Logits and BCE with Logits Loss – Explained

Mubashir Mohsin[1]
1 Department of Computer Science, AIUB, Dhaka-1229, Bangladesh

17 February 2025

Binary classification is a fundamental task in machine learning, often using the Binary Cross-Entropy, **`BCELoss()`** function combined with a Sigmoid activation. However, an alternative, **`BCEWithLogitsLoss()`**, provides better numerical stability and is preferred in many cases. In this article, we will try to understand logits and its relation to BCE loss calculations.

To make this topic easier to understand, I will explain it in two ways:

1. **A simple, non-technical manner**
2. **A technical and mathematically reasoned approach**

I. Logits and BCE Loss: Simple Explanation

What are Logits?

Logits are the raw outputs of a neural network before applying an activation function like Sigmoid or SoftMax. Unlike probabilities (which range from 0 to 1), logits can take any real value $(-\infty, +\infty)$.

How Logits Relate to BCE Loss?

We normally have multiple parameters inside an NN architecture with multiple different types of layers (e.g., weights & bias) connected to one another. While we train our models to learn to adjust the parameters, we send the parameters with a forward pass function where we use the activation function like sigmoid. The activation function is nothing but a normalizing function that represents a raw linear combination of inputs and learned weights (**logits**) into a probabilistic range (0 to 1).

So, when we use the BCE loss function, we technically apply the sigmoid function first to the final layer before calculating the loss in backpropagation. This means, when using BCE loss for a binary classification task, we must include a sigmoid activation function in the final layer to represent the logits into a range of probability.

Why do we apply Sigmoid?

But what will happen if we do not use an activation function? Well, the model's network could output values outside the normal range [0, 1] of probability and lead to negative logs while calculating loss while causing instability with gradients.

Why use Logits?

To simply put, logits are more expressive than a restricted range of probability [0, 1]. Logits provide an unrestricted range $(-\infty, +\infty)$, meaning a more expressive range of a parameter's learning pattern rather than a restricted range of (0, 1). Eventually, this avoids saturation later when we use sigmoid activation.

How are Logits useful?

We have already covered the first usefulness of logits above; now let's see how it helps future sigmoid activation later.

If we apply sigmoid activation before training (what we do in BCE Loss), we might face gradient vanishing issues (for more details on vanishing gradients, *click here*) when the network produces very high or very low values. For example:

> *If the logits range is very large, let's say (-100, 100), then the sigmoid function saturates to 0 or 1, leading to a near-zero gradient. Which by all means is a bad thing for our model. This leads to computational instability with gradients. Because computing BCE losses directly with probabilities can lead to undefined value or undefined results beyond the probability range.*

So, we could say logits are important for **three things** mainly:

- Logits are more expressive.
- Logits help avoid vanishing gradients.
- Logits are computationally stable.

Key Differences Between *BCEWithLogitsLoss()* and *BCELoss()*

Feature	`nn.BCELoss()`	`nn.BCEWithLogitsLoss()`
Input to loss function	Probabilities (0 to 1)	Logits (any real number)
Final layer requirement	Requires **Sigmoid** activation	No activation needed (Sigmoid applied internally)
Numerical stability	Less stable (prone to vanishing gradients)	More stable (avoids extreme log calculations)
Common use case	When manually applying Sigmoid	When using raw logits directly

***** If you are interested in exploring the mathematical reasons of logit and BCE loss, continue from here.***

II. Logits and BCE Loss: Mathematical Explanation

What are Logits?

Logits are linearly transformed outputs of the given inputs. During the forward pass, the parameters (weights and biases) are transformed into an unnormalized range $(-\infty, +\infty)$ of real numerical values. Unlike the probability range (0 to 1), logits are more expressive with a bigger range of numbers. Mathematically, an NN's final layer (before activation) computes a linear combination of the input parameters (weights and bias), which could be represented like this:

$$z = Wx + b$$

where ***W*** is the weight matrix, ***x*** is input features, and ***b*** is bias term. Logits can take any real value, positive or negative. They don't represent probabilities but rather unscaled scores.

How Logits Relate to BCE Loss?

Since logits are not probabilities, we typically apply the Sigmoid function to convert them into probabilities before using BCE loss:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the raw output (logits) and *σ(z) is* the probability prediction. **Basically, if we use sigmoid on logits, we transform that into probability**.

Why Use Logits Instead of Probabilities Directly?

i. Logits provide an unrestricted range $(-\infty, +\infty)$, while probabilities are constrained to [0,1]. This makes it easier for the model to learn without being limited to a fixed range during training.

ii. Logits can avoid saturation in the sigmoid function before the final layer. If we apply the sigmoid function before training, we may face gradient vanishing issues when the network produces very high or very low values. For example:
   - *If,* $z \gg 1$*, then* $\sigma(z) \approx 1$*, and* $\log(\sigma(z)) \approx 0$
   - *If,* $z \ll -1$*, then* $\sigma(z) \approx 0$*, and* $\log(1 - \sigma(z)) \approx 0$

   Here, ***σ*** is the activation and ***z*** is the scaler point per sample of the parameters (logits).

iii. Logits provide computational stability in calculating BCE Loss, because computing BCE loss directly with probabilities can lead to numerical instability. For instance:
   - If $\hat{y}$ (probability) is exactly 0 or 1, then $\log(\hat{y})$ or $\log(1 - \hat{y})$ will result in NaN or, undefined values.
   - Logits avoid this problem because they are transformed into stable log terms before exponentiation.

In both cases, gradients become very small, slowing down learning. Using logits allows the model to operate in an unconstrained space during training, improving gradient flow while learning.

How *BCEWithLogitsLoss()* Works?

Instead of explicitly applying Sigmoid before BCE loss, PyTorch's `BCEWithLogitsLoss()` internally applies the Sigmoid function in a numerically stable way.

The mathematical formulation of calculating BCE loss is:

$$\mathcal{L} = -y \cdot log(\sigma(z)) - (1 - y) \cdot log(1 - \sigma(z))$$

which is sensitive to floating-point precision. Therefore, PyTorch reformulates this as:

$$\mathcal{L} = max(z, 0) - zy + log(1 + e^{|-z|})$$

and this avoids exponentiating large values, preventing numerical instability.

When to Use Logits and Probabilities?

Scenario	Use Logits	Use Probabilities
Neural Network Training	✓ Yes (logits work better)	✗ No (vanishing gradient issue)
Output interpretation (inference)	✗ No	✓ Yes
Loss Function	`torch.nn.BCEWithLogitsLoss()`	`torch.nn.BCELoss()`

Example Codes

A) Using `BCELoss()` with Sigmoid Activation:

[1]:
```python
import torch
import torch.nn as nn

"""   Here, we manually apply torch.sigmoid().
      Without sigmoid, BCELoss() would fail.
"""

loss_fn = nn.BCELoss()

logits = torch.tensor([3.0, -1.5, 0.0])  # Raw scores (logits)
y_true = torch.tensor([1.0, 0.0, 1.0])          # Ground truth labels

# Print logits to show
print(f"Raw Logits Representation\t: {logits}")
# Convert logits to probabilities
probs = torch.sigmoid(logits)

# Compute BCE loss

loss = loss_fn(probs, y_true)
print(f"Probability Representation\t: {loss.item()}")  # Output BCE loss
```

```
Raw Logits Representation     : tensor([ 3.0000, -1.5000,  0.0000])
Probability Representation    : 0.3143825829029083
```

B) Using `BCEWithLogitsLoss()` (recommended):

```
[2]: import torch
     import torch.nn as nn

     """   No need for an explicit sigmoid activation.
           Works directly with logits.
           More stable numerically.
     """

     loss_fn = nn.BCEWithLogitsLoss()

     logits = torch.tensor([3.0, -1.5, 0.0])  # Raw scores (logits)

     y_true = torch.tensor([1.0, 0.0, 1.0])          # Ground truth labels

     # Directly pass logits (no sigmoid needed)

     loss = loss_fn(logits, y_true)
     print(f"Probability Representation\t: {loss.item()}")  # Output BCEWithLogitsLoss
```

```
       Probability Representation          : 0.3143825829029083
```

Final Thoughts and Summary

- Logits are raw model outputs before activation.
- They are numerically more stable than probabilities for loss computation.
- Sigmoid converts logits into probabilities for BCELoss().
- BCEWithLogitsLoss() internally applies sigmoid and is recommended for training.
- BCEWithLogitsLoss() is preferred for stability and better gradient flow.
- During inference, BCEWithLogitsLoss() might perform better.

<< The End >>