MoE Odyssey: 6. Optimal Allocation for Equilibrium

Su Jianlin

2026-02-22

Introduction

We already know that load balancing is a fundamental and critical component in the MoE (Mixture of Experts) architecture, directly affecting model efficiency and performance. This series has previously introduced two mainstream approaches to achieving load balancing: the classic Aux Loss method described in *MoE Odyssey: 2. Not the Amount, But the Imbalance Matters*, and the Loss-Free method proposed by DeepSeek in *MoE Odyssey: 3. A Different Approach to Allocation*. Each has its strengths and limitations.

This article explores a third approach: **Optimal Allocation**, which treats load balancing as a linear programming problem under equality constraints. In its final form, it still belongs to the Loss-Free category but is based on a radically different principle, providing a more accurate and hyperparameter-free update method.

1 Review of Methods

The Aux Loss approach is relatively straightforward: its core idea is "penalize where imbalance occurs" by imposing a regularization term that penalizes uneven loads. However, Aux Loss has two main issues: first, the penalty coefficient is difficult to tunetoo large and it interferes with the main loss optimization, too small and the balancing effect is poor; second, Aux Loss relies on the STE (Straight-Through Estimator), which means its gradient is suboptimal and may introduce unintended side effects beyond load balancing.

To address this, DeepSeek proposed the Loss-Free method, which introduces an additional bias term to assist in sorting:

$$\boldsymbol{y} = \sum_{i\in \mathop{\text{argtop}}_k \boldsymbol{\rho}} \rho_i \boldsymbol{e}_i \qquad \to \qquad \boldsymbol{y} = \sum_{i\in \mathop{\text{argtop}}_k \boldsymbol{\rho}+\boldsymbol{b}} \rho_i \boldsymbol{e}_i \tag{1}$$

Note that $\boldsymbol{b}$ is only used to adjust the expert ranking; the actual weights applied are still ρ_i, so it does not directly participate in model computation or interfere with gradient flow. However, since $\boldsymbol{b}$ has no gradient, we must define its update rule manually. The idea is intuitive: the larger b_i, the higher the probability that the i-th expert is selected. Thus, we first collect the current load distribution $\boldsymbol{F}$, and if F_i exceeds the expected value $1/n$, we reduce b_i, otherwise we increase it:

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \gamma \,\text{sign}(\boldsymbol{F} - 1/n) \tag{2}$$

Overall, Loss-Free is less intrusive and arguably more elegant than Aux Loss, but it is not without flaws. While it avoids the penalty coefficient, it still requires tuning γ, which acts as the learning rate for $\boldsymbol{b}$. The paper recommends $\gamma = 10^{-3}$, which is tightly coupled with the Sigmoid activation of $\boldsymbol{\rho}$. Changing the activation function would require re-tuning γ.

Furthermore, even with Sigmoid, if the $\boldsymbol{\rho}$ distribution in certain layers is "distorted", the model becomes sensitive to γ, making load balancing difficult with a fixed γ. This is not uncommonfor instance, early layers in an MoE model often struggle to balance, leading to the "first_k_dense" operation. Similarly, when the model is large or the number of experts n is large, certain layers may be hard to balance.

2 Linear Programming

Let's formalize the problem: suppose we have m tokens, and the router assigns a score $\boldsymbol{s}_i = (s_{i,1}, s_{i,2}, \cdots, s_{i,n})$ to each of the n experts for the i-th token. There are mn scores in total, which may be positive or negative and not necessarily confined to a predefined range. We want to devise an allocation scheme based on these scores that determines which expert each token should activate.

We impose two constraints:

- Each token selects only k experts.
- Each expert is activated exactly mk/n times.

Under these constraints, we seek the allocation that maximizes the total score:

$$\max_{x_{i,j}\in\{0,1\}} \sum_{i,j} x_{i,j} s_{i,j} \qquad \text{s.t.} \qquad \sum_j x_{i,j} = k, \quad \sum_i x_{i,j} = \frac{mk}{n} \tag{3}$$

where $x_{i,j} = 1$ means the i-th token selects the j-th expert, and $x_{i,j} = 0$ otherwise. We assume mk/n is an integer to ensure strict equality in constraints.

This is an integer programming problem, which is generally difficult to solve. We consider its relaxed version:

$$\max_{x_{i,j}\in[0,1]} \sum_{i,j} x_{i,j} s_{i,j} \qquad \text{s.t.} \qquad \sum_j x_{i,j} = k, \quad \sum_i x_{i,j} = \frac{mk}{n} \tag{4}$$

Now $x_{i,j}$ can take any value in $[0, 1]$, but the constraints remain unchanged. Since both the objective and constraints are linear in $x_{i,j}$, this becomes a linear programming problem over a bounded region.

3 Max-Min Formulation

To handle the constraints, we use the Lagrangian multiplier method:

$$\max_{x_{i,j}\in[0,1]} \min_{\alpha_i,\beta_j} \sum_{i,j} x_{i,j} s_{i,j} - \sum_i \alpha_i \left(\sum_j x_{i,j} - k\right) - \sum_j \beta_j \left(\sum_i x_{i,j} - \frac{mk}{n}\right) \tag{5}$$

If $\sum_j x_{i,j} = k$ and $\sum_i x_{i,j} = mk/n$, this is equivalent to (4). Otherwise, the min step yields $-\infty$, which is less than the finite maximum. Thus, only the feasible region is valid.

Since the objective is linear in $x_{i,j}, \alpha_i, \beta_j$, and $[0, 1]$ is a convex set, we can swap max and min:

$$\min_{\alpha_i,\beta_j} \max_{x_{i,j}\in[0,1]} \sum_{i,j} x_{i,j}(s_{i,j} - \alpha_i - \beta_j) + k\sum_i \alpha_i + \frac{mk}{n}\sum_j \beta_j \tag{6}$$

The max step can be solved directly:

$$\begin{cases} x^*_{i,j} = 1, & s_{i,j} - \alpha_i - \beta_j > 0 \\ x^*_{i,j} = 0, & s_{i,j} - \alpha_i - \beta_j < 0 \\ x^*_{i,j} \in [0,1], & s_{i,j} - \alpha_i - \beta_j = 0 \end{cases} \tag{7}$$

The special case $s_{i,j} - \alpha_i - \beta_j = 0$ is rare and can be ignored, allowing $x^*_{i,j}$ to be either 0 or 1. Thus, the relaxed problem (4) and the original integer problem (3) have the same optimal solution.

4 Divide and Conquer

Substituting $x^*_{i,j}$ into (6), we get:

$$\min_{\alpha_i,\beta_j} \sum_{i,j} \max(0, s_{i,j} - \alpha_i - \beta_j) + k\sum_i \alpha_i + \frac{mk}{n}\sum_j \beta_j \tag{8}$$

We solve this using alternating minimization: fix β_j, solve for α_i; fix α_i, solve for β_j, and repeat. Since α_i, β_j are symmetric, the two steps are essentially the same. Fixing β_j, we minimize:

$$\min_{\alpha_i} \sum_{i,j} \max(0, s_{i,j} - \alpha_i - \beta_j) + k\sum_i \alpha_i \tag{9}$$

Each α_i is independent, so we can reduce it to:

$$\min_{\alpha} k\alpha + \sum_j \max(0, s_j - \beta_j - \alpha) \tag{10}$$

Sorting $s_j - \beta_j$ descending, we find that the minimum occurs at the $(k+1)$-th largest value, so we set α^* to this value.

5 Alternating Iteration

Restoring the index i, we find that α^*_i is the $(k+1)$-th largest value of $s_{i,j} - \beta_j$. Similarly, fixing α_i, β^*_j is the $(mk/n+1)$-th largest value of $s_{i,j} - \alpha_i$.

After sufficient iterations, $x^*_{i,j}$ satisfies the constraints and is binary. For each token i, the selected experts are the top-k of $\boldsymbol{s}_i - \boldsymbol{\beta}^*$. Thus, only $\boldsymbol{\beta}^*$ is needed during inference, while $\boldsymbol{\alpha}^*$ is an intermediate variable.

This leads to the Quantile Balancing (QB) algorithm:

```
Quantile Balancing (QB): Alternating Algorithm for Problem (3)
Input: Score matrix $\boldsymbol{s} \in \mathbb{R}^{m\times n}$
Output: Allocation matrix $\boldsymbol{x} \in \{0,1\}^{m\times n}$
1: Initialize $\boldsymbol{\beta} = \mathbf{0}_{1\times n}$
2: For $t = 1, 2, \cdots, T$ do
3:     $\boldsymbol{\alpha} \leftarrow \text{des\_sort}(\boldsymbol{s} - \boldsymbol{\beta}, \text{axis=1})_{[:,k:k+1]}$
4:     $\boldsymbol{\beta} \leftarrow \text{des\_sort}(\boldsymbol{s} - \boldsymbol{\alpha}, \text{axis=0})_{[mk/n:mk/n+1]}$
5: Output $x_{i,j} = 1$ if $j \in \text{argtop}_k\, \boldsymbol{s}_i - \boldsymbol{\beta}$, else 0
```

Using the concept of quantiles, we unify the $(k+1)$-th and $(mk/n+1)$-th largest values as the $(1-k/n)$-th quantile in their respective dimensions. This avoids full sorting and saves computational cost.

6 Potential Pitfall

There is a subtle pitfall: during training, we must ensure that the bias $\boldsymbol{\beta}$ is not updated before selecting experts. That is, we must use the old $\boldsymbol{\beta}$ to select experts, then update $\boldsymbol{\beta}$, to avoid information leakage.

While the risk may seem negligible, especially for small models, it becomes significant for large models with strong capabilities. To mitigate this, we can update $\boldsymbol{\beta}$ iteratively from the previous step rather than initializing from zero, reducing overfitting and computation.

7 Image Illustration

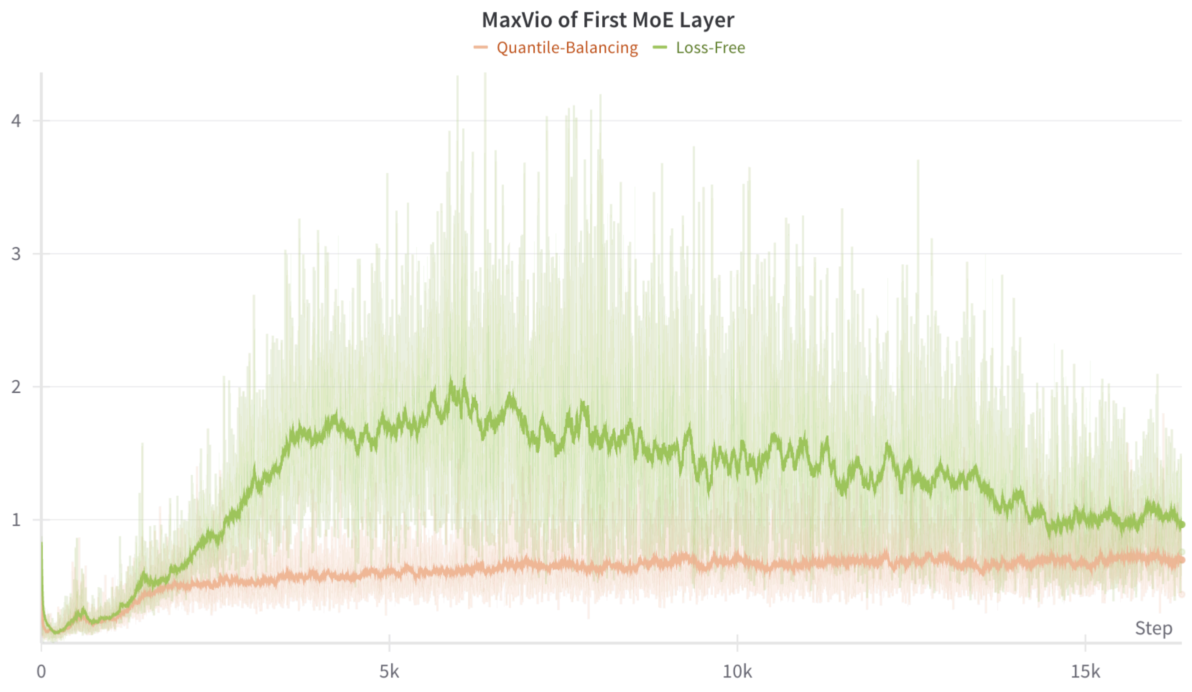

Figure 1: Comparison of MaxVio for the First MoE Layer

8 Demonstration Code

Here is a Python code snippet to demonstrate the Quantile Balancing method:

```python
import numpy as np

def quantile_bias(s, k, T=5):
    """Alternating quantile for optimal bias"""
    m, n = s.shape
    beta = np.zeros((1, n))
    for _ in range(T):
        alpha = np.quantile(s - beta, 1 - k / n, axis=1, keepdims=True)
        beta = np.quantile(s - alpha, 1 - k / n, axis=0, keepdims=True)
    return beta

def max_min_avg_vio(s, k):
    """Compute max, min, and avg violation"""
    m, n = s.shape
    topk = np.argsort(-s, axis=1)[:, :k]
    f = np.bincount(topk.reshape(-1), minlength=n)
    f = f / f.sum() * n - 1
    return f.max(), f.min(), np.abs(f).mean()

m, n, k = 100000, 256, 8
s = np.random.rand(m, n) + np.random.rand(n)
b = quantile_bias(s, k, 5)
max_min_avg_vio(s, k)
max_min_avg_vio(s - b, k)
```

9 Related Work

The idea of viewing MoE load balancing as an optimal allocation problem was first introduced in *BASE Layers: Simplifying Training of Large, Sparse Models*, but a general solution was proposed in *Binary-Integer-Programming Based Algorithm for Expert Load Balancing in Mixture-of-Experts Models* (BIP), which QB improves upon.

BIP relaxes the equality constraints to inequalities:

$$\max_{x_{i,j}\in\{0,1\}} \sum_{i,j} x_{i,j} s_{i,j} \qquad \text{s.t.} \qquad \sum_j x_{i,j} \le k, \quad \sum_i x_{i,j} \le \frac{mk}{n} \tag{11}$$

It adds a non-negativity constraint on α_i, β_j, leading to:

$$\begin{aligned} \boldsymbol{\alpha} &\leftarrow \max(0, \text{des\_sort}(\boldsymbol{s}-\boldsymbol{\beta}, \text{axis=1})_{[:,k:k+1]}) \\ \boldsymbol{\beta} &\leftarrow \max(0, \text{des\_sort}(\boldsymbol{s}-\boldsymbol{\alpha}, \text{axis=0})_{[mk/n:mk/n+1]}) \end{aligned} \tag{12}$$

However, this clipping operation often degrades performance, preventing "assistance to the under-represented." QB removes this constraint, leading to better convergence and balance.

10 Gradient Descent

Finally, we explore a hybrid approach between Loss-Free and QB. Given $\boldsymbol{\alpha}$, the optimization of $\boldsymbol{\beta}$ becomes:

$$\min_{\beta_j} \sum_{i,j} \max(0, s_{i,j} - \alpha_i - \beta_j) + \frac{mk}{n} \sum_j \beta_j \tag{13}$$

This objective is differentiable, allowing gradient descent:

$$\frac{\partial \ell}{\partial \beta_j} = \frac{mk}{n} - \sum_{i=1}^{m} \chi(s_{i,j} - \alpha_i - \beta_j > 0) \tag{14}$$

Using SignSGD:

$$\beta_j \leftarrow \beta_j - \gamma \,\text{sign}\left(\frac{\partial \ell}{\partial \beta_j}\right) \tag{15}$$

This matches the cost of Loss-Free and performs between it and QB in practice.

11 Summary

This article explored the load balancing problem in MoE from an optimal allocation perspective, leading to a new auxiliary-loss-free algorithm called Quantile Balancing (QB). It is more stable and accurate than existing methods, works for any score range, and requires no hyperparameter tuning.