Analysis of Loss function

BY XIA PENGNING 12012608

2023/11/3

1 Introduction:

In the ever-evolving landscape of machine learning, the choice of loss function plays a pivotal role in shaping the performance of classification models. This report embarks on a journey to explore and compare the test performance of distinct loss functions in the realm of multi-class classification tasks. The selected list of loss functions includes the Mean Absolute Error (MAE or L1) loss, Cross-Entropy (CE) loss, and two variants of the Focal loss with different gamma values (gamma=0.5 and gamma=2). Each loss function embodies unique characteristics that can significantly influence the model's ability to discern and classify multiple classes accurately. Through systematic experimentation and analysis, we aim to unravel the nuanced impact of these loss functions, providing insights that can guide practitioners in choosing the most suitable approach for their specific classification challenges.

2 Different Loss Functions in Multiclass Classification Tasks

In the multifaceted domain of multi-class classification tasks, the choice of an appropriate loss function stands as a critical factor influencing the efficacy of machine learning models. Extensive literature has delved into the applications and implications of various loss functions, shedding light on their unique characteristics and performance nuances.

Mean Absolute Error (MAE) Loss: The MAE, or L1 loss, has found prominence in scenarios where the absolute differences between predicted and actual values hold significance. Its application in multi-class classification tasks is rooted in its ability to capture deviations without being overly sensitive to outliers, providing a robust metric for model evaluation.

$$L_{L1}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|$$

Cross-Entropy (CE) Loss: Cross-Entropy loss, a staple in classification problems, excels in scenarios where the model needs to assign probabilities to multiple classes. Its mathematical formulation encourages the model to minimize the difference between predicted probabilities and true class distributions, making it a preferred choice in the realm of multi-class classification.

$$L_{\mathrm{CE}}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i)$$

Focal Loss: The Focal loss introduces a dynamic weighting mechanism that addresses class imbalance issues. With a focus on difficult-to-classify samples, a Focal loss with gamma=0.5 tends to down-weight well-classified examples, providing a refined optimization approach for multi-class problems with imbalanced class distributions.

$$L_{\mathrm{Focl}}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} (1 - \hat{y}_i)^{\gamma} \cdot y_i \cdot \log(\hat{y}_i)$$

As we delve into the research landscape, it becomes evident that each loss function brings a unique set of advantages and considerations. The literature surveyed establishes a foundation for our exploration, guiding the formulation of experiments aimed at unraveling the comparative performance of these loss functions in multi-class classification tasks.

3 Reseach Design

In the pursuit of comparing the test performance of different loss functions in a multi-class classification task, a well-structured research design is imperative. The experiment is conducted using a Convolutional Neural Network (ConvNet) implemented in PyTorch. The chosen dataset for evaluation is the CIFAR-10 dataset, renowned for its diverse range of 10 classes, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

3.1 Data Preprocessing:

The CIFAR-10 dataset undergoes essential transformations for both training and testing sets. For training, data augmentation is employed, incorporating random crops and horizontal flips to enhance the model's robustness. All images are then normalized to ensure consistent feature scaling.

```
transform_cifar10_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_cifar10_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

train_set = torchvision.datasets.CIFAR10(root='../data', train=True,
                                         download=True,
transform=transform_cifar10_train)
train_dataloader = torch.utils.data.DataLoader(train_set,
batch_size=BATCH_SIZE,
                                           shuffle=True, num_workers=2)

test_set = torchvision.datasets.CIFAR10(root='../data', train=False,
                                        download=True,
transform=transform_cifar10_test)
test_dataloader = torch.utils.data.DataLoader(test_set, batch_size=BATCH_SIZE,
                                           shuffle=False, num_workers=2)
```

3.2 Model Architecture:

The ConvNet architecture is composed of two convolutional layers followed by max-pooling operations. The fully connected layers at the end serve as the classifier. The model is designed to capture hierarchical features through convolutional layers and make predictions for each of the ten classes in CIFAR-10.

```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 4, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(4, 8, 3)
        self.fc1 = nn.Linear(8 * 6 * 6, 32)
        self.fc2 = nn.Linear(32, 10)
```

```
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 8 * 6 * 6)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

3.3 Model Training and Optimization:

The model is trained using the Stochastic Gradient Descent (SGD) optimizer with a specified learning rate and momentum. A learning rate scheduler is incorporated to dynamically adjust the learning rate during training. This ensures an adaptive optimization process, potentially improving the model's convergence and performance.

```
model = ConvNet()
model.to(device)

optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE, momentum=MOMENTUM)

scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=STEP,
gamma=GAMMA)
```

This well-defined research design sets the stage for the systematic comparison of different loss functions in the subsequent experimentation phase.

4 Methods:

In this section, we delineate the methodology employed to conduct experiments comparing the test performance of different loss functions in the context of the provided code.

4.1 Loss Function Definitions:

Three distinct loss functions are employed for the evaluation: Cross-Entropy Loss (criterion1), Focal Loss with Gamma=0.5 (criterion2), and Focal Loss with Gamma=2 (criterion3). The Focal Loss introduces a dynamic weighting mechanism based on the difficulty of classifying instances, with different gamma values providing flexibility in emphasizing either easy or hard-to-classify samples.

```
class FocalLoss(nn.Module):
    def __init__(self, gamma=2):
        super(FocalLoss, self).__init__()
        self.gamma = gamma

    def forward(self, input, target):
        ce_loss = F.cross_entropy(input, target, reduction='none')
        pt = torch.exp(-ce_loss)
        focal_loss = (1 - pt) ** self.gamma * ce_loss
        return torch.mean(focal_loss)

criterion1 = nn.CrossEntropyLoss()
criterion2 = FocalLoss(gamma=0.5)
criterion3 = FocalLoss(gamma=2)
criterion4 = nn.L1Loss()
```

4.2 Training and Testing Procedures:

The training and testing procedures are orchestrated through the `train_batch` and `test_batch` functions. The model is trained using the specified loss function, and optimization is performed using the Stochastic Gradient Descent (SGD) optimizer. Learning rate scheduling is employed to adaptively adjust the learning rate during training.

```
def train_batch(model, image, target, criterion):
    output = model(image)
    loss = criterion(output, target)
    return output, loss

def test_batch(model, image, target, criterion):
    output = model(image)
    loss = criterion(output, target)
    return output, loss

# Training loop
for epoch in range(NUM_EPOCHS):
    model.train()
    # ... (Training procedure)

    # Testing loop
    if (epoch + 1) % EVAL_INTERVAL == 0 or (epoch + 1) == NUM_EPOCHS:
        model.eval()
        # ... (Testing procedure)
```

4.3 Evaluation Metrics:

Throughout the training and testing loops, metrics such as training loss, training accuracy, test loss, test accuracy, and F1 score are recorded for each epoch. F1 score, a metric that balances precision and recall, provides insights into the model's overall classification performance.

4.4 Results Logging:

The results of each epoch, including the aforementioned metrics, are stored in a DataFrame (`XX_results_df`). This DataFrame is periodically updated during training and is later saved to a CSV file (`XX_results.csv`) for comprehensive analysis and visualization.

```
# Focal loss with gamma=2 as example
results_df.to_csv('Focal2_results.csv', index=False)
```

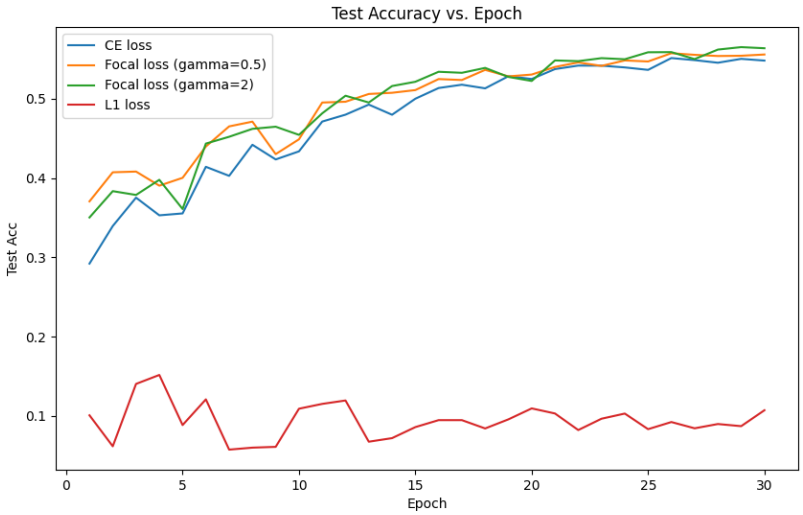
This comprehensive methodology ensures a systematic and thorough evaluation of the chosen loss functions, allowing for a nuanced understanding of their impact on the model's performance over the training epochs.

5 Results and Analysis:

This section delves into the results obtained from the experiments and provides an analysis of the performance of different loss functions in the multi-class classification task. The visualizations presented below are derived from the provided code, showcasing the trends in test accuracy, F1 score, and test loss across epochs for each loss function.

5.1 Test Accuracy

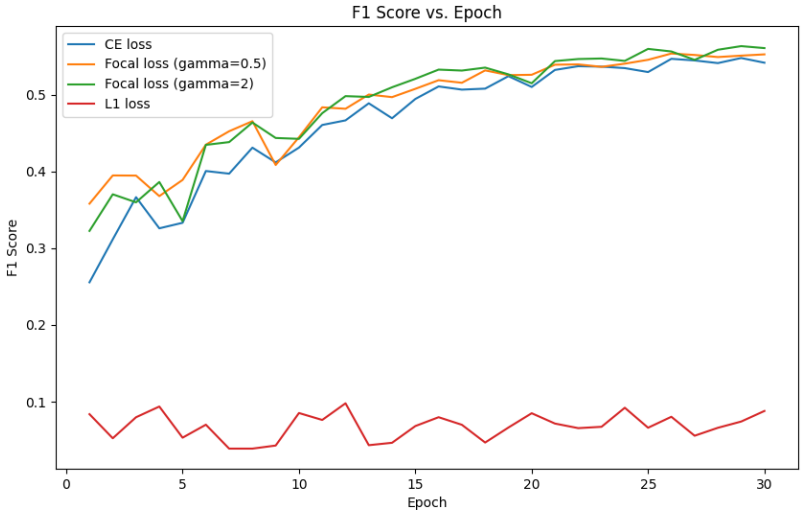
The following plot illustrates how the test accuracy evolves across training epochs for each loss function.



From this graph, it can be seen that as the number of epochs increases, the models trained with three out of four loss functions, excluding the L1 loss function, continuously converge, and their accuracy steadily improves. On the other hand, the model trained with the L1 loss function shows a testing accuracy hovering around 0.1, indicating that the L1 loss function is not suitable for Multiclass Classification Tasks. Among the remaining three loss functions, both focal loss models perform better than the cross-entropy (ce) loss. The final convergence results show that Focal loss (gamma=2) exhibits the best performance.

5.2 F1 Score

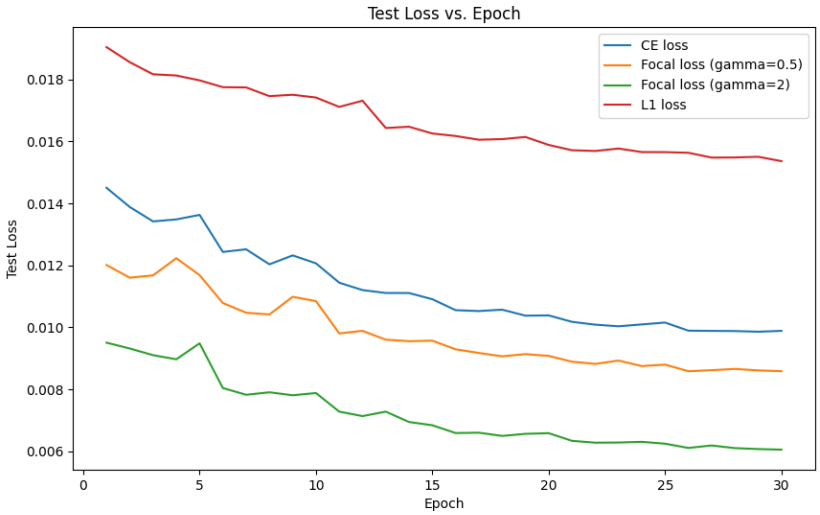
The following visualization showcases the F1 score trends across training epochs for each loss function.



The conclusions drawn from this graph are similar to those of the test loss graph. The L1 loss function is not suitable for Multiclass Classification Tasks. Among the remaining three loss functions, both focal loss models outperform the cross-entropy (ce) loss. The final convergence results indicate that Focal loss (gamma=2) performs the best.

5.3 Test Loss

The final plot illustrates how the test loss changes over training epochs for each loss function.



From this graph, it can be observed that as the number of epochs increases, the test loss of models trained with four different loss functions decreases. In each epoch, the ordering of test loss from smallest to largest among the four loss functions is Focal loss (gamma=2), Focal loss (gamma=0.5), ce loss, and L1 loss. This indicates that the best-performing loss function is Focal loss (gamma=2).

6 Analysis and Discussion:

Upon analyzing the results from the experiments in Section V, the following conclusions have been drawn: L1 loss is not suitable for this multi-class classification task, while Cross-Entropy (CE) loss, Focal loss with gamma=0.5, and Focal loss with gamma=2 exhibit favorable performance. Specifically, Focal loss with gamma=2 performs exceptionally well, followed by Focal loss with gamma=0.5, and CE loss ranks third in this particular task.

6.1 Reasons for the Performance of different loss function

L1 loss typically excels in regression problems, but for multi-class classification tasks, models are better suited to employ classification loss functions. L1 loss may not effectively guide the model to learn complex classification features, hence its suboptimal performance in this task.

Focal loss introduces dynamic sample weights by adjusting the weights of hard-to-classify samples, allowing the model to focus more on learning challenging instances. When gamma=2, Focal loss emphasizes difficult-to-classify samples more strongly, contributing to improved accuracy in classifying complex categories.

Compared to gamma=2, when gamma=0.5, Focal loss treats all samples more evenly without the strong emphasis on hard-to-classify samples seen in gamma=2. This may result in slightly lower learning capabilities for some complex categories.

While Cross-Entropy loss is a common choice for classification tasks, in this specific multi-class classification problem, the complex relationships between categories may lead to relatively poorer performance in certain classes.

6.2 Limitations and Future Improvements:

Single Model Architecture:

- *Limitation:* The experiments were conducted using a single convolutional neural network (CNN) architecture. The choice of architecture may impact the generalizability of the findings.

- *Improvement:* Evaluate multiple architectures to ensure the robustness of the results and identify if certain architectures complement specific loss functions.

Lack of Interpretability Analysis:

- *Limitation:* The study does not delve into the interpretability of the models and how well they generalize to real-world scenarios.
- *Improvement:* Conduct interpretability analysis, such as feature importance or attention mapping, to understand which features contribute most to the model's decision-making.

Single Experiment Run:

- *Limitation:* The experiments were run once, and the results may be subject to random fluctuations.
- *Improvement:* Implement multiple runs with different random seeds to ensure the stability and reproducibility of the results.