Deepseek V2笔记

Decoder-Only架构，对FFN和Attention做出了改进

Attention的改进（MLA）：

1. 传统：需要$2 * len * nhead * d_{nhead}$的空间存储KV Cache
2. MLA: 将K和V进行低秩联合压缩，以减少KV缓存，类似LoRA。将KV以低维度缓存，使用时再通过投影层变换回去。

$$\begin{aligned}\mathbf{c}_t^{KV} &= W^{DKV}\mathbf{h}_t, \\ \mathbf{k}_t^{C} &= W^{UK}\mathbf{c}_t^{KV}, \\ \mathbf{v}_t^{C} &= W^{UV}\mathbf{c}_t^{KV},\end{aligned}$$

3. 旋转位置编码：略

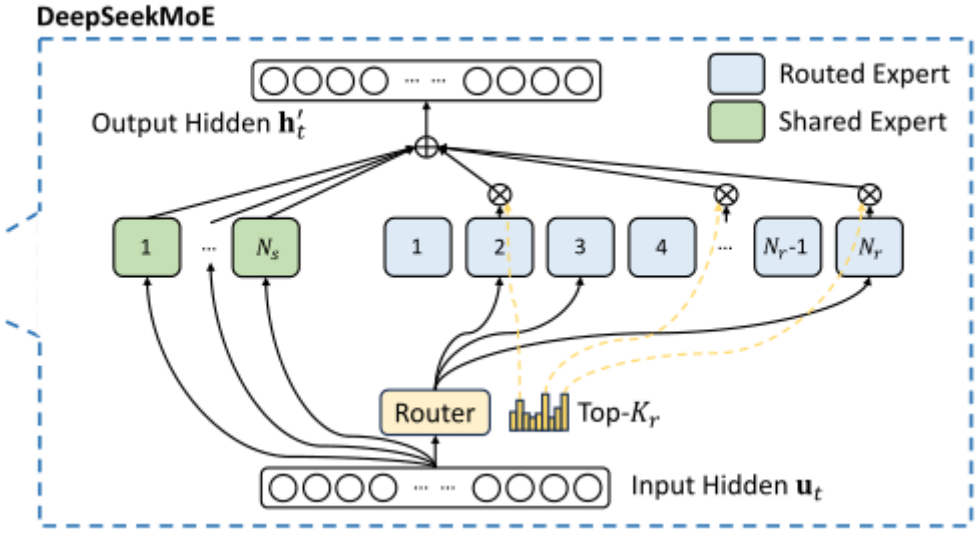
FFN(MoE):

1. MoE: 混合专家模型，在本模型架构中主要是在FFN层体现
2. 将专家分为共享专家与路由专家，以token为粒度：
    1. 共享专家：每个$input$必须要通过N_s个共享专家，无权重，直接相加。
    2. 路由专家：在N_r个路由专家中选取$Topk$个亲和度最高的专家激活，并计算每个激活专家归一化后的权重$g_{i,t}$，将输出按权重分配相加。
    3. **如何选择专家**：router的权重为$R^{n_routed_experts * gating_dim}$(gating_dim=h)，对每个token($R^{(bsz*l)*h}$)计算一个门控分数，再对最后一维做softmax得到分数结果$R^{(bsz*l)*n_routed_experts}$，每个token排序得到$topk$个亲和力最高的专家，进行后续运算。
    4. 专家通常分配到不同的硬件加速器上，为减少通信开销，额外要求每个token最多分布到M个设备上。

$$\mathbf{h}_t' = \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}\left(\mathbf{u}_t\right) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}\left(\mathbf{u}_t\right),$$

$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise}, \end{cases}$$

$$s_{i,t} = \mathrm{Softmax}_i\left(\mathbf{u}_t{}^T \mathbf{e}_i\right),$$



三个辅助损失函数

1. 专家级负载平衡：减少路由崩溃的风险
    1. 理想情况下，每个专家的负载应该为 $seq * topk / n\_router\_experts$
    2. 统计现实情况下的负载（对于一个样本）：$R^{n_router_experts}$统计每个experts使用的次数
    3. 现实负载除以理想负载，得到差异f_i，理想情况下应该为全1，现实情况应当部分大于1，部分小于1。
    4. 计算每个专家对于所有token的平均亲和度，记为$P_i \in R^{n_router_experts}$
    5. 二者点积后乘超参数得到损失
    6. 设计理念：既希望每个专家能够均衡负载，又需要考虑每个专家的贡献（负载均衡+贡献均衡）。

$$\mathcal{L}_{\text{ExpBal}} = \alpha_1 \sum_{i=1}^{N_r} f_i P_i,$$

$$f_i = \frac{N_r}{K_r T} \sum_{t=1}^{T} \mathbb{1}(\text{Token } t \text{ selects Expert } i),$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s_{i,t},$$

```
scores_for_aux = scores
aux_topk = self.top_k
# always compute aux loss based on the naive greedy topk method
topk_idx_for_aux_loss = topk_idx.view(bsz, -1)
scores_for_seq_aux = scores_for_aux.view(bsz, seq_len, -1)
ce = torch.zeros(
    bsz, self.n_routed_experts, device=hidden_states.device
)
ce.scatter_add_(
    1,
    topk_idx_for_aux_loss,
    torch.ones(bsz, seq_len * aux_topk, device=hidden_states.device),
).div_(seq_len * aux_topk / self.n_routed_experts)
aux_loss = (ce * scores_for_seq_aux.mean(dim=1)).sum(
    dim=1
).mean() * self.alpha
```

2. 设备级负载平衡：将专家分为D组，分配到D个设备上，类似于专家损失，此时负载差异值的计算仍是以专家为单位，只不过计算了每个设备上专家负载的均值
    1. 对于每台设备，计算$f_i^{'}$，即一台设备上每个专家的负载差异均值
    2. 对于每台设备，计算$P_i^{'}$，即一台设备上每个专家的平均亲和度之和
    3. 代码实现：

```
# TODO: 实现device_loss，假设按顺序将专家分组
# [bsz, num_experts] -> [bsz, n_groups]
alpha2 = self.alpha
experts_per_device = self.n_routed_experts // self.n_group
ce_groups = ce.view(bsz, self.n_group, experts_per_device).mean(dim=-1)
# [bsz, seq_len, num_experts] -> [bsz, num_experts] -> [bsz, n_groups]
p_groups = scores_for_seq_aux.mean(dim=1).view(bsz, self.n_group,
experts_per_device).sum(dim=-1)
# [bsz, num_groups] -> [bsz] -> value
device_loss = (ce_groups * p_groups).sum(dim=1).mean() * alpha2
```

$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^{D} f_i' P_i',$$

$$f_i' = \frac{1}{|\mathcal{E}_i|} \sum_{j \in \mathcal{E}_i} f_j,$$

$$P_i' = \sum_{j \in \mathcal{E}_i} P_j,$$

3. 通信级负载平衡：保证各设备的通信均衡，每个设备接收到的token数量也应该保持均衡
    1. 理想情况下，由于设备限制，每个token最多被发送到M台设备，则每台设备的接收量大约应为$M * seqlen/D$个token。
    2. 计算现实情况下每台设备的实际token接收量
    3. 后续流程相同
    4. 代码实现：

```
# TODO: 实现comm_loss
alpha3 = self.alpha
ce_comm = torch.zeros(
    bsz, self.n_routed_experts, device=hidden_states.device
)
# [bsz, n_groups]
ce_comm = ce_comm.scatter_add_(
    1,
    topk_idx_for_aux_loss,
    torch.ones(bsz, seq_len * aux_topk, device=hidden_states.device),
).view(bsz, self.n_group, experts_per_device).sum(dim=-1).div_(seq_len *
self.topk_group / self.n_group)
p_comm = p_groups
comm_loss = (ce_comm * p_comm).sum(dim=1).mean() * alpha3
```

$$\mathcal{L}_{\text{CommBal}} = \alpha_3 \sum_{i=1}^{D} f_i'' P_i'',$$

$$f_i'' = \frac{D}{MT} \sum_{t=1}^{T} \mathbb{1}(\text{Token } t \text{ is sent to Device } i),$$

$$P_i'' = \sum_{j \in \mathcal{E}_i} P_j,$$

总结

三个负载平衡机制层次由低至高，从底层的**专家平衡**、到中层的**设备平衡**、再到顶层的**通信平衡**共同作用，旨在优化MoE系统的资源利用并解决专家崩溃等问题，实现了强大的性能。

提交文件包括:

1. 小红书面试题1.pdf：阅读论文，代码，解决问题时的笔记记录
2. modeling_deepseek.py（line 490~512）：添加两个损失函数
3. test.py：损失函数外部测试