OPTIMIZADORES

1. SGD (Stochastic Gradient Descent)

Código python: optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

- **Descripción**: Es el optimizador más básico, ajusta los pesos del modelo gradualmente usando un solo ejemplo o un pequeño subconjunto de los datos de entrenamiento en cada paso.
- **Learning rate**: En SGD, la tasa de aprendizaje es fundamental, ya que determina qué tan grandes son los pasos que da el modelo al ajustar los pesos.
    - **Valor típico**: learning_rate=0.01.
    - **Explicación**: Un valor demasiado grande podría hacer que el modelo no converja, saltando sobre los mínimos locales, mientras que un valor demasiado pequeño puede hacer que el entrenamiento sea muy lento.
    - **Otros valores**:
        - Puedes probar con **0.001** para un entrenamiento más lento pero más preciso.
        - Un valor más alto, como **0.1**, podría ser útil si los datos son simples y se quiere una convergencia rápida, pero se corre el riesgo de oscilaciones.
- **Ventajas**: Es simple y eficiente para problemas pequeños.
- **Cuándo usarlo**: Para problemas donde se necesita mucho control sobre el ajuste de los pesos o cuando tienes un dataset balanceado.
- **Contras**: Converge lentamente y puede quedarse atrapado en mínimos locales.
- **Una variante es SGD con momentum**:

  Código Python: optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

    - **Momentum**: Ayuda a acelerar el entrenamiento y a evitar quedarse en mínimos locales. **momentum=0.9** es un valor común, pero valores entre **0.5** y **0.99** pueden ser probados dependiendo de los datos.

2. Adam (Adaptive Moment Estimation)

Código python: optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

- **Descripción**: Combina las ventajas de los optimizadores Adagrad y RMSProp, adaptando la tasa de aprendizaje para cada parámetro de forma automática.
- **Learning rate**:
    - **Valor típico**: learning_rate=0.001.
    - **Explicación**: Este valor por defecto es generalmente adecuado para la mayoría de los problemas, ya que Adam ajusta la tasa de aprendizaje de manera adaptativa. Un valor más bajo puede mejorar la exactitud pero hacer que el entrenamiento sea más lento.
    - **Otros valores**:
        - **0.0001** puede ser útil cuando el modelo tiene problemas de sobreajuste, ya que reduce el tamaño de los pasos y puede mejorar la precisión.
        - **0.01** puede ser usado si se requiere una convergencia rápida en un problema sencillo, aunque esto aumenta el riesgo de saltar sobre mínimos locales.
- **Ventajas**: Funciona bien en muchos problemas y ajusta los parámetros de manera eficiente.
- **Cuándo usarlo**: Es generalmente la primera opción en la mayoría de los problemas de redes neuronales profundas.
- **Contras**: En algunos casos, puede llevar al sobreajuste si la tasa de aprendizaje no está ajustada correctamente.

3. RMSProp (Root Mean Square Propagation)

Código python: optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)

- **Descripción**: Similar a Adam, RMSProp ajusta la tasa de aprendizaje basándose en las medias cuadráticas de los gradientes recientes.
- **Learning rate**:
  - **Valor típico**: learning_rate=0.001.
  - **Explicación**: Este es un valor seguro para la mayoría de los problemas, especialmente en redes recurrentes (RNNs) y series temporales. Funciona bien para tareas donde los gradientes varían mucho.
  - **Otros valores**:
    - **0.0001** puede ser utilizado para un ajuste más preciso en problemas donde se observan oscilaciones durante el entrenamiento.
    - **0.01** podría funcionar si se quiere una convergencia rápida, aunque esto puede hacer que el modelo oscile si la tasa es demasiado alta.
- **Ventajas**: Funciona muy bien en tareas de **series temporales** y procesamiento de secuencias.
- **Cuándo usarlo**: Cuando se entrenan **RNNs**, problemas de series temporales, o cuando los gradientes varían mucho.
- **Contras**: Similar a Adam, si la tasa de aprendizaje no está bien ajustada, puede no converger.

4. Adagrad (Adaptive Gradient)

Código Python: optimizer = tf.keras.optimizers.Adagrad(learning_rate=0.01)

- **Descripción**: Adagrad ajusta la tasa de aprendizaje adaptativamente, de forma que da pasos más pequeños para características que se actualizan frecuentemente.
- **Learning rate**:
  - **Valor típico**: learning_rate=0.01.
  - **Explicación**: Debido a que Adagrad ajusta automáticamente la tasa de aprendizaje para cada parámetro, puedes usar un valor inicial relativamente alto como **0.01**.
  - **Otros valores**:
    - **0.001** puede ser útil si la convergencia es muy rápida o si observas que el modelo está dejando de aprender demasiado pronto.
- **Ventajas**: Funciona muy bien en **datos dispersos**, como los problemas de clasificación de texto o procesamiento de secuencias donde ciertas características son raras.
- **Cuándo usarlo**: Ideal para **datasets dispersos** como los usados en **procesamiento del lenguaje natural**.
- **Contras**: La tasa de aprendizaje disminuye con el tiempo, lo que puede hacer que el modelo deje de aprender demasiado pronto.

5. Adadelta

Código Python: optimizer = tf.keras.optimizers.Adadelta(learning_rate=1.0)

- **Descripción**: Adadelta es una extensión de Adagrad que corrige la disminución excesiva de la tasa de aprendizaje.
- **Learning rate**:
  - **Valor típico**: learning_rate=1.0.

    - **Explicación**: Adadelta ajusta la tasa de aprendizaje automáticamente, por lo que se suele usar un valor más alto como **1.0**. Sin embargo, en algunos casos puedes bajar el valor si observas que el modelo no converge.
    - **Otros valores**:
        - **0.1** puede ser útil si se observa que el modelo no está aprendiendo bien con una tasa de aprendizaje mayor.
- **Ventajas**: No necesita ajuste manual de la tasa de aprendizaje.
- **Cuándo usarlo**: Útil cuando no quieres preocuparte por ajustar manualmente la tasa de aprendizaje.
- **Contras**: Menos popular que Adam, aunque puede funcionar bien en ciertos problemas.

6. Adamax

Código python: optimizer = tf.keras.optimizers.Adamax(learning_rate=0.002)

- **Descripción**: Adamax es una variante de Adam basada en la norma infinita y funciona bien con datos grandes.
- **Learning rate**:
    - **Valor típico**: learning_rate=0.002.
    - **Explicación**: **0.002** es el valor recomendado, ya que Adamax es más estable en problemas con grandes datasets y puede beneficiarse de un paso un poco más grande que el utilizado en Adam.
    - **Otros valores**:
        - **0.001** para un ajuste más preciso en problemas con gran variabilidad en los datos.
- **Ventajas**: Funciona bien en redes neuronales profundas y grandes datasets.
- **Cuándo usarlo**: Cuando tienes una red profunda o grandes cantidades de datos.
- **Contras**: Puede no ser necesario a menos que estés trabajando con modelos complejos y grandes datasets.

7. Nadam (Nesterov-accelerated Adaptive Moment Estimation)

Código Python: optimizer = tf.keras.optimizers.Nadam(learning_rate=0.001)

- **Descripción**: Es una variante de Adam que incorpora las correcciones de Nesterov, lo que ayuda a anticipar los cambios futuros de los gradientes.
- **Learning rate**:
    - **Valor típico**: learning_rate=0.001.
    - **Explicación**: Este valor suele ser suficiente para la mayoría de los casos, ya que Nadam ajusta la tasa de aprendizaje de manera eficiente.
    - **Otros valores**:
        - **0.0001** si el modelo está sobreajustándose o si se quiere una convergencia más precisa.
        - **0.01** puede funcionar si se necesita una convergencia rápida.
- **Ventajas**: Convergencia más rápida que Adam en algunos casos, gracias a la técnica de Nesterov.
- **Cuándo usarlo**: Cuando Adam no está funcionando bien o se requiere una optimización más rápida.
- **Contras**: Las diferencias frente a Adam pueden ser mínimas en muchos casos.