que con todos los cambios y con los experimentos combinandolo, lo más que se logró encontrar fue de 98.14%