Phase 2

SYSTEMS FOR MACHINE LEARNING

EE 599

Authors:

Hengyi Tang (8016801929)
Caoyi Zou (2234578518)

Contents

1 LLaMA2 Model Inference

The KV caching only happens in the inference phase, so we need to modify all forward propagation in model.py.

In class $Attention(nn.Module)$, we removed all computations about $self.cache_k$ and $self.cache_v$, because this two variables selected specific keys and values to calculate the attention map. We replaced them with the original xk and xv.

```python
class Attention(nn.Module):
    """Multi-head attention module."""
    def __init__(self, args: ModelArgs):
        """
        Initialize the Attention module.

        Args:
            args (ModelArgs): Model configuration parameters.

        Attributes:
            n_kv_heads (int): Number of key and value heads.
            n_local_heads (int): Number of local query heads.
            n_local_kv_heads (int): Number of local key and value heads.
            n_rep (int): Number of repetitions for local heads.
            head_dim (int): Dimension size of each attention head.
            wq (ColumnParallelLinear): Linear transformation for queries.
            wk (ColumnParallelLinear): Linear transformation for keys.
            wv (ColumnParallelLinear): Linear transformation for values.
            wo (RowParallelLinear): Linear transformation for output.
            cache_k (torch.Tensor): Cached keys for attention.
            cache_v (torch.Tensor): Cached values for attention.

        """
        super().__init__()
        self.n_kv_heads = args.n_heads if args.n_kv_heads is None else args.n_kv_heads
        self.n_local_heads = args.n_heads
        self.n_local_kv_heads = self.n_kv_heads
        self.n_rep = self.n_local_heads // self.n_local_kv_heads
        self.head_dim = args.dim // args.n_heads

        self.wq = nn.Linear(args.dim, args.n_heads * self.head_dim, bias=False)
        self.wk = nn.Linear(args.dim, self.n_kv_heads * self.head_dim, bias=False)
        self.wv = nn.Linear(args.dim, self.n_kv_heads * self.head_dim, bias=False)
        self.wo = nn.Linear(args.n_heads * self.head_dim, args.dim, bias=False)

        # self.cache_k = torch.zeros(
        #     (
        #         args.max_batch_size,
        #         args.max_seq_len,
        #         self.n_local_kv_heads,
        #         self.head_dim,
        #     )
        # ).cuda()
        # self.cache_v = torch.zeros(
        #     (
        #         args.max_batch_size,
        #         args.max_seq_len,
        #         self.n_local_kv_heads,
        #         self.head_dim,
        #     )
        # ).cuda()
```

```python
def forward(
    self,
    x: torch.Tensor,
    # start_pos: int,
    freqs_cis: torch.Tensor,
    mask: Optional[torch.Tensor],
):
    """
    Forward pass of the attention module.

    Args:
        x (torch.Tensor): Input tensor.
        start_pos (int): Starting position for caching.
        freqs_cis (torch.Tensor): Precomputed frequency tensor.
        mask (torch.Tensor, optional): Attention mask tensor.

    Returns:
        torch.Tensor: Output tensor after attention.

    """
    bsz, seqlen, _ = x.shape
    xq, xk, xv = self.wq(x), self.wk(x), self.wv(x)

    xq = xq.view(bsz, seqlen, self.n_local_heads, self.head_dim)
    xk = xk.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)
    xv = xv.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)

    xq, xk = apply_rotary_emb(xq, xk, freqs_cis=freqs_cis)

    # self.cache_k = self.cache_k.to(xq)
    # self.cache_v = self.cache_v.to(xq)        You, 8小时前 • updata no kv caching
    # self.cache_k[:bsz, start_pos : start_pos + seqlen] = xk
    # self.cache_v[:bsz, start_pos : start_pos + seqlen] = xv

    # keys = self.cache_k[:bsz, : start_pos + seqlen]
    # values = self.cache_v[:bsz, : start_pos + seqlen]

    # repeat k/v heads if n_kv_heads < n_heads
    keys = repeat_kv(xk, self.n_rep)  # (bs, cache_len + seqlen, n_local_heads, head_dim)
    values = repeat_kv(xv, self.n_rep)  # (bs, cache_len + seqlen, n_local_heads, head_dim)

    xq = xq.transpose(1, 2)  # (bs, n_local_heads, seqlen, head_dim)
    keys = keys.transpose(1, 2) # (bs, n_local_heads, cache_len + seqlen, head_dim)
    values = values.transpose(1, 2) # (bs, n_local_heads, cache_len + seqlen, head_dim)
    scores = torch.matmul(xq, keys.transpose(2, 3)) / math.sqrt(self.head_dim)
    if mask is not None:
        scores = scores + mask  # (bs, n_local_heads, seqlen, cache_len + seqlen)
    scores = F.softmax(scores.float(), dim=-1).type_as(xq)
    output = torch.matmul(scores, values)  # (bs, n_local_heads, seqlen, head_dim)
    output = output.transpose(1, 2).contiguous().view(bsz, seqlen, -1)
    return self.wo(output)
```

In the last part, class $Llama(Generation)$, we modified the mask and reserved all elements so that the attention map remains rectangular.

```python
class Llama(Generation):
    def __init__(self, params: ModelArgs):

        self.freqs_cis = precompute_freqs_cis(
            # Note that self.params.max_seq_len is multiplied by 2 because the token limit for the Llama 2 genera
            # Adding this multiplier instead of using 4096 directly allows for dynamism of token lengths while tr
            self.params.dim // self.params.n_heads, self.params.max_seq_len * 2
        )

    def forward(self, tokens: torch.Tensor, start_pos: int):
        """
        Perform a forward pass through the Transformer model.

        Args:
            tokens (torch.Tensor): Input token indices.
            start_pos (int): Starting position for attention caching.

        Returns:
            torch.Tensor: Output logits after applying the Transformer model.

        """
        _bsz, seqlen = tokens.shape
        h = self.tok_embeddings(tokens)
        self.freqs_cis = self.freqs_cis.to(h.device)
        freqs_cis = self.freqs_cis[start_pos : start_pos + seqlen]

        mask = None
        if seqlen > 1:
            mask = torch.full(
                (seqlen, seqlen), float("-inf"), device=tokens.device
            )

            mask = torch.triu(mask, diagonal=1)

            # When performing key-value caching, we compute the attention scores
            # only for the new sequence. Thus, the matrix of scores is of size
            # (seqlen, cache_len + seqlen), and the only masked entries are (i, j) for
            # j > cache_len + i, since row i corresponds to token cache_len + i.
            mask = mask.unsqueeze(0).unsqueeze(1)  # (1, 1, seqlen, seqlen)
            # mask = torch.hstack([
            #     torch.zeros((seqlen, start_pos), device=tokens.device),
            #     mask
            # ]).type_as(h)

        for layer in self.layers:
            h = layer(h, start_pos, freqs_cis, mask)
        h = self.norm(h)
        output = self.output(h).float()
        return output        You, 9小时前 • updata no kv caching
```

The original output is shown as below. It can be found that all prompts output meaningful context.

```
[hengyita@d23-13 ml-systems-final-project-HelloworldHarley]$ python inference.py
/home1/hengyita/.local/lib/python3.11/site-packages/torch/__init__.py:747: UserWarning: torch.set_default_tensor_type() is dep
recated as of PyTorch 2.1, please use torch.set_default_dtype() and torch.set_default_device() as alternatives. (Triggered int
ernally at ../torch/csrc/tensor/python_tensor.cpp:431.)
  _C._set_default_tensor_type(t)
I believe the meaning of life is
> to learn to love.
Love is not a feeling. It is a decision.
I believe the meaning of life is to learn to love. Love is not a feeling. It is a decision. It is a commitment. You decide to
love. You commit to love. You make a choice to love

==================================

Simply put, the theory of relativity states that
> 1) the speed of light is constant for all observers and 2) the laws of physics are the same for all observers.
The theory of relativity is a very important concept in physics, but it is also one of the most misunderstood.
There are a lot of misconceptions about

==================================

A brief message congratulating the team on the launch:

        Hi everyone,

        I just
>
        <a href="https://www.google.com">Google</a>

        your website.

        I hope you enjoy the new look and feel.

        I'll be in touch soon to discuss your next project.

        Best

==================================

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        cheese =>
> fromage
        penguin => pinguin
        handbag => sac à main
        mug => tasse
        toothpaste => dentifrice
        t-shirt => tee-shirt
        pencil => crayon
        parrot => perroquet
```

The output that removes KV Caching is shown as below. The output becomes meaningless gibberish. It is speculated that the current model will consider future inputs, making the output logic confusing.

```
[hengyita@d23-13 ml-systems-final-project-HelloworldHarley]$ python inference.py
/home1/hengyita/.local/lib/python3.11/site-packages/torch/__init__.py:747: UserWarning: torch.set_default_tensor_type() is deprecated as o
 ../torch/csrc/tensor/python_tensor.cpp:431.)
  _C._set_default_tensor_type(t)
I believe the meaning of life is
> to do not a kids.
###include "f)

I thought I' s of the United States in the project.
            else, we can be included in the first half of the SIMERCIALIA, and the best way to the trucking the only, then his

==================================


Simply put, the theory of relativity states that
> 200000000000196.com/1820018.
Comment:
The Company, the way to a superfight, the world.
        return (11999499, but are in the 14

==================================


A brief message congratulating the team on the launch:

        Hi everyone,

        I just
> 20.
During the legalizing the best for a small businesses to the oil and a choice of the students and the actual retailer.
#162000001950001464, but this year, the first time.
    }

==================================


Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        cheese =>
> {
The most of the result in the grillance, 20000020, and the T.
    if the world.




==================================
```