Understanding Model Calibration: Expected Calibration Error and Temperature Scaling

Subhasish Bandyopadhyay

November, 2024

Introduction

Deep neural networks have achieved remarkable success in classification tasks, but their confidence scores often don't align with their actual accuracy. A model might be 95% confident in a prediction that's only correct 70% of the time. This misalignment between confidence and accuracy is a calibration problem, and it has serious implications for real-world applications where decision-making depends on reliable uncertainty estimates. In this article, we'll explore Expected Calibration Error (ECE) as a metric for measuring calibration and temperature scaling as a simple yet effective post-processing technique to improve it.

Mathematical Foundation of Model Calibration

Formal Definition

Consider a classification model that outputs a probability distribution over K classes. For an input x, the model produces:

- A predicted class: $\hat{y} = \arg\max_k p(y = k|x)$
- A confidence score: $\hat{p} = \max_k p(y = k|x)$

A model is perfectly calibrated if:

$$P(Y = \hat{y} \mid \hat{P} = p) = p, \quad \forall p \in [0, 1]$$

This means that among all predictions where the model reports confidence p, the actual accuracy should be exactly p. For example, if we collect all predictions where the model is 70% confident, exactly 70% of them should be correct.

The Calibration Gap

The calibration gap at confidence level p is defined as:

$$\text{Gap}(p) = \Big| P(Y = \hat{y} \mid \hat{P} = p) - p \Big|$$

This measures how far off the model's confidence is from its actual accuracy at that confidence level.

Expected Calibration Error (ECE) - Deep Dive

Mathematical Formulation

Since we work with finite samples, we can't evaluate calibration at every possible confidence value. Instead, ECE discretizes the probability space into M bins and computes a weighted average of calibration gaps:

$$\text{ECE} = \sum_{m=1}^{M} \frac{n_m}{n} \times |\text{acc}(B_m) - \text{conf}(B_m)|$$

Where:

- B_m is the m-th bin with boundaries $[(m-1)/M, m/M]$
- $n_m = |\{i : \hat{p}_i \in B_m\}|$ (number of samples in bin m)
- n = total number of samples
- $\text{acc}(B_m) = \frac{1}{n_m} \sum_{i \in B_m} \mathbb{1}(\hat{y}_i = y_i)$ (accuracy in bin m)
- $\text{conf}(B_m) = \frac{1}{n_m} \sum_{i \in B_m} \hat{p}_i$ (average confidence in bin m)
- $\mathbb{1}$ is the indicator function

Detailed ECE Computation Example

```
# Bin 1: [0.0, 0.33] - Low confidence
# Contains 20 samples
# Average confidence: 0.25
# Actual accuracy: 0.30
# Contribution: (20/100) * |0.30 - 0.25| = 0.01
# Bin 2: [0.33, 0.67] - Medium confidence
# Contains 30 samples
# Average confidence: 0.50
# Actual accuracy: 0.45
# Contribution: (30/100) * |0.45 - 0.50| = 0.015
# Bin 3: [0.67, 1.0] - High confidence
# Contains 50 samples
# Average confidence: 0.85
# Actual accuracy: 0.70
# Contribution: (50/100) * |0.70 - 0.85| = 0.075
# ECE = 0.01 + 0.015 + 0.075 = 0.10
```

Implementation Details

```python
def compute_ece(probs, labels, n_bins=10):
    """ Compute Expected Calibration Error
    Args:
        probs: np.array of shape (n_samples, n_classes) - probability distributions
        labels: np.array of shape (n_samples,) - true labels
        n_bins: number of bins for discretization
    Returns:
        ece: scalar Expected Calibration Error
    """
    # Clip probabilities to avoid numerical issues
    probs = np.clip(probs, 1e-5, 1-1e-5)
    # Extract confidence (maximum probability) and predictions
    confidences = np.max(probs, axis=1)  # p_i for each sample
    predictions = np.argmax(probs, axis=1)  # _i for each sample
    accuracies = predictions == labels  # binary array of correct/incorrect
    # Create bins
    bin_boundaries = np.linspace(0, 1, n_bins + 1)
    bin_lowers = bin_boundaries[:-1]
    bin_uppers = bin_boundaries[1:]
    ece = 0.0
    for bin_lower, bin_upper in zip(bin_lowers, bin_uppers):
        # Find samples in this bin
        in_bin = (confidences >= bin_lower) & (confidences < bin_upper)
        prop_in_bin = np.mean(in_bin)  # n_m / n
        if prop_in_bin > 0:
            # Compute accuracy and average confidence in bin
            accuracy_in_bin = np.mean(accuracies[in_bin])  # acc(B_m)
            avg_confidence_in_bin = np.mean(confidences[in_bin])  # conf(B_m)
            # Add weighted absolute difference to ECE
            ece += prop_in_bin * np.abs(avg_confidence_in_bin - accuracy_in_bin)
    return ece
```

Temperature Scaling: The Mathematical Framework

Softmax and Logits

Neural networks for K-class classification typically output logits $z \in \mathbb{R}^K$, which are converted to probabilities via softmax:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$$

The maximum probability (confidence) is:

$$\hat{p} = \max_i p_i = \max_i \left[\frac{\exp(z_i)}{\sum_j \exp(z_j)} \right]$$

Temperature-Modified Softmax

Temperature scaling introduces a scalar parameter $T > 0$:

$$p_i^{(T)} = \frac{\exp(z_i/T)}{\sum_{j=1}^{K} \exp(z_j/T)}$$

This can be understood through several mathematical perspectives:

1. **Entropy Perspective** The entropy of the distribution changes with temperature:

$$H(p^{(T)}) = -\sum_i p_i^{(T)} \log(p_i^{(T)})$$

- As $T \to \infty$: $H(p^{(T)}) \to \log(K)$ (maximum entropy, uniform distribution)
- As $T \to 0$: $H(p^{(T)}) \to 0$ (minimum entropy, one-hot distribution)
- $T = 1$: Original entropy

2. **Gradient Perspective** The gradient of log-probabilities with respect to temperature:

$$\frac{\partial \log(p_i^{(T)})}{\partial T} = \frac{z_i - \sum_j p_j^{(T)} z_j}{T^2}$$

This shows how confidence changes: if z_i is larger than the weighted average, decreasing T increases p_i.

3. **Preservation of Accuracy** Crucially, temperature scaling preserves the argmax:

$$\arg\max_i p_i^{(T)} = \arg\max_i \exp(z_i/T) = \arg\max_i z_i$$

This is because $\exp(\cdot)$ is monotonic and $T > 0$. Therefore:

- The predicted class remains unchanged
- Top-1 accuracy is preserved
- Only confidence scores are adjusted

Optimal Temperature Selection

The optimal temperature T^* minimizes ECE on a validation set:

$$T^* = \arg\min_T \text{ECE}(p^{(T)}, y_{\text{val}})$$

This optimization problem has nice properties:

- **Convexity**: ECE is often approximately convex in T near the optimum
- **Smoothness**: The objective is differentiable with respect to T
- **Bounded**: Practical T values typically lie in $[0.5, 5.0]$

Mathematical Analysis of Temperature Effects

Consider the logit difference between the top two classes:

$$\Delta z = z_{\max} - z_{\text{second}}$$

The confidence after temperature scaling is approximately:

$$\hat{p}^{(T)} \approx \sigma(\Delta z/T) = \frac{1}{1 + \exp(-\Delta z/T)}$$

Where σ is the sigmoid function. This reveals:

- Large T: $\hat{p}^{(T)} \to 0.5 + \frac{\Delta z}{4T}$ (linear approximation near 0.5)
- Small T: $\hat{p}^{(T)} \to 1$ or 0 (saturation)

The rate of confidence change:

$$\left|\frac{d\hat{p}}{dT}\right| \propto \Delta z \times \hat{p}(1-\hat{p})/T^2$$

Advanced Calibration Metrics

Maximum Calibration Error (MCE)

While ECE measures average calibration error, MCE captures the worst-case:

$$\text{MCE} = \max_m |\text{acc}(B_m) - \text{conf}(B_m)|$$

Adaptive ECE (ACE)

Instead of equal-width bins, ACE uses equal-frequency bins:

- Each bin contains exactly n/M samples

This ensures all bins contribute equally to the metric, avoiding issues with sparse bins.

Classwise ECE

For multi-class problems, we can compute ECE per class:

$$\text{ECE}_k = \sum_m \frac{n_m^k}{n^k} \times |\text{acc}_k(B_m) - \text{conf}_k(B_m)|$$

Where calculations are restricted to samples where class k was predicted.

Theoretical Insights

Why Neural Networks Are Miscalibrated

Recent research suggests several factors:

- **Capacity and Overfitting**: Modern networks have capacity to memorize training data, leading to overconfident predictions
- **Cross-Entropy Loss**: The negative log-likelihood loss drives logits to infinity for correctly classified examples:
$$L = -\log(p_y) \to 0 \quad \text{implies} \quad z_y \to \infty$$
- **Batch Normalization**: Can amplify confidence by reducing internal covariate shift
- **Model Size**: Deeper and wider networks tend to be more miscalibrated:
$$\text{ECE} \propto f(\text{depth}, \text{width}, \text{parameters})$$

Statistical Properties

Consistency Temperature scaling is consistent: as $n \to \infty$, the estimated T^* converges to the true optimal temperature.

Sample Complexity The number of validation samples needed for reliable temperature estimation:

$$n_{\text{val}} \geq O\left(\frac{K \log(K)}{\varepsilon^2}\right)$$

Where K is the number of classes and ε is the desired ECE accuracy.

Implementation: Complete Framework

```python
import numpy as np
import torch
import torch.nn.functional as F
from scipy.optimize import minimize_scalar

class TemperatureScaling:
    def __init__(self, n_bins=15):
        self.n_bins = n_bins
        self.temperature = 1.0

    def compute_ece(self, probs, labels, n_bins=None):
        """ Enhanced ECE computation with additional metrics """
        if n_bins is None:
            n_bins = self.n_bins
        probs = np.clip(probs, 1e-5, 1-1e-5)
        confidences = np.max(probs, axis=1)
        predictions = np.argmax(probs, axis=1)
        accuracies = predictions == labels
        # Standard ECE
        bin_boundaries = np.linspace(0, 1, n_bins + 1)
        bin_lowers = bin_boundaries[:-1]
        bin_uppers = bin_boundaries[1:]
        ece = 0.0
        mce = 0.0  # Maximum Calibration Error
        bin_stats = []
        for bin_lower, bin_upper in zip(bin_lowers, bin_uppers):
            in_bin = (confidences >= bin_lower) & (confidences < bin_upper)
            prop_in_bin = np.mean(in_bin)
            if prop_in_bin > 0:
                accuracy_in_bin = np.mean(accuracies[in_bin])
                avg_confidence_in_bin = np.mean(confidences[in_bin])
                calibration_gap = np.abs(avg_confidence_in_bin -
                    accuracy_in_bin)
                ece += prop_in_bin * calibration_gap
                mce = max(mce, calibration_gap)
                bin_stats.append({
                    'range': (bin_lower, bin_upper),
                    'count': np.sum(in_bin),
                    'accuracy': accuracy_in_bin,
                    'confidence': avg_confidence_in_bin,
                    'gap': calibration_gap
                })
        return {
            'ece': ece,
            'mce': mce,
            'bin_stats': bin_stats
        }

    def find_optimal_temperature(self, val_logits, val_labels):
        """ Find optimal temperature using Brent's method """
        val_logits_tensor = torch.tensor(val_logits)
        def objective(temp):
            # Compute ECE for given temperature
            scaled_probs = F.softmax(val_logits_tensor / temp, dim=1).numpy()
            return self.compute_ece(scaled_probs, val_labels)['ece']
        # Optimize using scipy's Brent method
        result = minimize_scalar(objective, bounds=(0.1, 10.0), method='bounded
            ')
        self.temperature = result.x
        return self.temperature

    def calibrate(self, logits):
        """ Apply temperature scaling to logits """
        return F.softmax(torch.tensor(logits) / self.temperature, dim=1).numpy
            ()
```

Experimental Validation

Typical Results

On standard datasets, temperature scaling achieves:

Model	Dataset	ECE (Before)	ECE (After)	Optimal T
ResNet-110	CIFAR-10	0.142	0.031	2.31
DenseNet-121	CIFAR-10	0.187	0.042	2.84
ResNet-50	ImageNet	0.121	0.019	1.73

Reliability Diagrams

The improvement can be visualized through reliability diagrams, where perfect calibration appears as a diagonal line:

- Perfect calibration: accuracy = confidence
- Before scaling: typically curved above diagonal (overconfident)
- After scaling: closely follows diagonal

Limitations and Extensions

Limitations of Temperature Scaling

- **Single Parameter**: Cannot correct class-specific miscalibration
- **Global Adjustment**: Applies uniform scaling across all confidence levels
- **Validation Dependency**: Requires held-out data for temperature selection

Advanced Calibration Methods

- **Vector Scaling** Uses a different temperature per class:

$$p_i^{(T)} = \frac{\exp(z_i/T_i)}{\sum_j \exp(z_j/T_j)}$$

- **Matrix Scaling** Applies a linear transformation:

$$z' = Wz + b$$

$$p = \text{softmax}(z')$$

- **Platt Scaling** Fits a sigmoid to logits:

$$p' = \sigma(az + b)$$

Conclusion

Expected Calibration Error provides a rigorous framework for measuring model calibration, while temperature scaling offers an elegant solution with strong theoretical foundations. The mathematical simplicity of temperature scalingmodifying logits by a single scalarbelies its effectiveness in practice. The key insights are:

- ECE quantifies the expected difference between confidence and accuracy
- Temperature scaling preserves accuracy while adjusting confidence
- The optimal temperature can be found efficiently through optimization

- The method is theoretically grounded and empirically validated

For practitioners, implementing temperature scaling should be standard practice when deploying classification models, especially in risk-sensitive applications. The minimal computational overhead and significant calibration improvements make it an essential component of reliable machine learning systems.