# Cross Entropy Loss Functions

• Binary Classification: Use **BinaryCrossentropy** if there are only two classes.

• Multi-Class Classification:

• • **CategoricalCrossentropy** for one-hot encoded labels or

• • **SparseCategoricalCrossentropy** for integer labels.

• Imbalanced Classes:

• • Focal Loss or

• • **a weighted version of**

• • **BinaryCrossentropy** or

• • **CategoricalCrossentropy**

• Probabilistic Output: Use KLDivergence if you’re working with soft probability distributions.

**1. Binary Cross-Entropy Loss (or Log Loss)**

• Function: tf.keras.losses.BinaryCrossentropy or tf.keras.losses.binary\_crossentropy

• Use Case: Binary classification tasks, where each sample belongs to one of two classes (e.g., 0 or 1).

• Input: Predicted values should be probabilities between 0 and 1, and true labels should be binary values.

• **Activation for Final Layer**: sigmoid

• **Explanation**: The sigmoid activation outputs a probability value between 0 and 1 for binary classification problems. The BinaryCrossentropy loss function expects this probability to calculate the loss.

• Equation: yi = y\_true pi = y\_pred

**2. Categorical Cross-Entropy Loss**

• Function: tf.keras.losses.CategoricalCrossentropy or tf.keras.losses.categorical\_crossentropy

• Use Case: Multi-class classification tasks, where each sample belongs to one of multiple classes, and the ***labels are one-hot encoded***.

• Input: Predicted values should be probabilities, and true labels should be one-hot encoded.

• **Activation for Final Layer**: softmax

• **Explanation**: The softmax activation outputs a probability distribution over multiple classes, where the sum of probabilities equals 1. This is suitable for multi-class classification with one-hot encoded labels.

• Equation:

**3. Sparse Categorical Cross-Entropy Loss**

• **Function**: tf.keras.losses.SparseCategoricalCrossentropy or tf.keras.losses.sparse\_categorical\_crossentropy

• **Use Case**: Multi-class classification tasks, where each sample belongs to one of multiple classes, and the ***labels are integers*** rather than one-hot encoded.

• **Input**: Predicted values should be probabilities, and true labels should be integers.

• **Activation for Final Layer**: softmax

• **Explanation**: Similar to CategoricalCrossentropy, softmax is used here, but the labels should be integers instead of one-hot encoded vectors.

**4. Kullback-Leibler Divergence (KL Divergence) Loss**

• Function: tf.keras.losses.KLDivergence or tf.keras.losses.kullback\_leibler\_divergence

• Use Case: Measures how one probability distribution diverges from a second, expected probability distribution. Useful for probabilistic outputs or when training with a soft label distribution.

• Input: Both true labels and predicted values should be probability distributions.

• **Activation for Final Layer**: softmax

• **Explanation**: KL Divergence measures the difference between two probability distributions. The softmax activation provides the model’s predicted probability distribution for each class.

• Equation:

**5. Poisson Loss**

• Function: tf.keras.losses.Poisson

• Use Case: Can be used when the output follows a Poisson distribution.

• Input: True labels and predictions should be positive values.

• **Activation for Final Layer**: linear (no activation) or exponential

• **Explanation**: The Poisson loss is used when the model outputs are expected to follow a Poisson distribution, typically for count data.

• Equation:

**6. Binary Focal Cross-Entropy Loss**

• Function: Not available directly in Keras. Implemented as a custom loss function.

• Use Case: Addresses class imbalance by assigning more focus to hard-to-classify samples.

• Input: Predicted values should be probabilities, and true labels should be binary values.

• **Activation for Final Layer**: sigmoid

• **Explanation**: Since Binary Focal Loss is derived from binary cross-entropy, it is designed to work with the sigmoid activation function.

• Equation:

where is the predicted probability, is a weighting factor, and is a focusing parameter.

def focal\_loss(gamma=2., alpha=0.25):

def focal\_loss\_fixed(y\_true, y\_pred):

epsilon = tf.keras.backend.epsilon()

y\_pred = tf.clip\_by\_value(y\_pred, epsilon, 1. - epsilon)

y\_true = tf.cast(y\_true, tf.float32)

alpha\_t = y\_true \* alpha + (tf.ones\_like(y\_true) - y\_true) \* (1 - alpha)

p\_t = y\_true \* y\_pred + (tf.ones\_like(y\_true) - y\_true) \* (1 - y\_pred)

fl = - alpha\_t \* tf.pow((tf.ones\_like(y\_true) - p\_t), gamma) \* tf.log(p\_t)

return tf.reduce\_mean(fl)

return focal\_loss\_fixed

model.compile(optimizer='adam', loss=focal\_loss())