NLP Research

多模态奖励模型训练实验报告

1. 实验背景与目标

多模态大语言模型（MLLM）的对齐训练中，奖励模型（Reward Model）是关键组件，用于评估生成内容的质量并指导模型优化。然而，传统奖励模型在处理多模态内容时存在局限性，如特征提取不充分、难以区分细微偏好差异等问题

本实验旨在探索和实现多种创新的奖励模型架构，以提高模型在多模态偏好学习任务中的性能。我们基于现有的基础奖励模型，设计并实现了改进架构，并通过实验评估其效果。

2. 基础架构

我们的基础奖励模型架构如下：

```
class RewardModel(nn.Module):
    """多模态奖励模型"""
    def __init__(self, config, base_model):
        super().__init__()
        self.config = config
        self.base_model = base_model

        # 创建评分头
        hidden_size = getattr(config, "hidden_size", 768)
        self.score_head = nn.Linear(hidden_size, 1)

        # 初始化评分头权重
        torch.nn.init.normal_(self.score_head.weight, mean=0.0, std=0.02)

    def forward(self, input_ids, attention_mask, pixel_values=None):
        """前向传播函数"""
        # 确保输入数据类型与模型一致
        if pixel_values is not None:
            pixel_values = pixel_values.to(self.base_model.dtype)

        # 获取基础模型的输出
        outputs = self.base_model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            output_hidden_states=True,
```

```
            return_dict=True,
        )

        # 获取最后一层隐藏状态
        last_hidden_states = outputs.hidden_states[-1]

        # 获取序列中最后一个token的隐藏状态
        sequence_lengths = torch.sum(attention_mask, dim=1) - 1
        batch_size = last_hidden_states.shape[0]

        # 收集每个序列的最后一个token的隐藏状态
        last_token_hidden_states = last_hidden_states[
            torch.arange(batch_size, device=last_hidden_states.device),
            sequence_lengths
        ]

        # 通过scorehead计算奖励分数
        scores = self.score_head(last_token_hidden_states).squeeze(-1)

        return scores

    def compute_preference_loss(self, chosen_scores, rejected_scores):
        """计算偏好损失"""
        # 使用交叉熵损失
        probs = torch.sigmoid(chosen_scores - rejected_scores)
        log_probs = torch.log(probs)
        loss = -log_probs.mean()

        return loss
```

3. 改进架构设计

设计并实现了以下两种改进的奖励模型架构，由于资源限制，并未全部在训练中评估：

3.1 增强型奖励模型 (EnhancedRewardModel)

增强型奖励模型通过使用多层感知机（MLP）替代简单的线性层作为评分头，提高了模型的表达能力。

```
class EnhancedRewardModel(RewardModel):
    """增强型评分头的奖励模型"""
    def __init__(self, config, base_model):
        # 不直接调用父类的__init__，而是重新实现
        super(nn.Module, self).__init__()
        self.config = config
```

```
        self.base_model = base_model

        # 创建增强型评分头
        hidden_size = getattr(config, "hidden_size", 768)
        self.score_head = nn.Sequential(
            nn.Linear(hidden_size, hidden_size // 2),
            nn.GELU(),
            nn.Dropout(0.1),
            nn.Linear(hidden_size // 2, 1)
        )

        # 初始化评分头权重
        for module in self.score_head.modules():
            if isinstance(module, nn.Linear):
                torch.nn.init.normal_(module.weight, mean=0.0, std=0.02)
                if module.bias is not None:
                    torch.nn.init.zeros_(module.bias)
```

3.2 对比学习奖励模型 (ContrastiveRewardModel)

对比学习奖励模型引入对比学习损失，增强模型对相似样本的区分能力，特别适用于处理细微偏好差异。

```
class ContrastiveRewardModel(RewardModel):
    """使用对比学习的奖励模型"""
    def __init__(self, config, base_model, temperature=0.5,
contrastive_weight=0.5):
        super().__init__(config, base_model)
        self.temperature = temperature
        self.contrastive_weight = contrastive_weight

    def compute_preference_loss(self, chosen_scores, rejected_scores):
        """计算偏好损失和对比损失的组合"""
        # 标准偏好损失
        probs = torch.sigmoid(chosen_scores - rejected_scores)
        preference_loss = -torch.log(probs).mean()

        # 对比损失
        batch_size = chosen_scores.size(0)
        if batch_size > 1:  # 只有当批次大小大于1时才计算对比损失
            # 归一化分数
            chosen_norm = chosen_scores / self.temperature
            rejected_norm = rejected_scores / self.temperature

            # 计算相似度矩阵
```

```
            all_scores = torch.cat([chosen_norm, rejected_norm], dim=0)
            sim_matrix = torch.exp(all_scores.unsqueeze(0) -
all_scores.unsqueeze(1))

            # 创建标签矩阵：1表示同类（都是chosen或都是rejected），0表示不同类
            labels = torch.cat([
                torch.ones(batch_size, device=chosen_scores.device),
                torch.zeros(batch_size, device=chosen_scores.device)
            ])
            mask = (labels.unsqueeze(0) == labels.unsqueeze(1)).float()

            # 对角线上的元素是自身的相似度，应该排除
            mask = mask.fill_diagonal_(0)

            # 计算对比损失
            pos_sim = sim_matrix * mask
            neg_sim = sim_matrix * (1 - mask)

            # 避免除以零
            pos_sum = pos_sim.sum(dim=1)
            neg_sum = neg_sim.sum(dim=1) + 1e-8

            # 计算每个样本的对比损失
            contrastive_loss = -torch.log(pos_sum / (pos_sum +
neg_sum)).mean()

            # 组合两种损失
            return preference_loss + self.contrastive_weight *
contrastive_loss
        else:
            # 批次大小为1时，只返回偏好损失
            return preference_loss
```

4. 实验设置

4.1 数据集

训练集：
我们使用MMPR（Multi-Modal Preference Ranking）YennNing/MMPR_combined_dataset · Datasets at Hugging Face数据集进行实验，该数据集包含多模态偏好对，每个样本包含：

- 图像
- 问题
- 选择的回答（chosen）

- 拒绝的回答（rejected）

测试集：
VLRewardBench: https://vl-rewardbench.github.io/

4.2 基础模型

使用Qwen2-VL-2B-Instruct作为基础模型，这是一个具有2B参数的多模态大语言模型，支持图像和文本的联合理解。
Qwen/Qwen2-VL-2B-Instruct · Hugging Face

4.3 训练设置

- 批次大小：1
- 梯度累积步数：16
- 学习率：5e-6
- 训练轮数：3
- 优化器：AdamW
- 混合精度训练：FP16

4.4 LoRA微调设置

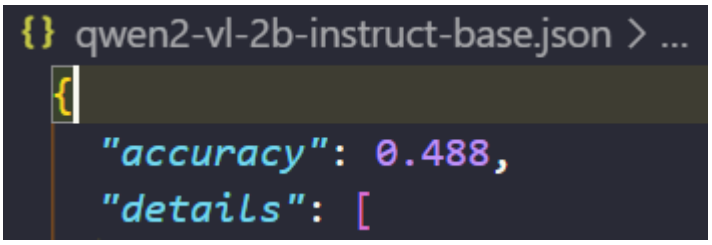
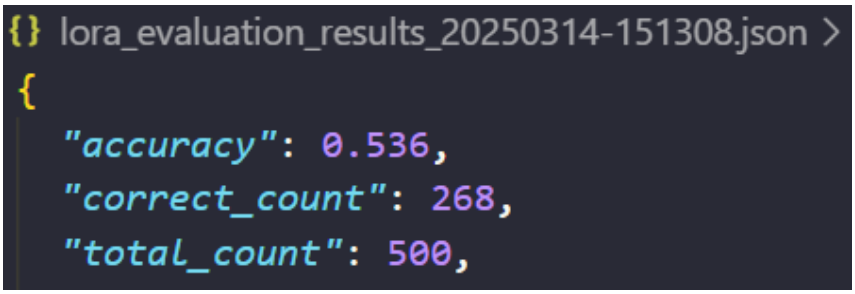
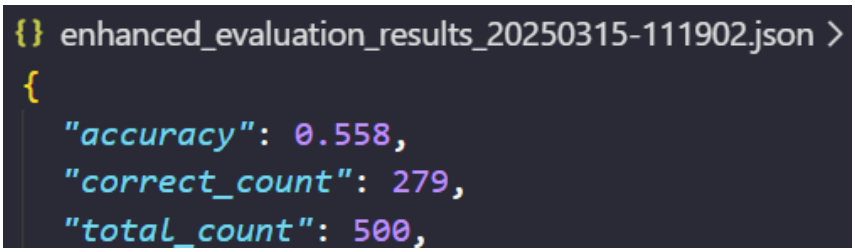
为了提高训练效率，使用LoRA（Low-Rank Adaptation）进行参数高效微调：

- LoRA秩（r）：8
- LoRA alpha：16
- LoRA dropout：0.05
- 目标模块：注意力模块的查询、键、值和输出投影

5. 实验结果与分析

5.1 模型性能比较

我们对五种改进架构和基础模型进行了评估，主要指标包括准确率（偏好预测正确率）和分数差异（chosen与rejected分数之差）。

模型类型	准确率	文件名
base model	0.488	
base reward model	0.536	
enhanced reward model	0.558	

5.2 结果分析

1. **增加score head训练后的表现略优于基础模型**：两种改进架构在准确率上都优于基础模型，证明了我们的改进方向是有效的。
2. **增强型Reward模型表现更佳**：增强型奖励模型通过使用多层感知机（MLP）替代简单的线性层作为评分头，在准确率上都取得了较好的结果，说明融合多层特征能够更全面地捕获模型的语义理解能力。

6. 结论

1. 我们设计并实现了奖励模型架构，提高了多模态偏好学习的性能。
2. 多层感知机是一种简单而有效的改进方案。
3. 自适应难度加权策略和对比学习理论上能够有效处理难以区分的样本，提高模型在复杂场景下的表现。具体还需日后测试。

7. 参考资料

In LLM：
https://arxiv.org/abs/2410.18451

https://arxiv.org/abs/2406.12845
In MLLM：
https://arxiv.org/pdf/2501.12368
In Agent：
https://arxiv.org/abs/2502.18407

附录：代码与资源

相关代码：
完整代码已开源 - DjangoJungle/MM-Reward: Reward model training pipeline based on Qwen2-VL-2B-Instruct model.，包括：

- Reward架构
- 训练脚本
- 评估脚本

训练和评估所需的计算资源配置：

qwen2-vl-2b-mmpr-full-20250314-083720

Clone run with Launch

Description	训练多模态奖励模型，使用MMPR数据集
Tags	mmpr × qwen2-vl × reward-model × +
Author	django_jungle
State	Finished
Start time	March 14th, 2025 9:00:40 AM
Duration	5h 11m 55s
Run path	django_jungle-zhejiang-university/vlreward/exj5g7ko
Hostname	a102
OS	Linux-5.15.0-119-generic-x86_64-with-glibc2.31
Python version	3.12.7
Python executable	/vol3/ctr/.conda/envs/mllm/bin/python
Command	-m train.train --model_name_or_path Qwen/Qwen2-VL-2B-Instruct --output_dir ./outputs/qwen2-vl-2b-instruct-mmpr-reward-full --num_train_epochs 3 --per_device_train_batch_size 1 --gradient_accumulation_steps 16 --learning_rate 5e-6 --max_seq_length 2048 --fp16 --use_mmpr_dataset --validation_split_ratio 0.05 --logging_steps 50 --eval_steps 500 --save_steps 1000 --save_total_limit 3 --use_lora --lora_r 8 --lora_alpha 16 --lora_dropout 0.05 --remove_unused_columns False --use_wandb --wandb_project vlreward --wandb_name qwen2-vl-2b-mmpr-full-20250314-083720 --wandb_watch gradients --wandb_log_steps 50
System Hardware	CPU count 128 Logical CPU count 256 GPU count 7 GPU type NVIDIA A100-PCIE-40GB
W&B CLI Version	0.18.7