NSA ??????? by deepseek

1. NSA致力于实现硬件对齐的推理加速，通过特定的算法设计减少内存访问和硬件调度瓶颈，NSA速度在64k inference相较Flash Attention前向加速9倍，反向加速6倍
2. NSA的总体框架是通过更紧凑和信息密集的表示来替换原始的键值对
3. NSA有三种映射策略，分别是压缩（cmp）、选择（slc）和滑动窗口（win）。通过将不同策略得到的键值对进行组合

??

1. 引入 动态 选择和压缩历史的KV，减少计算量，符合实际的自然语言规律，但是
    1. 不一定完全匹配语言的表达逻辑
    2. 没有改变transformer的固有问题，多层信息不共享等
2. 一定程度上等价于增加一层attention，增加训练难度

??

[![image.png](NSA 稀疏注意力机制 by deepseek/image.png)](NSA 稀疏注意力机制 by deepseek/image.png)

假设上下文为64k时，如果我们取128个全局压缩KV，8个 512选择块KV和就近窗口4096个 KV，那么我们得到了压缩倍数7.88

[![image.png](NSA 稀疏注意力机制 by deepseek/Jkximage.png)](NSA 稀疏注意力机制 by deepseek/Jkximage.png)

1. **tokens**压缩：通过将连续的键或值块聚合为块级表示，得到压缩后的键值，从而捕获整个块的信息
    1.
    ```
    W_K_cmp = torch.randn(l, 1) #MLP: W2[1,4l]@(W1[4l, l]@X[l, d])
    W_V_cmp = torch.randn(l, 1)
    W_pe = torch.randn(l, dim)

    K_cmp = []
    V_cmp = []
    for i in range(max_idx):
        cur_K = K[:, i * d + 0: i * d + l , :] + W_pe.unsqueeze(0)
        cur_V = V[:, i * d + 0: i * d + l , :] + W_pe.unsqueeze(0)
        cur_K = cur_K.transpose(1, 2) @ W_K_cmp
        cur_V = cur_V.transpose(1, 2) @ W_V_cmp
    ```

```
    K_cmp.append(cur_K)
    V_cmp.append(cur_V)

K_cmp = torch.cat(K_cmp, dim = 2).transpose(1,2)
V_cmp = torch.cat(V_cmp, dim = 2).transpose(1,2)
print(K_cmp.shape) # torch.Size([1, 4, 16]) # 长度为32->4
print(V_cmp.shape) # torch.Size([1, 4, 16]) # 长度为32->4
```

2. **tokens**选择：仅使用压缩键值可能会丢失重要的细粒度信息，因此需要选择性地保留单个键值
    1.
    ```
    idx_slc_start = idx * d
    idx_slc_end = idx * d + l
    K_slc = torch.randn(batch_size, t, d * select_top_k, dim)
    V_slc = torch.randn(batch_size, t, d * select_top_k, dim)
    for i in range(batch_size):
        for j in range(t):
            for k in range(select_top_k):
                K_slc[i, j, k * d : k * d + l, :] = K[i, idx_slc_start[i, j, k ] :  idx_slc_end[i, j, k ] , :]
                V_slc[i, j, k * d : k * d + l, :] = V[i, idx_slc_start[i, j, k ] :  idx_slc_end[i, j, k ] , :]
    print(K_slc.shape) # bs, seq_len, select_kv, dim, 1,32,16,16, 不同t时刻选到不同的select_kv
    print(V_slc.shape) # bs, seq_len, select_kv, dim  1,32,16,16, 不同t时刻选到不同的select_kv
    ```
3. 滑动窗口：为了防止局部模式主导学习过程，影响模型从压缩和选择tokens中学习，NSA引入了专门的滑动窗口分支来处理局部context，窗口注意力是捕捉与当前q最近的kv片段，这里做了假设，即越相近的KV就越重要
    1.
    ```
    # built sliding window attention
    def get_window_mask(seq_len, window):
        mask = torch.ones(seq_len, seq_len)
        mask = torch.tril(mask)
        win_mask = torch.ones(seq_len - window, seq_len - window)
        win_mask = 1.0 - torch.tril(win_mask)
        mask[window:, :seq_len - window] = win_mask
        return mask
    print(get_window_mask(7, 3)) # test
    window_mask = get_window_mask(t, 8)
    ```
4. 注意力聚合：在上述三个注意力计算中，我们都得到了同样维度 `[1, 32, 16]` 的注意力输出

1.
```
o_list = [o_cmp, o_slc, o_win]
o_star = torch.zeros(batch_size, t, dim)
for i in range(3):
    o_star += gate[:, :, i].unsqueeze(2) * o_list[i]
print(o_star.shape)
```

????

[![image.png](NSA 稀疏注意力机制 by deepseek/jxLimage.png)](NSA 稀疏注意力机制 by deepseek/jxLimage.png)

Revision #6
Created 2025-03-08 08:42:58 UTC by Colin
Updated 2026-04-29 07:34:30 UTC by Colin