Exercise 11 Part 1: Self-Attention

Summer Semester 2024

Author: Stefan Baumann (stefan.baumann@lmu.de)

Task: Implement Self-Attention

In this exercise, you will implement multi-head self-attention for a 2D sequence of tokens (shape `B D H W`) yourself using **only basic functions (no pre-made attention implementations!)**. You're allowed to use simple functions such as, e.g., `torch.bmm()`, `torch.nn.functional.softmax()`, ... and simple modules such as `torch.nn.Linear`.

Usage of functions provided by the `einops` library (such as `einops.rearrange()`) is also allowed and encouraged (but completely optional!), as it allows writing the code in a nice and concise way by specifying operations across axes of tensors as strings instead of relying on dimension indices. A short introduction into einops is available at https://nbviewer.org/github/arogozhnikov/einops/blob/master/docs/1-einops-basics.ipynb.

```
import math

import torch
import torch.nn as nn
import torch.nn.functional as F

# Optional
import einops

device = 'mps' if torch.backends.mps.is_available() else ('cuda' if
torch.cuda.is_available() else 'cpu')
print(f'Using device "{device}".')
```

```
Using device "mps".
```

```
class SelfAttention2d(nn.Module):
    def __init__(
        self,
        embed_dim: int = 256,
        head_dim: int = 32,
        value_dim: int = 32,
        num_heads: int = 8,
    ):
        """Multi-Head Self-Attention Module with 2d token input &
output
           Allows the model to jointly attend to information from
different representation subspaces.

        Args:
```

```
            embed_dim (int, optional): Dimension of the tokens at the
input & output (total dimensions of model). Defaults to 256.
            head_dim (int, optional): Per-head dimension of query &
key. Defaults to 32.
            value_dim (int, optional): Per-head dimension of values
(total number of features for values). Defaults to 32.
            num_heads (int, optional): Number of parallel attention
heads. Defaults to 6.
        """
        super().__init__()

        self.embed_dim = embed_dim
        self.head_dim = head_dim
        self.value_dim = value_dim
        self.num_heads = num_heads

        # Define linear layers for q/k/v/output
        self.q = nn.Linear(embed_dim, num_heads * head_dim)
        self.k = nn.Linear(embed_dim, num_heads * head_dim)
        self.v = nn.Linear(embed_dim, num_heads * value_dim)
        self.out = nn.Linear(num_heads * value_dim, embed_dim)

        self.softmax = nn.Softmax(dim=-1)

        self.scale = 1 / math.sqrt(self.head_dim)  # Scaling factor
for attention logits 1/sqrt(head_dim)
        self.init_weights()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Forward of multi-head self-attention
           The convention is that each head's part in q/k/v is
contiguous,
           i.e., if you want to get the query for head 0, it's at
q[..., :head_dim], head 1 is at q[..., head_dim:2*head_dim] ...

        Args:
            x (torch.Tensor): Input tensor of shape (B, D, H, W)
(batch, embedding dimension, height, width)

        Returns:
            torch.Tensor: Output tensor of shape (B, D, H, W) (batch,
embedding dimension, height, width)
        """
        B, D, H, W = x.shape    # Batch size, Channels, Height, Width
        N = H * W               # Number of tokens

        # Reshape input to (B, N, D) for linear projections
        x_flat = x.reshape(B, D, N).permute(0, 2, 1)  # Shape: (B, N,
D)
```

```
        # linear projections for q, k, v
        Q = self.q(x_flat)
        K = self.k(x_flat)
        V = self.v(x_flat)

        # Reshape and transpose to split heads
        Q = Q.reshape(B, N, self.num_heads, self.head_dim).permute(0,
2, 1, 3)  # Shape: (B, H, N, head_dim)
        K = K.reshape(B, N, self.num_heads, self.head_dim).permute(0,
2, 1, 3)  # Shape: (B, H, N, head_dim)
        V = V.reshape(B, N, self.num_heads, self.value_dim).permute(0,
2, 1, 3)  # Shape: (B, H, N, value_dim)

        # Compute attention scores with scaling of attention logits by
1/sqrt(head_dim)
        attn_scores = Q @ K.transpose(-2, -1) * self.scale
        attn_probs = self.softmax(attn_scores)


        # Apply attention to values
        attn_output = attn_probs @ V

        # Concatenate heads and reshape to (B, N, D)
        if self.head_dim > 1:
            attn_output = attn_output.permute(0, 2, 1,
3).contiguous().view(B, N, -1)
        else:
            attn_output = attn_output.squeeze(-1)

        # Apply output linear layer
        out = self.out(attn_output)

        # Reshape back to (B, D, H, W)
        out = out.permute(0, 2, 1).view(B, D, H, W)

        return out

    def init_weights(self):
        for m in [self.q, self.k, self.v, self.out]:
            nn.init.xavier_uniform_(m.weight)
            nn.init.zeros_(m.bias)

# Unit Test (single-head) DO NOT CHANGE!
with torch.no_grad():
    layer = SelfAttention2d(embed_dim=256, head_dim=256,
value_dim=256, num_heads=1).to(device)
    x = torch.randn((4, 256, 24, 24), device=device)
    res_layer = layer(x)

    layer_ref = nn.MultiheadAttention(layer.embed_dim,
```

```
layer.num_heads).to(device)
    layer_ref.load_state_dict({ 'in_proj_weight':
torch.cat([layer.q.weight, layer.k.weight, layer.v.weight]),
'out_proj.weight': layer.out.weight }, strict=False)
    res_ref = layer_ref(*[x.view(*x.shape[:2], -1).permute(2, 0, 1)] *
3)[0].permute(1, 2, 0).view(*x.shape)
    assert torch.allclose(res_layer, res_ref, rtol=1e-2, atol=1e-5),
'Single-head attention result incorrect.'

# Unit Test (multi-head) DO NOT CHANGE!
with torch.no_grad():
    layer = SelfAttention2d().to(device)
    x = torch.randn((4, 256, 24, 24), device=device)
    res_layer = layer(x)

    layer_ref = nn.MultiheadAttention(layer.embed_dim,
layer.num_heads).to(device)
    layer_ref.load_state_dict({ 'in_proj_weight':
torch.cat([layer.q.weight, layer.k.weight, layer.v.weight]),
'out_proj.weight': layer.out.weight }, strict=False)
    res_ref = layer_ref(*[x.view(*x.shape[:2], -1).permute(2, 0, 1)] *
3)[0].permute(1, 2, 0).view(*x.shape)
    assert torch.allclose(res_layer, res_ref, rtol=1e-2, atol=1e-5),
'Multi-head attention result incorrect.'

print('All tests passed.')
```

```
All tests passed.
```