Funciones de perdida (Loss)

En redes neuronales, la **función de pérdida** mide qué tan bien el modelo está funcionando, comparando las predicciones con los valores reales. Dependiendo del tipo de tarea (clasificación, regresión, etc.), es importante elegir la función de pérdida adecuada. A continuación se explican los tipos más comunes de funciones de pérdida y cuándo aplicarlas:

1. Mean Squared Error (MSE)

- **Descripción**: Calcula la media de los errores al cuadrado entre las predicciones del modelo y los valores reales.
- **Fórmula**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Donde y_i es el valor real y $\hat{y}_i$ es el valor predicho.

- **Cuándo usarla**: Para problemas de **regresión**, donde la salida es un valor continuo. Ejemplos incluyen predecir precios, temperaturas, o cualquier otro valor numérico.
- **Ejemplo**:

  Código Python: loss = 'mean_squared_error'

- **Ventajas**: Penaliza fuertemente los errores grandes debido a que los errores se elevan al cuadrado, lo cual es útil cuando queremos minimizar grandes diferencias entre la predicción y el valor real.
- **Contras**: Sensible a valores atípicos (outliers), ya que un solo error grande puede afectar significativamente la pérdida total.

2. Mean Absolute Error (MAE)

- **Descripción**: Calcula la media de los errores absolutos entre las predicciones y los valores reales.
- **Fórmula**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Cuándo usarla**: Para problemas de **regresión** donde no quieres que los errores grandes afecten tanto el modelo. MAE es más robusto a los outliers que MSE.
- **Ejemplo**:

  Código Python: loss = 'mean_absolute_error'

- **Ventajas**: Menos sensible a los outliers que MSE, ya que no eleva los errores al cuadrado.

- **Contras**: Puede no penalizar suficientemente los errores grandes en comparación con MSE.

3. Huber Loss

- **Descripción**: Es una mezcla entre MSE y MAE. Utiliza MSE para errores pequeños y MAE para errores grandes, lo que lo hace más robusto a outliers que MSE, pero aún penaliza errores pequeños más que MAE.
- **Fórmula**:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{si } |y - \hat{y}| \leq \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{si } |y - \hat{y}| > \delta \end{cases}$$

- **Cuándo usarla**: Para problemas de **regresión** donde quieres un equilibrio entre penalizar grandes errores y ser más tolerante a outliers.
- **Ejemplo**:

  Código Python: loss = tf.keras.losses.Huber(delta=1.0)

- **Ventajas**: Proporciona un buen balance entre MSE y MAE.
- **Contras**: Requiere ajustar el valor de delta manualmente, dependiendo del problema.

4. Categorical Crossentropy

- **Descripción**: Mide la diferencia entre dos distribuciones de probabilidad. Se usa cuando la salida es una **distribución categórica**.
- **Fórmula**:

$$L = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

Donde y_i es la etiqueta verdadera y $\hat{y}_i$ es la probabilidad predicha por el modelo para la clase correcta.

- **Cuándo usarla**: Para problemas de **clasificación multiclase** donde las etiquetas están **one-hot encoded**. Esto es típico en problemas como clasificación de imágenes con más de dos clases (por ejemplo, MNIST, CIFAR-10).
- **Ejemplo**:

  Código Python: loss = 'categorical_crossentropy'

- **Ventajas**: Funciona bien para tareas de clasificación con muchas clases. Es sensible a cambios pequeños en las probabilidades predichas.
- **Contras**: Las etiquetas deben estar **one-hot encoded**, lo que significa que para cada muestra, la etiqueta es un vector en el que solo una posición tiene valor 1.

5. Sparse Categorical Crossentropy

- **Descripción**: Similar a Categorical Crossentropy, pero en lugar de usar etiquetas one-hot encoded, se utiliza un entero que representa la clase correcta.
- **Cuándo usarla**: Para problemas de **clasificación multiclase** donde las etiquetas no están **one-hot encoded**, sino que son simplemente enteros que representan la clase correcta. Por ejemplo, en MNIST, donde las etiquetas son números enteros entre 0 y 9.
- **Ejemplo**:

  Código Python: loss = 'sparse_categorical_crossentropy'

- **Ventajas**: Ahorra espacio y es más eficiente que categorical_crossentropy porque no requiere que las etiquetas estén one-hot encoded.
- **Contras**: Solo es aplicable cuando las etiquetas son enteros.

6. Binary Crossentropy

- **Descripción**: Es una versión simplificada de Categorical Crossentropy que se usa en problemas de **clasificación binaria** (dos clases).
- **Fórmula**:

$$L = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

Donde y es la etiqueta verdadera y $\hat{y}$ es la probabilidad predicha para la clase positiva.

- **Cuándo usarla**: Para problemas de **clasificación binaria** (por ejemplo, clasificación de imágenes como "gato" vs "no gato").
- **Ejemplo**:

  Código Python: loss = 'binary_crossentropy'

- **Ventajas**: Funciona bien para problemas de clasificación binaria y proporciona probabilidades en la salida.
- **Contras**: Solo es aplicable cuando hay dos clases.

7. Kullback-Leibler Divergence (KL Divergence)

- **Descripción**: Mide la diferencia entre dos distribuciones de probabilidad, tratando de minimizar la divergencia entre la distribución predicha y la verdadera.
- **Fórmula**:

$$D_{KL}(P||Q) = \sum_{i=1}^{n} P(i) \log \frac{P(i)}{Q(i)}$$

Donde $P(i)$ es la distribución verdadera y $Q(i)$ es la distribución predicha por el modelo.

- **Cuándo usarla**: Para problemas donde la salida es una **distribución de probabilidad** y no un valor discreto o continuo. Por ejemplo, en modelos de autoencoders variacionales (VAEs) o en tareas que involucran distribuciones de probabilidad.
- **Ejemplo**:

Código Python: loss = 'kullback_leibler_divergence'

- **Ventajas**: Muy útil para problemas de modelado de distribuciones.
- **Contras**: Puede ser más complejo de interpretar y aplicar en problemas típicos de clasificación o regresión.

8. Poisson

- **Descripción**: Mide la discrepancia entre la predicción del modelo y los valores reales asumiendo que los datos siguen una distribución de Poisson.
- **Fórmula**:

$$L = \hat{y} - y \log(\hat{y})$$

- **Cuándo usarla**: Para problemas donde los datos siguen una **distribución de Poisson**, como en conteo de eventos. Por ejemplo, predecir el número de visitas a un sitio web en un día.
- **Ejemplo**:

Código Python: loss = 'poisson'

- **Ventajas**: Muy útil cuando los datos representan conteos de eventos y siguen una distribución de Poisson.
- **Contras**: Solo aplicable en contextos donde los datos siguen esta distribución.

9. Cosine Similarity

- **Descripción**: Mide la similitud entre dos vectores al calcular el coseno del ángulo entre ellos. Se usa para medir qué tan cercanas están dos distribuciones.
- **Fórmula**:

$$L = -\frac{\vec{y} \cdot \vec{\hat{y}}}{\|\vec{y}\| \|\vec{\hat{y}}\|}$$

- **Cuándo usarla**: Para tareas donde quieras maximizar la similitud entre las predicciones y las etiquetas, como en tareas de embeddings de texto o reconocimiento facial.
- **Ejemplo**:

Código Python: loss = 'cosine_similarity'

- **Ventajas**: Útil cuando trabajas con vectores de características en lugar de valores discretos o continuos.
- **Contras**: No mide el error directo entre los valores predichos y reales, sino la dirección de los vectores.

Resumen

1. **Regresión**:
    - **MSE** (cuando quieres penalizar grandes errores).

    - **MAE** (cuando quieres ser más robusto frente a outliers).
    - **Huber Loss** (para un equilibrio entre MSE y MAE).
2. **Clasificación**:
    - **Binary Crossentropy** (para problemas de clasificación binaria).
    - **Categorical Crossentropy** (para clasificación multiclase con etiquetas one-hot encoded).
    - **Sparse Categorical Crossentropy** (para clasificación multiclase sin etiquetas one-hot encoded).
3. **Distribuciones y Embeddings**:
    - **KL Divergence** (para distribuciones de probabilidad).
    - **Cosine Similarity** (para medir la similitud de vectores).