Lab 1

In [ ]:
```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
```

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?

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
```

In [ ]:
```
tamanio_capa_escondida = 200
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 cap
```

```
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 2nda cap

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

modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
540/540 - 3s - loss: 0.2748 - accuracy: 0.9223 - val_loss: 0.1219 - val_accuracy:
0.9643 - 3s/epoch - 6ms/step
Epoch 2/5
540/540 - 2s - loss: 0.1059 - accuracy: 0.9677 - val_loss: 0.0787 - val_accuracy:
0.9762 - 2s/epoch - 4ms/step
Epoch 3/5
540/540 - 2s - loss: 0.0711 - accuracy: 0.9783 - val_loss: 0.0570 - val_accuracy:
0.9830 - 2s/epoch - 4ms/step
Epoch 4/5
540/540 - 2s - loss: 0.0529 - accuracy: 0.9832 - val_loss: 0.0595 - val_accuracy:
0.9820 - 2s/epoch - 4ms/step
Epoch 5/5
540/540 - 2s - loss: 0.0393 - accuracy: 0.9879 - val_loss: 0.0412 - val_accuracy:
0.9862 - 2s/epoch - 4ms/step
```

Out[ ]:
```
<keras.callbacks.History at 0x24e06ec27d0>
```

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
# Si se desea, se puede aplicar un formateo "bonito"
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 0s 310ms/step - loss: 0.0709 - accuracy:
0.9797
Pérdida de prueba: 0.07. Precisión de prueba: 97.97%
1/1 [==============================] - 0s 310ms/step - loss: 0.0709 - accuracy:
0.9797
Pérdida de prueba: 0.07. Precisión de prueba: 97.97%
```

Respuestas

- La precisión aumentó ligeramente, ahora con un valor de 97.99 en vez de 96.73.
- El algoritmo se tardó 13.2 segundos en ser entrenado
- Encontrar un tamaño de capa escondida óptimo es un proceso de prueba y error, por lo que se puede tardar mucho tiempo en encontrar el tamaño óptimo. Se intentaron con valores entre 350 y 200, pero no se encontró un valor que mejorara la precisión de manera considerable y de manera consistente. El valor que más se acercó fue 256, con una precisión de 98.82, pero no se pudo replicar el resultado, por lo que no se considera como un valor óptimo.

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.

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:
```
modelo = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'),
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'),
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'),  # Nueva c
    tf.keras.layers.Dense(tamanio_salida, activation='softmax')
])
```

In [ ]:
```
modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
540/540 - 6s - loss: 0.2597 - accuracy: 0.9241 - val_loss: 0.1339 - val_accuracy:
0.9598 - 6s/epoch - 11ms/step
Epoch 2/5
540/540 - 3s - loss: 0.1013 - accuracy: 0.9691 - val_loss: 0.0871 - val_accuracy:
0.9752 - 3s/epoch - 6ms/step
Epoch 3/5
540/540 - 3s - loss: 0.0680 - accuracy: 0.9788 - val_loss: 0.0719 - val_accuracy:
0.9780 - 3s/epoch - 6ms/step
Epoch 4/5
540/540 - 3s - loss: 0.0516 - accuracy: 0.9840 - val_loss: 0.0624 - val_accuracy:
0.9812 - 3s/epoch - 6ms/step
Epoch 5/5
540/540 - 3s - loss: 0.0410 - accuracy: 0.9865 - val_loss: 0.0453 - val_accuracy:
0.9858 - 3s/epoch - 6ms/step
```

Out[ ]: <keras.callbacks.History at 0x24e06e0b220>

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 906ms/step - loss: 0.0723 - accuracy:
0.9785
1/1 [==============================] - 1s 906ms/step - loss: 0.0723 - accuracy:
0.9785
Pérdida de prueba: 0.07. Precisión de prueba: 97.85%
```

Respuestas

- El valor de la precisión bajó ligeramente, pero sigue siendo muy alto, por lo que no se ve afectado el modelo.
- Por otro lado, el tiempo de entrenamiento aumentó considerablemente, tomando en cuenta que originalmente este tomaba cerca de 13 segundos, ahora toma casi 19 segundos; un aumento de casi el 50%.

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?

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
```

```
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:

```
modelo = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(tamanio_salida, activation='softmax')
])

modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

NUMERO_EPOCAS = 5
```

In [ ]:

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

```
Epoch 1/5
540/540 - 6s - loss: 0.0450 - accuracy: 0.9857 - val_loss: 0.0412 - val_accuracy:
0.9873 - 6s/epoch - 11ms/step
Epoch 2/5
540/540 - 5s - loss: 0.0343 - accuracy: 0.9894 - val_loss: 0.0442 - val_accuracy:
0.9867 - 5s/epoch - 10ms/step
Epoch 3/5
540/540 - 5s - loss: 0.0338 - accuracy: 0.9897 - val_loss: 0.0332 - val_accuracy:
0.9900 - 5s/epoch - 10ms/step
Epoch 4/5
540/540 - 7s - loss: 0.0285 - accuracy: 0.9913 - val_loss: 0.0359 - val_accuracy:
0.9888 - 7s/epoch - 13ms/step
Epoch 5/5
540/540 - 7s - loss: 0.0277 - accuracy: 0.9913 - val_loss: 0.0381 - val_accuracy:
0.9895 - 7s/epoch - 13ms/step
```

Out[ ]: <keras.callbacks.History at 0x24e1109db40>

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 940ms/step - loss: 0.0952 - accuracy:
0.9779
1/1 [==============================] - 1s 940ms/step - loss: 0.0952 - accuracy:
0.9779
Pérdida de prueba: 0.10. Precisión de prueba: 97.79%
```

Respuestas

- La eficiencia no mejoró fuertemente, por más cambios y experimentos que se hicieron en cuanto al ancho de las capas.
- El tiempo de ejecucuión dobló el valor original, por lo que no se considera una mejora.

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

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta
```

```
datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_r
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:
```
modelo = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='sigmoid'),  # Camb
    tf.keras.layers.Dense(tamanio_salida, activation='softmax')
])
modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics
NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
540/540 - 4s - loss: 0.5191 - accuracy: 0.8713 - val_loss: 0.2950 - val_accuracy:
0.9198 - 4s/epoch - 8ms/step
Epoch 2/5
540/540 - 3s - loss: 0.2573 - accuracy: 0.9269 - val_loss: 0.2242 - val_accuracy:
0.9393 - 3s/epoch - 5ms/step
Epoch 3/5
540/540 - 3s - loss: 0.2016 - accuracy: 0.9422 - val_loss: 0.1854 - val_accuracy:
0.9485 - 3s/epoch - 5ms/step
Epoch 4/5
540/540 - 3s - loss: 0.1681 - accuracy: 0.9517 - val_loss: 0.1579 - val_accuracy:
0.9580 - 3s/epoch - 5ms/step
Epoch 5/5
540/540 - 3s - loss: 0.1404 - accuracy: 0.9599 - val_loss: 0.1352 - val_accuracy:
0.9628 - 3s/epoch - 5ms/step
```

Out[ ]: <keras.callbacks.History at 0x24e1269dd20>

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 0s 458ms/step - loss: 0.1351 - accuracy:
0.9604
Pérdida de prueba: 0.14. Precisión de prueba: 96.04%
1/1 [==============================] - 0s 458ms/step - loss: 0.1351 - accuracy:
0.9604
Pérdida de prueba: 0.14. Precisión de prueba: 96.04%
```

Respuestas

- El valor de la precisión empezó muy bajo, pero a medida que se fue entrenando la red, el valor de la precisión fue aumentando, hasta llegar a un valor de 0.96, lo cual es un muy buen valor, pero no es el mejor que se ha obtenido.

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".

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:
```
modelo = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'),
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='tanh'),  # Cambio
    tf.keras.layers.Dense(tamanio_salida, activation='softmax')
])
```

```
modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
540/540 - 7s - loss: 0.2557 - accuracy: 0.9255 - val_loss: 0.1373 - val_accuracy:
0.9598 - 7s/epoch - 14ms/step
Epoch 2/5
540/540 - 5s - loss: 0.1002 - accuracy: 0.9697 - val_loss: 0.0813 - val_accuracy:
0.9752 - 5s/epoch - 9ms/step
Epoch 3/5
540/540 - 4s - loss: 0.0660 - accuracy: 0.9795 - val_loss: 0.0613 - val_accuracy:
0.9810 - 4s/epoch - 8ms/step
Epoch 4/5
540/540 - 4s - loss: 0.0469 - accuracy: 0.9855 - val_loss: 0.0569 - val_accuracy:
0.9823 - 4s/epoch - 6ms/step
Epoch 5/5
540/540 - 3s - loss: 0.0351 - accuracy: 0.9893 - val_loss: 0.0421 - val_accuracy:
0.9867 - 3s/epoch - 6ms/step
```

Out[ ]:
```
<keras.callbacks.History at 0x24e12660df0>
```

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 704ms/step - loss: 0.0708 - accuracy:
0.9783
1/1 [==============================] - 1s 704ms/step - loss: 0.0708 - accuracy:
0.9783
Pérdida de prueba: 0.07. Precisión de prueba: 97.83%
```

Respuestas

- Este acercamiento tiene un muy bajo valor de pérdida y un alto valor de precisión, por lo que se considera que utilizar tangente hiperbólica como función de activación, junto con ReLU, es una buena opción para este problema.

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?

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
```

```
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
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)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:

```
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 cap
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 2nda cap

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

modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

NUMERO_EPOCAS = 5
```

In [ ]:

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

```
Epoch 1/5
6/6 - 4s - loss: 1.9456 - accuracy: 0.4706 - val_loss: 1.3710 - val_accuracy: 0.7
242 - 4s/epoch - 691ms/step
Epoch 2/5
6/6 - 1s - loss: 1.0834 - accuracy: 0.7710 - val_loss: 0.7020 - val_accuracy: 0.8
340 - 1s/epoch - 238ms/step
Epoch 3/5
6/6 - 2s - loss: 0.5887 - accuracy: 0.8486 - val_loss: 0.4589 - val_accuracy: 0.8
692 - 2s/epoch - 252ms/step
Epoch 4/5
6/6 - 1s - loss: 0.4196 - accuracy: 0.8780 - val_loss: 0.3730 - val_accuracy: 0.8
875 - 1s/epoch - 235ms/step
Epoch 5/5
6/6 - 1s - loss: 0.3529 - accuracy: 0.8970 - val_loss: 0.3282 - val_accuracy: 0.9
037 - 1s/epoch - 241ms/step
```

Out[ ]: <keras.callbacks.History at 0x24e16aa2d70>

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 649ms/step - loss: 0.3205 - accuracy:
0.9100
1/1 [==============================] - 1s 649ms/step - loss: 0.3205 - accuracy:
0.9100
Pérdida de prueba: 0.32. Precisión de prueba: 91.00%
```

Respuestas

- El tiempo de ejecución se encuentra cerca del valor original, con un valor un poco menor, por lo que se considera un acrecamiento más rápido que el original.
- Por otro lado, la precisión cayó un 6%, lo más que ha caído en cualquiera de las otras configuraciones probadas.

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?

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta
```

```
datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_r
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
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)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:
```
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 cap
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 2nda cap

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

modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
54000/54000 - 186s - loss: 0.2566 - accuracy: 0.9289 - val_loss: 0.1901 - val_acc
uracy: 0.9548 - 186s/epoch - 3ms/step
Epoch 2/5
54000/54000 - 198s - loss: 0.1801 - accuracy: 0.9565 - val_loss: 0.1477 - val_acc
uracy: 0.9655 - 198s/epoch - 4ms/step
Epoch 3/5
54000/54000 - 270s - loss: 0.1640 - accuracy: 0.9632 - val_loss: 0.1553 - val_acc
uracy: 0.9690 - 270s/epoch - 5ms/step
Epoch 4/5
54000/54000 - 229s - loss: 0.1543 - accuracy: 0.9663 - val_loss: 0.1429 - val_acc
uracy: 0.9613 - 229s/epoch - 4ms/step
Epoch 5/5
54000/54000 - 192s - loss: 0.1442 - accuracy: 0.9696 - val_loss: 0.1539 - val_acc
uracy: 0.9677 - 192s/epoch - 4ms/step
```

Out[ ]:
```
<keras.callbacks.History at 0x24e1ee60670>
```

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 769ms/step - loss: 0.2233 - accuracy:
0.9657
1/1 [==============================] - 1s 769ms/step - loss: 0.2233 - accuracy:
0.9657
Pérdida de prueba: 0.22. Precisión de prueba: 96.57%
```

Respuestas

- El tiempo es, por lejos, el más alto de todos los acercamientos. El tiempo fue más de 80 veces mayor que el del acercaiento original.
- El valor de la precisión, por otro lado, no mejora mucho. El valor de la precisión es de 0.97 (aproximadamente), casi lo mismo que en el acercamiento original.
- Sí, el resultado es coherente con la teoría del gradiente descendiente estocástico. Al reducir tanto el tamaño de la tanda, se puede lograr una mayor precisión, pero introduce mucho ruido en el proceso, por lo que el tiempo de ejecución es mucho mayor y el resultado no es mucho mejor.

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

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
```

```
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:
```
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 cap
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 2nda cap

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

modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001) , loss='

NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
540/540 - 9s - loss: 0.7265 - accuracy: 0.8199 - val_loss: 0.3280 - val_accuracy:
0.9042 - 9s/epoch - 16ms/step
Epoch 2/5
540/540 - 5s - loss: 0.2756 - accuracy: 0.9235 - val_loss: 0.2401 - val_accuracy:
0.9333 - 5s/epoch - 9ms/step
Epoch 3/5
540/540 - 5s - loss: 0.2188 - accuracy: 0.9393 - val_loss: 0.1934 - val_accuracy:
0.9440 - 5s/epoch - 9ms/step
Epoch 4/5
540/540 - 5s - loss: 0.1817 - accuracy: 0.9477 - val_loss: 0.1702 - val_accuracy:
0.9492 - 5s/epoch - 9ms/step
Epoch 5/5
540/540 - 5s - loss: 0.1579 - accuracy: 0.9542 - val_loss: 0.1515 - val_accuracy:
0.9543 - 5s/epoch - 9ms/step
```

Out[ ]:
```
<keras.callbacks.History at 0x24e1ee63040>
```

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.1536 - accuracy: 0.95
48
1/1 [==============================] - 1s 1s/step - loss: 0.1536 - accuracy: 0.95
48
Pérdida de prueba: 0.15. Precisión de prueba: 95.48%
```

Respuestas

- Ya que la tasa de aprendizaje es muy pequeña, el ajuste toma más tiempo en converger. El tiempo es aproximadamente 2 veces mayor que el tiempo del acercamiento original.
- En cuanto a temas de precisión, no se considera que haya una diferencia significativa entre ambos acercamientos.

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

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta

datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_n
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 100

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
tamanio_capa_escondida = 200
```

In [ ]:
```
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 cap
    tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'), # 2nda cap

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

modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.2) , loss='spa

NUMERO_EPOCAS = 5
```

In [ ]:
```
modelo.fit(datos_entreno,
          epochs = NUMERO_EPOCAS,
          validation_data = (entradas_validacion, metas_validacion),
```

```
            validation_steps = 10,
            verbose = 2)
```

```
Epoch 1/5
540/540 - 7s - loss: 6.9588 - accuracy: 0.1418 - val_loss: 2.2806 - val_accuracy:
0.1158 - 7s/epoch - 14ms/step
Epoch 2/5
540/540 - 4s - loss: 2.1904 - accuracy: 0.1637 - val_loss: 2.1023 - val_accuracy:
0.1882 - 4s/epoch - 8ms/step
Epoch 3/5
540/540 - 4s - loss: 2.1004 - accuracy: 0.1843 - val_loss: 2.0783 - val_accuracy:
0.1918 - 4s/epoch - 7ms/step
Epoch 4/5
540/540 - 4s - loss: 2.2288 - accuracy: 0.1345 - val_loss: 2.3247 - val_accuracy:
0.0967 - 4s/epoch - 7ms/step
Epoch 5/5
540/540 - 4s - loss: 2.3163 - accuracy: 0.1029 - val_loss: 2.3143 - val_accuracy:
0.1025 - 4s/epoch - 7ms/step
```

Out[ ]: <keras.callbacks.History at 0x24e1efd1510>

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 772ms/step - loss: 2.3125 - accuracy:
0.1028
1/1 [==============================] - 1s 772ms/step - loss: 2.3125 - accuracy:
0.1028
Pérdida de prueba: 2.31. Precisión de prueba: 10.28%
```

Respuestas

- Con un valor de 0.2 en la tasa de aprendizaje, el desempeño del modelo es muy malo, con una precisión de 10% en la validación. Esto se debe a que el modelo no logra converger a un mínimo local, por lo que no logra aprender nada.

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

Código

In [ ]:
```
datos_mnist, info_mnist = tfds.load(name='mnist',
                                    shuffle_files = False,
                                    with_info=True,
                                    as_supervised=True)

entreno_mnist, prueba_mnist = datos_mnist['train'], datos_mnist['test']
num_obs_validacion = 0.1 * info_mnist.splits['train'].num_examples
num_obs_validacion = tf.cast(num_obs_validacion, tf.int64)
num_obs_prueba = info_mnist.splits['test'].num_examples
num_obs_prueba = tf.cast(num_obs_prueba, tf.int64)

def normalizar(imagen, etiqueta):
    imagen = tf.cast(imagen, tf.float32)
    imagen /= 255.
    return imagen, etiqueta
```

```
datos_entrenamiento_y_validacion_normalizados = entreno_mnist.map(normalizar)
datos_prueba = prueba_mnist.map(normalizar)
TAMANIO_BUFFER = 10000
datos_entrenamiento_y_validacion_barajeados = datos_entrenamiento_y_validacion_r
datos_validacion = datos_entrenamiento_y_validacion_barajeados.take(num_obs_vali
datos_entreno = datos_entrenamiento_y_validacion_barajeados.skip(num_obs_validac
TAMANIO_TANDA = 25

datos_entreno = datos_entreno.batch(TAMANIO_TANDA)

datos_validacion = datos_validacion.batch(num_obs_validacion)

datos_prueba = datos_prueba.batch(num_obs_prueba)
entradas_validacion, metas_validacion = next(iter(datos_validacion))

tamanio_entrada = 784
tamanio_salida = 10
```

In [ ]:
```
modelo = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(192, activation='sigmoid'),
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='relu'),
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='tanh'),
    #tf.keras.layers.Dense(tamanio_capa_escondida, activation='sigmoid'),
    tf.keras.layers.Dense(tamanio_salida, activation='softmax')
])
modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005) , loss='

NUMERO_EPOCAS = 5
```

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

```
Epoch 1/5
2160/2160 - 12s - loss: 0.2701 - accuracy: 0.9238 - val_loss: 0.1337 - val_accura
cy: 0.9613 - 12s/epoch - 6ms/step
Epoch 2/5
2160/2160 - 10s - loss: 0.0970 - accuracy: 0.9708 - val_loss: 0.1040 - val_accura
cy: 0.9690 - 10s/epoch - 5ms/step
Epoch 3/5
2160/2160 - 11s - loss: 0.0638 - accuracy: 0.9805 - val_loss: 0.0673 - val_accura
cy: 0.9797 - 11s/epoch - 5ms/step
Epoch 4/5
2160/2160 - 11s - loss: 0.0475 - accuracy: 0.9853 - val_loss: 0.0540 - val_accura
cy: 0.9865 - 11s/epoch - 5ms/step
Epoch 5/5
2160/2160 - 10s - loss: 0.0360 - accuracy: 0.9887 - val_loss: 0.0601 - val_accura
cy: 0.9838 - 10s/epoch - 5ms/step
```

Out[ ]: <keras.callbacks.History at 0x24e21508580>

In [ ]:
```
perdida_prueba, precision_prueba = modelo.evaluate(datos_prueba)
print('Pérdida de prueba: {0:.2f}. Precisión de prueba: {1:.2f}%'.format(perdida
```

```
1/1 [==============================] - 1s 779ms/step - loss: 0.0812 - accuracy:
0.9771
Pérdida de prueba: 0.08. Precisión de prueba: 97.71%
```

Respuestas

- Después de múltiples intentos con diferentes acercamientos, utilizando cambios en las funciones de activación, el ancho de las capas, el número de capas, el tamaño de la tanda y la tasa de aprendizaje, el valor que más se acercó fue 97.71% con una tasa de aprendizaje de 0.0005, 3 capas ocultas de tamaños variados, tres funciones de activación diferentes y un tamaño de tanda de 25.
- No se pudo llegar al 98.5% de precisión, pero se logró un 97.71% con una tasa de aprendizaje de 0.0005, 3 capas ocultas de tamaños variados, tres funciones de activación diferentes y un tamaño de tanda de 25.