第12章 模型优化与调试

> 优化是一门艺术，调试是一门科学

12.1 训练问题诊断

12.1.1 Loss曲线分析

```python
import matplotlib.pyplot as plt

def analyze_training_loss(loss_history):
    """
    分析训练loss曲线
    """
    plt.figure(figsize=(12, 4))

    # 绘制Loss曲线
    plt.subplot(1, 2, 1)
    plt.plot(loss_history['train_loss'], label='Train Loss')
    plt.plot(loss_history['val_loss'], label='Val Loss')
    plt.xlabel('Step')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Loss Curve')

    # 平滑后的曲线
    plt.subplot(1, 2, 2)
    smooth_train = moving_average(loss_history['train_loss'], window=100)
    smooth_val = moving_average(loss_history['val_loss'], window=100)
    plt.plot(smooth_train, label='Train Loss (smoothed)')
    plt.plot(smooth_val, label='Val Loss (smoothed)')
    plt.xlabel('Step')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Smoothed Loss Curve')

    plt.tight_layout()
    plt.savefig('loss_analysis.png')

    # 诊断
    diagnosis = diagnose_loss_curve(loss_history)
    return diagnosis


def diagnose_loss_curve(loss_history):
    """
```

```python
    诊断loss曲线问题
    """
    train_loss = loss_history['train_loss']
    val_loss = loss_history['val_loss']

    issues = []

    # 1. 检查是否不下降
    if train_loss[-1] > train_loss[0] * 0.9:
        issues.append({
            "problem": "训练loss不下降",
            "possible_causes": [
                "学习率过小",
                "学习率过大（爆炸）",
                "数据问题",
                "模型初始化问题"
            ],
            "solutions": [
                "调整学习率（尝试1e-5到1e-3）",
                "检查梯度范数",
                "验证数据质量",
                "使用更好的初始化"
            ]
        })

    # 2. 检查过拟合
    gap = val_loss[-1] - train_loss[-1]
    if gap > train_loss[-1] * 0.3:
        issues.append({
            "problem": "过拟合",
            "possible_causes": [
                "模型太大",
                "训练数据太少",
                "训练时间太长"
            ],
            "solutions": [
                "增加正则化（dropout, weight decay）",
                "数据增强",
                "Early stopping",
                "使用更小的模型"
            ]
        })

    # 3. 检查欠拟合
    if train_loss[-1] > expected_final_loss * 1.5:
        issues.append({
            "problem": "欠拟合",
            "possible_causes": [
                "模型太小",
```

```python
                "训练不充分",
                "学习率太小"
            ],
            "solutions": [
                "增大模型",
                "训练更多epochs",
                "提高学习率",
                "检查数据质量"
            ]
        })

    # 4. 检查震荡
    volatility = np.std(train_loss[-100:]) / np.mean(train_loss[-100:])
    if volatility > 0.1:
        issues.append({
            "problem": "训练不稳定/震荡",
            "possible_causes": [
                "学习率过大",
                "batch size太小",
                "梯度爆炸"
            ],
            "solutions": [
                "降低学习率",
                "增大batch size",
                "梯度裁剪",
                "使用warmup"
            ]
        })

    if not issues:
        return {"status": "健康", "message": "训练曲线看起来正常"}

    return {"status": "有问题", "issues": issues}
```

12.1.2 梯度监控

```python
class GradientMonitor:
    """
    梯度监控器
    """
    def __init__(self, model):
        self.model = model
        self.gradient_norms = []
        self.param_norms = []

    def log_gradients(self):
        """
        记录梯度信息
```

```python
        """
        total_norm = 0
        param_norm = 0

        for name, param in self.model.named_parameters():
            if param.grad is not None:
                # 参数梯度的范数
                grad_norm = param.grad.data.norm(2).item()
                total_norm += grad_norm ** 2

                # 参数本身的范数
                param_norm += param.data.norm(2).item() ** 2

        total_norm = total_norm ** 0.5
        param_norm = param_norm ** 0.5

        self.gradient_norms.append(total_norm)
        self.param_norms.append(param_norm)

        return total_norm, param_norm

    def check_gradient_health(self):
        """
        检查梯度健康状况
        """
        if not self.gradient_norms:
            return "No gradient data"

        recent_grads = self.gradient_norms[-100:]
        avg_grad = np.mean(recent_grads)

        issues = []

        # 1. 梯度消失
        if avg_grad < 1e-7:
            issues.append({
                "problem": "梯度消失",
                "solutions": [
                    "检查激活函数（避免sigmoid）",
                    "使用残差连接",
                    "使用Layer Normalization",
                    "降低网络深度"
                ]
            })

        # 2. 梯度爆炸
        if avg_grad > 100:
            issues.append({
                "problem": "梯度爆炸",
```

```python
            "solutions": [
                "梯度裁剪（clip_grad_norm）",
                "降低学习率",
                "检查数据是否异常",
                "使用Layer Normalization"
            ]
        })

    # 3. 梯度NaN
    if np.isnan(recent_grads).any():
        issues.append({
            "problem": "梯度包含NaN",
            "solutions": [
                "检查数据是否有NaN/Inf",
                "降低学习率",
                "使用混合精度时注意loss scaling",
                "检查除零错误"
            ]
        })

    return issues if issues else "健康"

def plot_gradient_flow(self):
    """
    可视化梯度流
    """
    plt.figure(figsize=(12, 5))

    # 绘制各层梯度
    layer_grads = {}
    for name, param in self.model.named_parameters():
        if param.grad is not None:
            layer_name = name.split('.')[0]
            grad_norm = param.grad.data.norm(2).item()

            if layer_name not in layer_grads:
                layer_grads[layer_name] = []
            layer_grads[layer_name].append(grad_norm)

    # 绘制
    layers = list(layer_grads.keys())
    avg_grads = [np.mean(layer_grads[l]) for l in layers]

    plt.bar(range(len(layers)), avg_grads)
    plt.xticks(range(len(layers)), layers, rotation=45)
    plt.ylabel('Average Gradient Norm')
    plt.title('Gradient Flow Across Layers')
```

```python
    plt.tight_layout()
    plt.savefig('gradient_flow.png')
```

12.1.3 学习率调优

```python
def learning_rate_finder(model, train_loader, optimizer, criterion):
    """
    学习率范围测试（LR Finder）

    在不同学习率下训练，找到最佳学习率范围
    """
    model.train()

    # 学习率范围: 1e-7 到 1
    lr_min = 1e-7
    lr_max = 1
    num_steps = 100

    lrs = np.logspace(np.log10(lr_min), np.log10(lr_max), num_steps)
    losses = []

    for i, lr in enumerate(lrs):
        # 设置学习率
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr

        # 训练一步
        batch = next(iter(train_loader))
        optimizer.zero_grad()

        outputs = model(batch['input_ids'])
        loss = criterion(outputs, batch['labels'])

        loss.backward()
        optimizer.step()

        losses.append(loss.item())

        # 如果loss爆炸，提前停止
        if loss.item() > losses[0] * 10:
            break

    # 绘制结果
    plt.figure(figsize=(10, 5))
    plt.semilogx(lrs[:len(losses)], losses)
    plt.xlabel('Learning Rate')
    plt.ylabel('Loss')
    plt.title('Learning Rate Finder')
```

```python
    plt.grid(True)
    plt.savefig('lr_finder.png')

    # 找到最佳学习率（Loss下降最快的点）
    gradients = np.gradient(losses)
    best_lr_idx = np.argmin(gradients)
    best_lr = lrs[best_lr_idx]

    print(f"建议学习率: {best_lr:.2e}")
    print(f"建议范围: {best_lr/10:.2e} 到 {best_lr*2:.2e}")

    return best_lr
```

12.2 数据问题排查

12.2.1 数据质量检查

```python
class DataQualityChecker:
    """
    数据质量检查器
    """
    def __init__(self, dataset, tokenizer):
        self.dataset = dataset
        self.tokenizer = tokenizer

    def run_checks(self):
        """
        运行所有检查
        """
        print("检查数据质量...")

        results = {}

        results['length_stats'] = self.check_length_distribution()
        results['duplicates'] = self.check_duplicates()
        results['quality'] = self.check_text_quality()
        results['balance'] = self.check_class_balance()

        self.generate_report(results)

        return results

    def check_length_distribution(self):
        """
        检查长度分布
        """
        lengths = []
```

```python
        for example in self.dataset:
            text = example['text']
            tokens = self.tokenizer.encode(text)
            lengths.append(len(tokens))

        stats = {
            "mean": np.mean(lengths),
            "std": np.std(lengths),
            "min": np.min(lengths),
            "max": np.max(lengths),
            "median": np.median(lengths),
            "p95": np.percentile(lengths, 95)
        }

        # 绘制分布
        plt.figure(figsize=(10, 5))
        plt.hist(lengths, bins=50)
        plt.xlabel('Length (tokens)')
        plt.ylabel('Count')
        plt.title('Length Distribution')
        plt.axvline(stats['mean'], color='r', linestyle='--', label=f"Mean: {stats['mear
        plt.legend()
        plt.savefig('length_distribution.png')

        # 检查异常
        warnings = []
        if stats['std'] > stats['mean']:
            warnings.append("长度方差很大，考虑分桶或裁剪")

        if stats['max'] > 2048:
            warnings.append(f"存在超长样本（{stats['max']} tokens），可能需要截断")

        return {"stats": stats, "warnings": warnings}

    def check_duplicates(self):
        """
        检查重复数据
        """
        from collections import Counter

        texts = [example['text'] for example in self.dataset]

        # 完全重复
        exact_duplicates = len(texts) - len(set(texts))

        # 近似重复（使用MinHash）
        # ... 实现略
```

```python
        duplicate_rate = exact_duplicates / len(texts)

        if duplicate_rate > 0.01:
            warning = f"检测到{duplicate_rate:.1%}的重复数据，建议去重"
        else:
            warning = None

        return {
            "exact_duplicates": exact_duplicates,
            "duplicate_rate": duplicate_rate,
            "warning": warning
        }

    def check_text_quality(self):
        """
        检查文本质量
        """
        issues = []

        for i, example in enumerate(self.dataset[:1000]):  # 抽样检查
            text = example['text']

            # 检查1: 过短
            if len(text) < 10:
                issues.append(f"样本{i}: 文本过短")

            # 检查2: 特殊字符比例过高
            special_char_ratio = sum(not c.isalnum() and not c.isspace()
                                     for c in text) / len(text)
            if special_char_ratio > 0.3:
                issues.append(f"样本{i}: 特殊字符过多")

            # 检查3: 重复字符
            if any(text.count(c*10) > 0 for c in 'abcdefghijklmnopqrstuvwxyz'):
                issues.append(f"样本{i}: 检测到重复字符")

        return {
            "issues_found": len(issues),
            "sample_issues": issues[:10]  # 只显示前10个
        }

    def check_class_balance(self):
        """
        检查类别平衡（如果是分类任务）
        """
        if 'label' not in self.dataset[0]:
            return "N/A (not a classification task)"

        from collections import Counter
```

```python
        labels = [example['label'] for example in self.dataset]
        label_counts = Counter(labels)

        # 计算不平衡程度
        max_count = max(label_counts.values())
        min_count = min(label_counts.values())
        imbalance_ratio = max_count / min_count

        warning = None
        if imbalance_ratio > 10:
            warning = f"类别严重不平衡（{imbalance_ratio:.1f}:1），考虑重采样或加权"

        return {
            "label_distribution": dict(label_counts),
            "imbalance_ratio": imbalance_ratio,
            "warning": warning
        }
```

12.2.2 数据增强

```python
def augment_data(text):
    """
    数据增强
    """
    augmented = []

    # 1. 回译（Back-translation）
    # translated = translate(text, 'en', 'fr')
    # back_translated = translate(translated, 'fr', 'en')
    # augmented.append(back_translated)

    # 2. 同义词替换
    # augmented.append(synonym_replacement(text))

    # 3. 随机插入
    # augmented.append(random_insertion(text))

    # 4. 随机删除
    # augmented.append(random_deletion(text, p=0.1))

    # 5. 随机交换
    # augmented.append(random_swap(text, n=3))

    return augmented
```

12.3 性能优化

12.3.1 训练加速技巧

```python
# 技巧1: 梯度累积
def train_with_gradient_accumulation(model, dataloader, accumulation_steps=4):
    """
    梯度累积：模拟大batch size
    """
    optimizer.zero_grad()

    for i, batch in enumerate(dataloader):
        # 前向+反向
        loss = model(batch)
        loss = loss / accumulation_steps  # 归一化
        loss.backward()

        # 每accumulation_steps步更新一次
        if (i + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()


# 技巧2: 混合精度训练
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()

for batch in dataloader:
    optimizer.zero_grad()

    # 使用autocast
    with autocast():
        output = model(batch)
        loss = criterion(output, labels)

    # scaled backward
    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()


# 技巧3: DataLoader优化
train_loader = DataLoader(
    dataset,
    batch_size=32,
    num_workers=4,  # 多进程加载
    pin_memory=True,  # 锁页内存
    prefetch_factor=2,  # 预取
)
```

```python
# 技巧4: 编译模型（PyTorch 2.0+）
model = torch.compile(model)
```

12.3.2 显存优化

```python
# 技巧1: 梯度检查点（Gradient Checkpointing）
from torch.utils.checkpoint import checkpoint

class TransformerBlockWithCheckpoint(nn.Module):
    def forward(self, x):
        # 使用checkpoint减少显存
        return checkpoint(self._forward, x)

    def _forward(self, x):
        x = self.attention(x)
        x = self.ffn(x)
        return x


# 技巧2: 清理缓存
import gc
import torch

def clear_memory():
    """
    清理显存
    """
    gc.collect()
    torch.cuda.empty_cache()


# 技巧3: 使用更小的数据类型
model = model.half()  # FP16
# 或
model = model.bfloat16()  # BF16


# 技巧4: 零冗余优化器（ZeRO）
from deepspeed import DeepSpeedEngine

model_engine = DeepSpeedEngine(
    model=model,
    config={
        "zero_optimization": {
            "stage": 3,  # ZeRO Stage 3
        }
```

```
    }
)
```

12.4 超参数调优

12.4.1 网格搜索 vs 随机搜索

```python
from sklearn.model_selection import ParameterGrid
import random

# 网格搜索
param_grid = {
    'learning_rate': [1e-5, 5e-5, 1e-4],
    'batch_size': [16, 32, 64],
    'warmup_steps': [100, 500, 1000]
}

grid = ParameterGrid(param_grid)

best_score = 0
best_params = None

for params in grid:
    score = train_and_evaluate(model, params)
    if score > best_score:
        best_score = score
        best_params = params

print(f"Best params: {best_params}")
print(f"Best score: {best_score}")


# 随机搜索（更高效）
def random_search(param_distributions, n_iter=10):
    """
    随机搜索超参数
    """
    results = []

    for i in range(n_iter):
        # 随机采样超参数
        params = {
            'learning_rate': random.choice([1e-5, 5e-5, 1e-4, 5e-4]),
            'batch_size': random.choice([16, 32, 64]),
            'warmup_steps': random.randint(100, 1000),
            'weight_decay': random.uniform(0.01, 0.1)
        }
```

```python
        # 训练评估
        score = train_and_evaluate(model, params)

        results.append({
            'params': params,
            'score': score
        })

    # 排序
    results.sort(key=lambda x: x['score'], reverse=True)

    return results[0]  # 返回最佳结果
```

12.4.2 贝叶斯优化

```python
from skopt import gp_minimize
from skopt.space import Real, Integer, Categorical

# 定义搜索空间
space = [
    Real(1e-6, 1e-3, name='learning_rate', prior='log-uniform'),
    Integer(8, 64, name='batch_size'),
    Integer(100, 2000, name='warmup_steps'),
    Real(0.0, 0.2, name='dropout'),
]

def objective(params):
    """
    目标函数（最小化）
    """
    lr, batch_size, warmup_steps, dropout = params

    # 训练模型
    val_loss = train_model(
        learning_rate=lr,
        batch_size=batch_size,
        warmup_steps=warmup_steps,
        dropout=dropout
    )

    return val_loss  # 返回验证损失

# 运行贝叶斯优化
result = gp_minimize(
    objective,
    space,
    n_calls=20,  # 评估20组参数
```

```python
    random_state=42
)

print(f"Best parameters: {result.x}")
print(f"Best validation loss: {result.fun}")
```

12.5 调试技巧

12.5.1 单样本过拟合测试

```python
def sanity_check_overfit_single_batch(model, single_batch, num_steps=100):
    """
    Sanity check: 在单个batch上过拟合

    如果模型无法在单个batch上过拟合，说明有bug
    """
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

    losses = []

    for step in range(num_steps):
        optimizer.zero_grad()

        outputs = model(single_batch['input_ids'])
        loss = criterion(outputs, single_batch['labels'])

        loss.backward()
        optimizer.step()

        losses.append(loss.item())

        if step % 10 == 0:
            print(f"Step {step}, Loss: {loss.item():.4f}")

    # 检查是否成功过拟合
    if losses[-1] < losses[0] * 0.1:
        print("✓ 通过：模型能够在单个batch上过拟合")
    else:
        print("✗ 失败：模型无法过拟合，可能存在问题")
        print("  - 检查损失函数")
        print("  - 检查数据标签")
        print("  - 检查模型前向传播")

    return losses
```

12.5.2 激活值和权重检查

```python
def check_activations_and_weights(model, batch):
    """
    检查激活值和权重
    """
    model.eval()

    # 注册hook收集激活值
    activations = {}

    def get_activation(name):
        def hook(module, input, output):
            activations[name] = output.detach()
        return hook

    # 为每层注册hook
    for name, module in model.named_modules():
        module.register_forward_hook(get_activation(name))

    # 前向传播
    with torch.no_grad():
        _ = model(batch['input_ids'])

    # 检查
    issues = []

    for name, activation in activations.items():
        # 检查NaN
        if torch.isnan(activation).any():
            issues.append(f"{name}: 包含NaN")

        # 检查Inf
        if torch.isinf(activation).any():
            issues.append(f"{name}: 包含Inf")

        # 检查全零
        if (activation == 0).all():
            issues.append(f"{name}: 全零激活（可能是dead neurons）")

        # 检查范围
        act_max = activation.abs().max().item()
        if act_max > 1000:
            issues.append(f"{name}: 激活值过大（{act_max}）")

    # 检查权重
    for name, param in model.named_parameters():
        # 检查NaN
        if torch.isnan(param).any():
            issues.append(f"{name}: 权重包含NaN")
```

```python
        # 检查权重范围
        weight_std = param.std().item()
        if weight_std < 1e-5:
            issues.append(f"{name}: 权重方差过小（{weight_std}）")
        elif weight_std > 10:
            issues.append(f"{name}: 权重方差过大（{weight_std}）")

    if issues:
        print("发现问题:")
        for issue in issues:
            print(f"  - {issue}")
    else:
        print("✓ 激活值和权重检查通过")

    return activations, issues
```

12.5.3 常见错误排查清单

```python
debugging_checklist = {
    "Loss不下降": [
        "□ 检查学习率（太大或太小）",
        "□ 检查数据是否shuffle",
        "□ 检查标签是否正确",
        "□ 检查损失函数是否合适",
        "□ 尝试更简单的模型",
        "□ 检查输入是否normalize",
    ],

    "Loss变NaN": [
        "□ 降低学习率",
        "□ 添加梯度裁剪",
        "□ 检查数据中是否有NaN/Inf",
        "□ 使用Layer Normalization",
        "□ 检查除零错误",
        "□ 使用更稳定的数据类型（FP32）",
    ],

    "显存不足": [
        "□ 减小batch size",
        "□ 使用梯度累积",
        "□ 启用梯度检查点",
        "□ 使用混合精度",
        "□ 减小模型大小",
        "□ 使用ZeRO优化器",
    ],

    "训练太慢": [
```

```python
        "□ 增大batch size",
        "□ 使用多GPU训练",
        "□ 优化DataLoader（num_workers）",
        "□ 使用混合精度",
        "□ 编译模型（torch.compile）",
        "□ 检查是否有瓶颈操作",
    ],

    "过拟合": [
        "□ 增加Dropout",
        "□ 增加权重衰减",
        "□ 数据增强",
        "□ Early stopping",
        "□ 减小模型",
        "□ 增加训练数据",
    ]
}

def print_debugging_checklist(problem):
    """
    打印调试清单
    """
    if problem in debugging_checklist:
        print(f"\n{problem}排查清单：")
        for item in debugging_checklist[problem]:
            print(f"  {item}")
    else:
        print("未找到对应问题的清单")
```

12.6 本章小结

本章介绍了模型优化与调试的实用技巧：

✅ **训练诊断**：Loss曲线分析、梯度监控、学习率调优 ✅ **数据排查**：质量检查、去重、增强 ✅ **性能优化**：训练加速、显存优化 ✅ **超参调优**：网格搜索、贝叶斯优化 ✅ **调试技巧**：单样本过拟合、激活值检查

关键要点：

- 先检查数据，再怀疑模型
- 从简单开始调试（单样本过拟合）
- 监控梯度和激活值
- 系统化超参数搜索

完成！ 至此第四部分（评估与优化篇）全部完成。

下一部分预告：第五部分将介绍领域应用，第六部分补充系统设计和公司分析。