### MINST

En esta sección se empleará el conjunto de datos MINST para aplicar la optimización bayesiana para la busqueda de los hiperparámetros de una red neuronal (NN).

\*\*Definición del problema\*\*

El problema consiste en clasificar imágenes de dígitos escritos a mano en el conjunto de datos MINST. Cada imagen es de 28x28 píxeles y contiene un dígito del 0 al 9. El objetivo, es entrenar una red neuronal para predecir el dígito correcto basándose en las características de la imagen.

\*\*Planteamiento matemático del problema\*\*

El problema de clasificación se puede modelar como una tarea de aprendizaje supervisado, donde se busca encontrar una función $f(X)$ que mapee las características de la imagen $X$ a la etiqueta correcta $Y$. Matemáticamente, esto se puede expresar como:

$$

Y = f(X; \Theta) + \epsilon

$$

Donde:

- $Y \in \{0, 1, 2, ..., 9\}$ representa la etiqueta del dígito.

- $X \in \mathbb{R}^{28 \times 28}$ es la matriz de píxeles de la imagen.

- $\Theta$ son los hiperparámetros de la red neuronal.

- $\epsilon$ es el término de error (ruido) que modela la variabilidad no explicada por $f(X; \Theta)$.

El objetivo es encontrar los hiperparámetros $\Theta^\*$ que minimicen el error de clasificación en el conjunto de datos MINST.

\*\*Hiperparámetros optimizados\*\*

Los hiperparámetros a optimizar en la red neuronal incluyen:

- Número de capas ocultas.

- Número de neuronas por capa.

- Tasa de aprendizaje.

- Número de épocas de entrenamiento.

- Función de activación.

\*\*Enfoque de optimización\*\*

Para optimizar los hiperparámetros de la red neuronal, se utilizará la optimización bayesiana. La optimización bayesiana construye un modelo probabilístico basado en procesos gaussianos para aproximar la relación entre los hiperparámetros y la función objetivo (precisión de la red neuronal). La técnica guía la búsqueda hacia regiones prometedoras del espacio de hiperparámetros.

1. \*\*Modelo probabilístico\*\*:

El modelo probabilístico de la precisión se denota como:

$$

f(\Theta) \sim \mathcal{GP}(\mu(\Theta), k(\Theta, \Theta'))

$$

Donde:

- $\mu(\Theta)$: Media predictiva de la precisión.

- $k(\Theta, \Theta')$: Función de covarianza que mide la similitud entre puntos $\Theta$ y $\Theta'$.

2. \*\*Función de adquisición\*\*:

La función de adquisición (Expected Improvement, EI) guía la selección del próximo conjunto de hiperparámetros a evaluar:

$$

\text{EI}(\Theta) = \mathbb{E}\left[ \max(0, f(\Theta^\*) - f(\Theta)) \right]

$$

Donde:

- $f(\Theta^\*)$ es el mejor valor conocido de la función objetivo.

- $f(\Theta)$ es el valor predicho por el modelo probabilístico.

3. \*\*Actualización iterativa\*\*:

En cada iteración:

- Se evalúa $\text{EI}(\Theta)$ en el espacio de búsqueda.

- Se selecciona el conjunto de hiperparámetros $\Theta\_{\text{next}}$ que maximiza $\text{EI}(\Theta)$.

- Se entrena la red neuronal con $\Theta\_{\text{next}}$ y se actualiza el modelo probabilístico.

Se utiliza la librería \*tensorflow\* para definir la NN. Todas las imágenes están etiquetadas y por lo tanto, se puede aplicar una estrategia de entrenamiento supervisado.

Se importa la librería \*GPyOpt\* para optimizar los hiperparámetros utilizando el \*Proceso Gaussiano\*.

GPyOpt es una librería de Python para la Optimización Bayesiana usando Procesos Gaussianos (GPs). Está diseñada para ayudar a optimizar funciones costosas de evaluar, especialmente aquellas que no son convexas, ruidosas o de caja negra (en las que se desconoce el mecanismo interno de la función). La biblioteca aprovecha las GP para modelar la función objetivo y proporciona métodos eficientes para explorar el espacio de búsqueda.

```{python modelo\_MNIST, echo=FALSE}

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

from sklearn.metrics import accuracy\_score

from GPyOpt.methods import BayesianOptimization

# Cargar el conjunto de datos MNIST

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# Normalizar las imágenes

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Aplanar las imágenes

X\_train = X\_train.reshape(-1, 28\*28)

X\_test = X\_test.reshape(-1, 28\*28)

# Definir la arquitectura de la red neuronal

def build\_model(n\_hidden, n\_neurons, learning\_rate, activation):

model = models.Sequential()

model.add(layers.InputLayer(input\_shape=(28\*28,)))

for \_ in range(n\_hidden):

model.add(layers.Dense(n\_neurons, activation=activation))

model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

return model

# Definir la función objetivo para la optimización

def nn\_score(n\_hidden, n\_neurons, learning\_rate, activation):

model = build\_model(int(n\_hidden), int(n\_neurons), learning\_rate, activation)

model.fit(X\_train, y\_train, epochs=5, verbose=0)

y\_pred = model.predict(X\_test)

return -accuracy\_score(y\_test, y\_pred.argmax(axis=1))

# Definir el espacio de búsqueda para la optimización

bds\_mnist = [

{'name': 'n\_hidden', 'type': 'discrete', 'domain': (1, 2, 3)},

{'name': 'n\_neurons', 'type': 'discrete', 'domain': (64, 128, 256)},

{'name': 'learning\_rate', 'type': 'continuous', 'domain': (1e-5, 1e-1)},

{'name': 'activation', 'type': 'discrete', 'domain': ('relu', 'tanh', 'sigmoid')}

]

# Configurar el optimizador bayesiano

optimizer\_mnist = BayesianOptimization(

f=nn\_score,

domain=bds\_mnist,

model\_type='GP',

acquisition\_type='EI',

acquisition\_jitter=0.05,

exact\_feval=True,

maximize=False # Minimizar el error de clasificación

)

# Ejecutar la optimización (20 iteraciones)

optimizer\_mnist.run\_optimization(max\_iter=20)

# Mostrar los mejores resultados

#print("Mejores hiperparámetros encontrados:")

#print(f"Número de capas ocultas: {int(optimizer\_mnist.X[np.argmin(optimizer\_mnist.Y), 0])}")

#print(f"Número de neuronas por capa: {int(optimizer\_mnist.X[np.argmin(optimizer\_mnist.Y), 1])}")

#print(f"Tasa de aprendizaje: {optimizer\_mnist.X[np.argmin(optimizer\_mnist.Y), 2]:.5f}")

#print(f"Función de activación: {['relu', 'tanh', 'sigmoid'][int(optimizer\_mnist.X[np.argmin(optimizer\_mnist.Y), 3])]}")

#print(f"Mejor precisión (negativa): {np.min(optimizer\_mnist.Y):.4f}")

```