**NAME: KINYUA EZRA MWANGI**

**REGISTRATION:CT201/102131/20**

**UNIT: NEURAL NETWORKS**

**COURSE: BCS**

1. **Regression Loss Function**

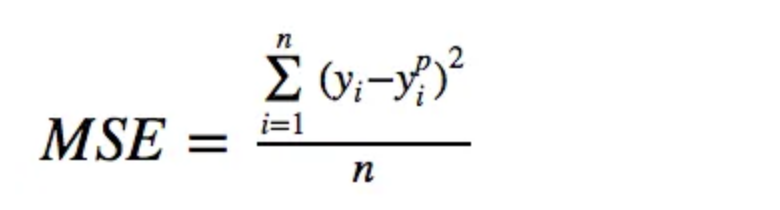
A regression loss function, often simply referred to as a "loss function" in the context of regression, is a mathematical function that quantifies the difference between predicted values generated by a regression model and the actual target values in a dataset. The purpose of a regression loss function is to measure how well the model's predictions align with the true values and to guide the training process by minimizing this difference. The choice of the loss function in regression is critical because it affects the model's ability to make accurate predictions and adapt its parameters during training. It is the one mostly used.

In regression loss function we have the following examples:

1. **Mean Square Error (MSE)**

This is also known as the L2 loss

It is expressed with the following equation:

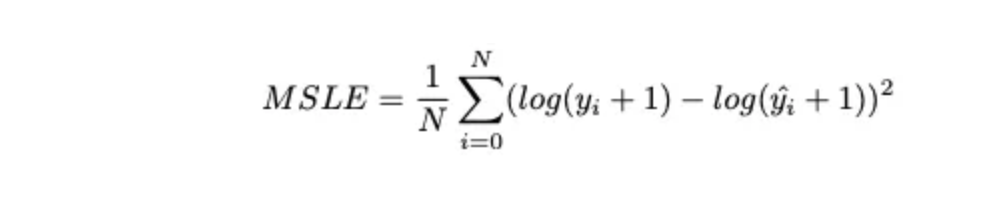


This loss function handles outliers in an efficient manner as outliers are detected due to the quadratic loss. Convergence is also smooth as the gradient becomes smaller as the loss decreases.

1. **Mean squared logarithmic loss**

The mean squared logarithmic error (MSLE) measures the difference between actual and expected values.

The equation is:



The loss function is scale independent as it is a difference of two log values which is the same as log of the ratio of the values. Due to the loss being log it penalizes underestimates more than overestimates

1. **Mean absolute error loss(MAE)**

MAE is robust to outliers because it doesn't square the differences, unlike Mean Squared Error (MSE). The loss is a measure of the absolute error, which means that errors in either direction (underestimating or overestimating) contribute equally.

It provides a straightforward and easily interpretable measure of prediction accuracy. The loss value directly represents the average magnitude of the prediction errors.Minimizing MAE leads to the median as the optimal prediction (the median minimizes the sum of absolute deviations).MAE loss is useful if the training data is corrupted with outliers (i.e. we erroneously receive unrealistically huge negative/positive values in our training environment, but not our testing environment).

1. **Binary classification loss functions**

Binary classification loss functions are used in machine learning to measure the difference between the predicted probabilities or scores generated by a binary classification model (which assigns data points to one of two classes) and the actual binary outcomes (e.g., 0 or 1). The choice of the loss function is crucial for training a binary classifier effectively. Here are two common binary classification loss functions:

Examples include:

1. **Binary cross-entropy**

Binary Cross-Entropy, also known as log loss, is one of the most widely used loss functions for binary classification.

It measures the dissimilarity between predicted probabilities and actual binary outcomes.

Log Loss = -[actual \* log(predicted) + (1 - actual) \* log(1 - predicted)]

Here, 'actual' is the true binary label (0 or 1), and 'predicted' is the predicted probability of the positive class (class 1).

1. **Hinge Loss**

· Hinge loss is commonly used in support vector machines (SVMs) for binary classification.

It encourages correct classification with a margin, and misclassified examples are penalized proportionally to how far they are from the correct side of the decision boundary.

Hinge Loss = max(0, 1 - actual \* predicted)

'actual' is either -1 (for the negative class) or 1 (for the positive class), and 'predicted' is the model's raw score.

1. **Squared Hinge Loss**

The Squared Hinge Loss is a variation of the Hinge Loss commonly used in binary classification problems. It encourages correct classification with a margin and penalizes misclassifications, but unlike the traditional Hinge Loss, it squares the hinge loss values, resulting in a smoother loss function. Here's the formula for the Squared Hinge Loss in binary classification:

Squared Hinge Loss = max(0, 1 - actual \* predicted)^2

Where:

actual is the true binary label (usually represented as -1 for the negative class and 1 for the positive class).

predicted is the raw model score or the predicted value for the positive class.

Here's how the loss function works:

If the actual and predicted values have the same sign, the term (1 - actual \* predicted) will be positive, and the loss is equal to the squared difference. Squaring it smoothens the loss function.

If the actual and predicted values have different signs, the term (1 - actual \* predicted) will be negative, and the loss is zero because max(0, negative)^2 is zero.

The Squared Hinge Loss aims to encourage a margin between the decision boundary and the data points, but it does not penalize misclassifications as severely as the traditional Hinge Loss. Squaring the loss makes it differentiable, which is beneficial for optimization algorithms in training machine learning models, particularly neural networks.

The Squared Hinge Loss can be used as a loss function when training binary classification models, and it provides a continuous and differentiable approximation to the Hinge Loss, making it easier to optimize with gradient-based methods.

1. **Multi class classification loss function.**

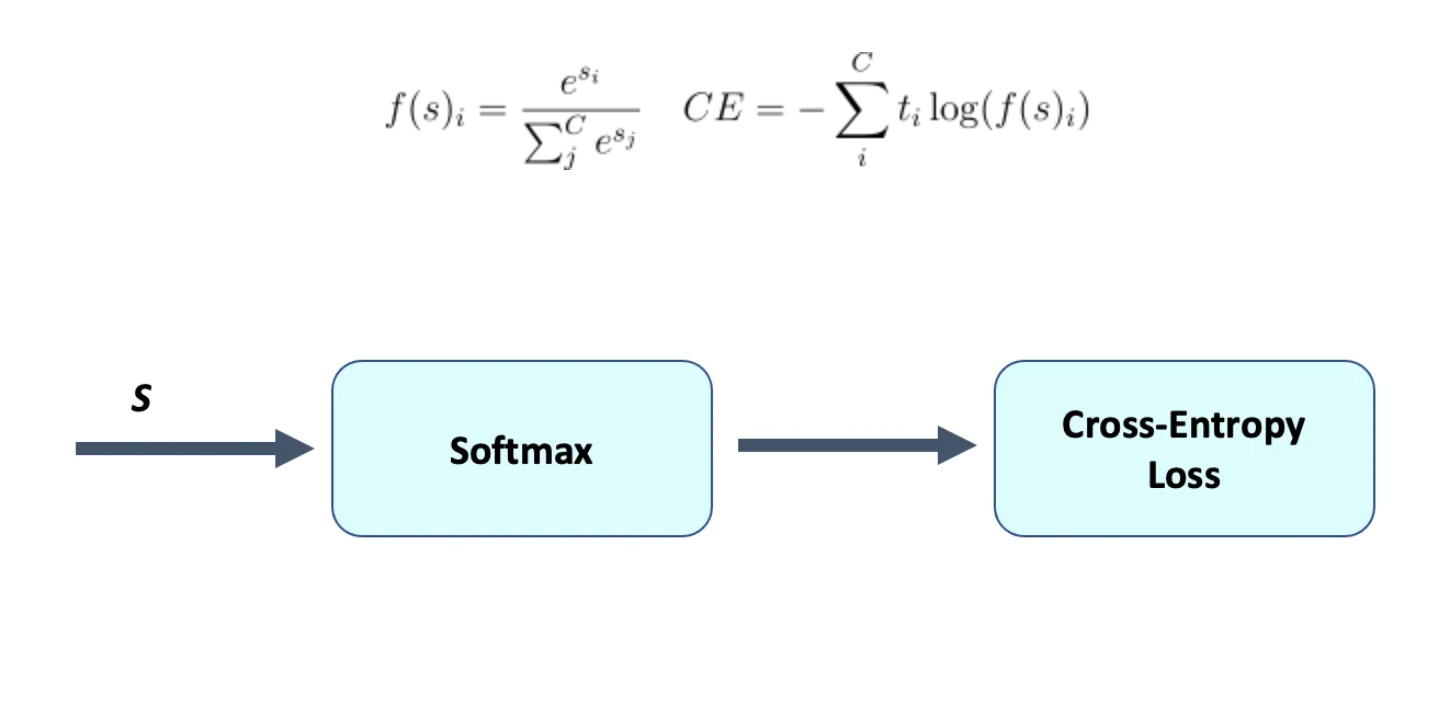
Multi-class classification loss functions are used in machine learning to measure the difference between the predicted class probabilities or scores generated by a model and the actual class labels when there are more than two classes. The choice of the loss function is crucial for effectively training a multi-class classifier. Here are some common multi-class classification loss functions:

Examples include:

1. **Multi-class cross-entropy loss**

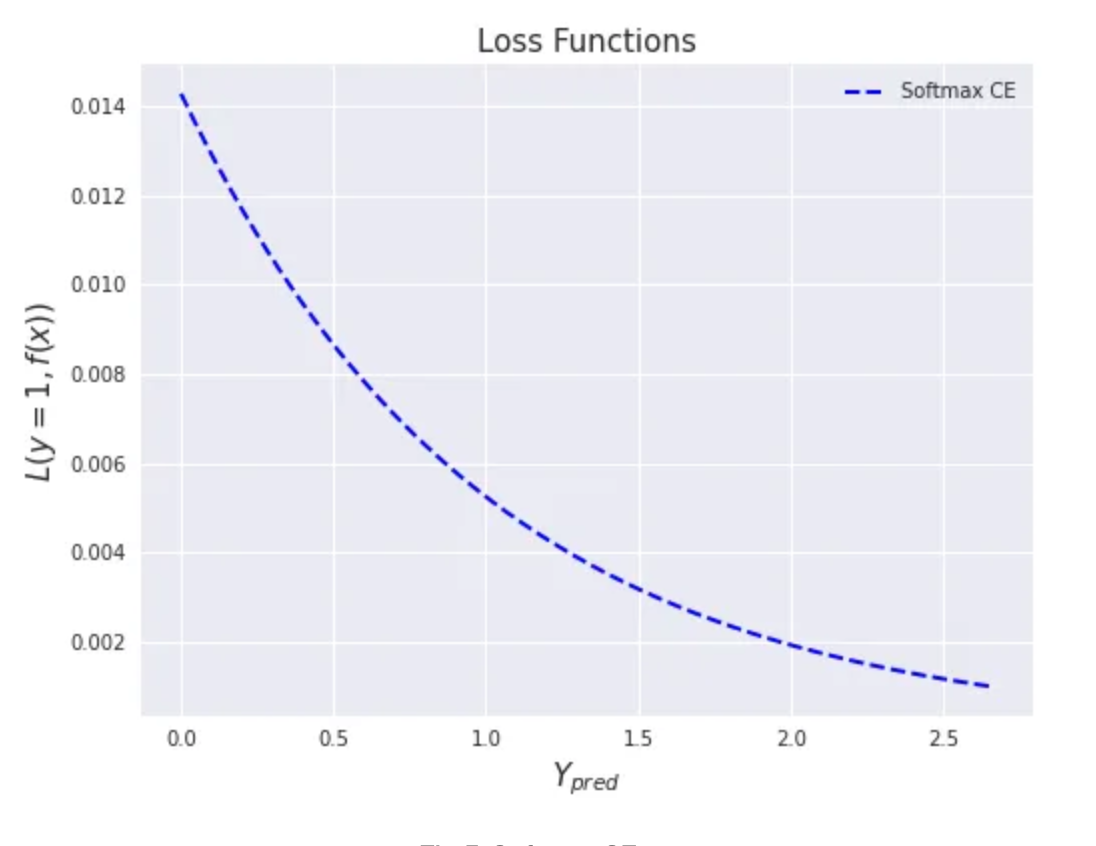
Instead of using sigmoid as the last layer activation, we use softmax for Categorical Cross-Entropy Loss.

The equation:



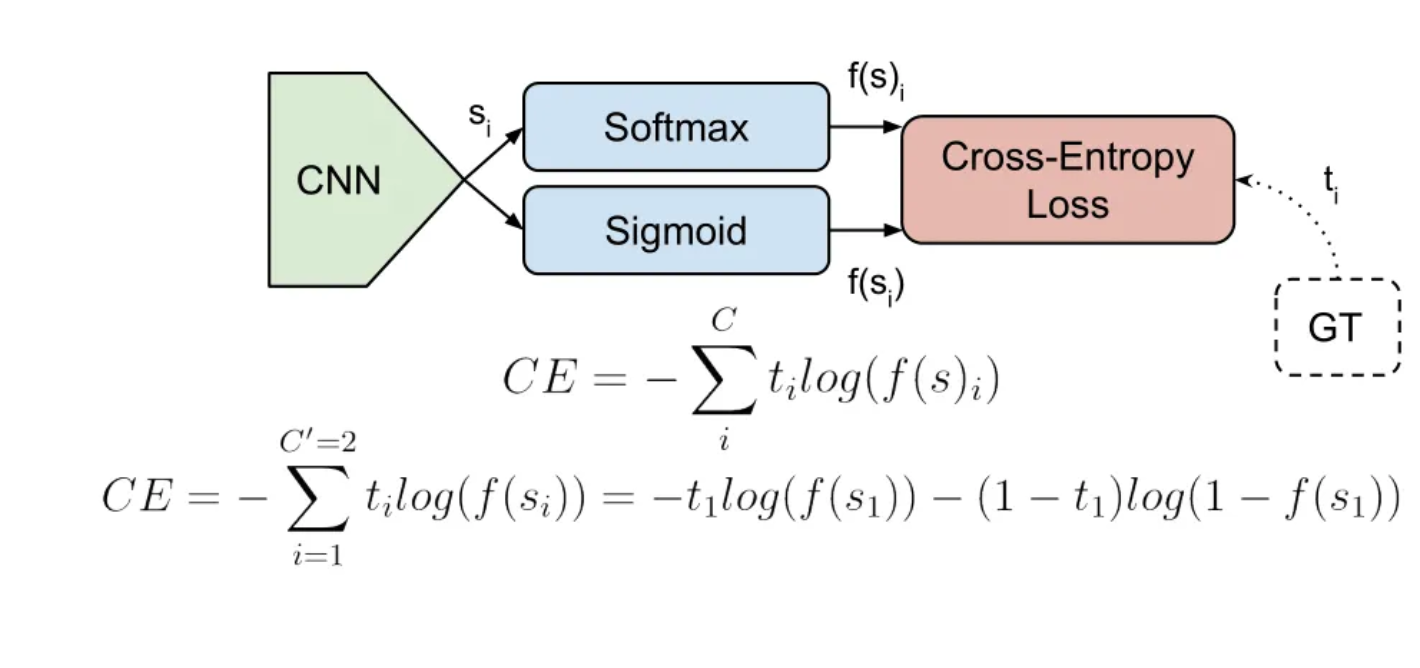
BCE\_list\_cat, loss = BinaryCrossEntropy(pred[:200].softmax(dim=0),target[:200])

The graphical representation is as follows:



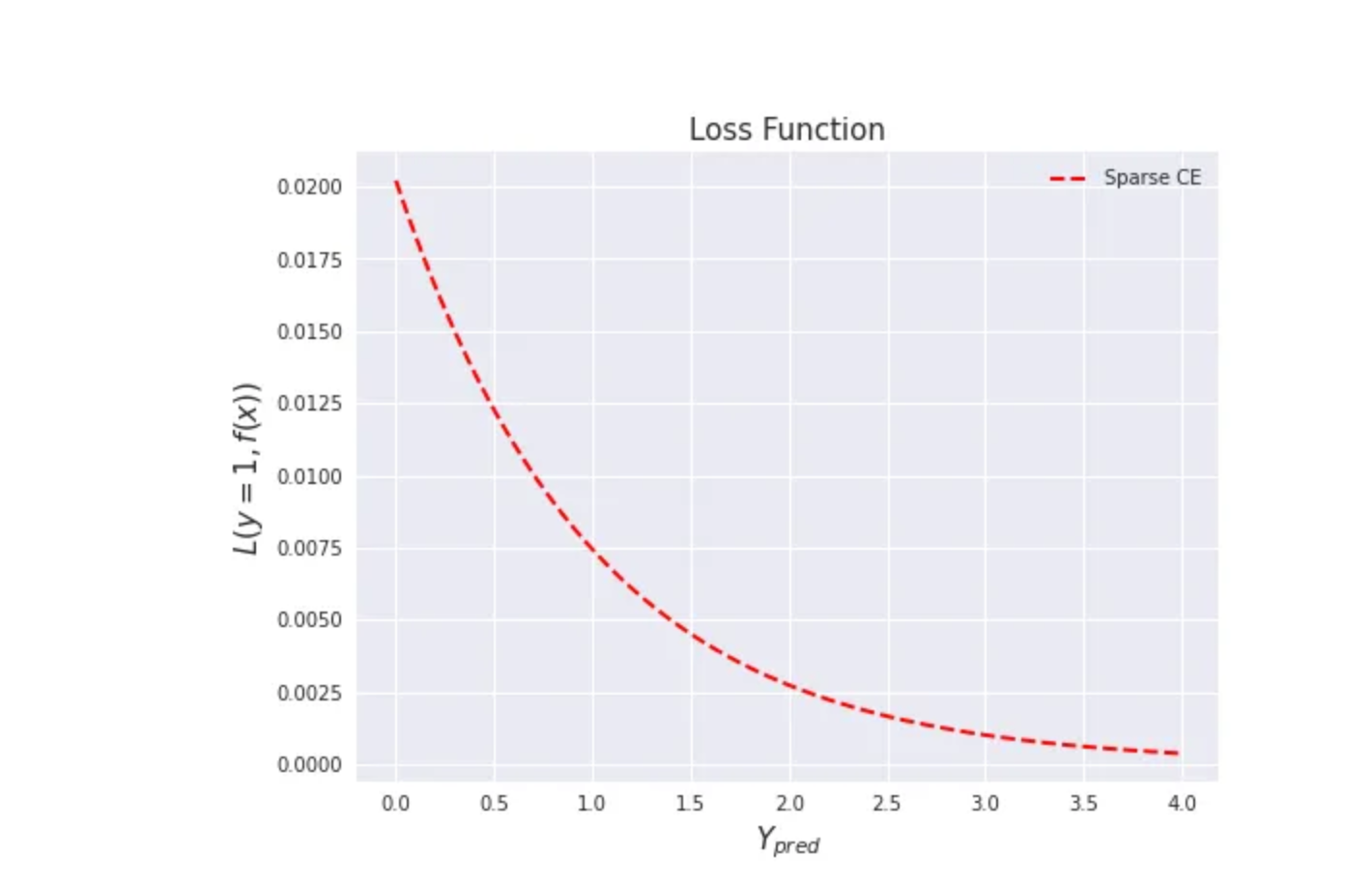
1. **Sparse multi-class cross-entropy loss**

This loss is the same as the softmax cross-entropy one, except instead of the target being a probability distribution, it is an index of which category is true. Instead of a sparse all-zero target vector with one value of one, we just pass in the index of which category is the true value, as follows:



m = nn.LogSoftmax(dim=0)  
  
criterion = nn.NLLLoss(reduction='none')  
  
x = torch.tensor(np.linspace(0,4,400).reshape(200,2), dtype=torch.float32)   
y = target[:200].long()#torch.ones(250, dtype=torch.long) # expect the target to be integer (Long)  
  
sparse\_loss = criterion(m(x), y)

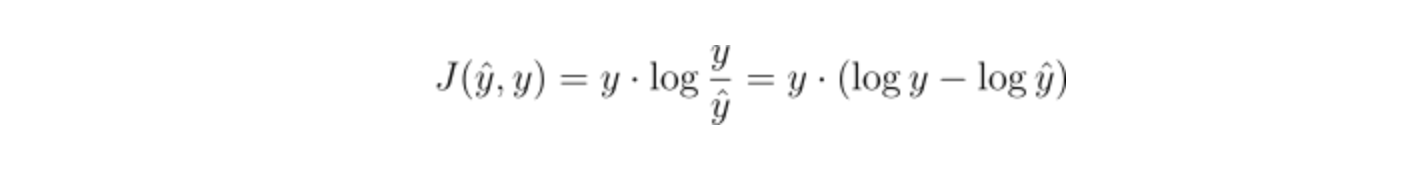
This is a graphical representation:



1. **Kullback leiber divergent loss.**

To avoid underflow issues when computing this quantity, this loss expects the argument input in the log-space. The argument target may also be provided in the log-space if log\_target= True.

Equation:



kl\_loss = torch.nn.KLDivLoss(reduction='none', log\_target=True) #specify whether the target is the log space  
# input should be a distribution in the log space  
input\_ = F.log\_softmax(pred[:200], dim=0)  
log\_target = F.log\_softmax(target[:200], dim=0)  
output = kl\_loss(input\_, log\_target)

Let’s visualize all the loss functions for multi-class classification!

The graphical representation is a s follows:

