

# Lecture 18: Multi Head Attention Part2

Instead of maintaining 2 separate classes; MultiHeadAttentionWrapper and CausalAttention, we combine both of these into a single MultiHeadAttention class.

## Step 1:

Start with the input (no. of batches, no. of tokens, d\_in=dim of the vector embedding)

```
b, num_tokens, d_in = x.shape
```

## Step 2:

Decide d\_out and number of heads.

In GPT-2 model, d\_in=d\_out.

head\_dimension = d\_out / num\_of\_heads

## Step 3:

Initialize trainable weight matrices for key, query, value (Wk, Wq, Wv)

Dimension will be: d\_in x d\_out

```
self.W_query = nn.Linear(d_in, d_out, bias=qkv_bias)
self.W_key = nn.Linear(d_in, d_out, bias=qkv_bias)
self.W_value = nn.Linear(d_in, d_out, bias=qkv_bias)
```

## Step 4:

Calculate Keys, Queries and Value matrix

(Input\*Wk, Input\*Wq, Input\*Wv)

Dimension: (no of batches, num\_tokens, d\_out)

```
keys = self.W_key(x) # Shape: (b, num_tokens, d_out)
queries = self.W_query(x)
values = self.W_value(x)
```

## Step 5:

Unroll last dimension of Keys, Queries and Values to include num\_heads and head\_dim

We know, head\_dimension = d\_out / num\_of\_heads

So, dimension of Keys, Queries and Values = (no of batches, num\_tokens, num\_of\_heads, head\_dimension)

Reshaped Queries matrix:

```
tensor([[[[-0.4888,  0.2361,  2.8463],
          [-0.2184,  5.4503, -1.8915]],
        [[-2.3531, -0.7912,  1.2578],
          [ 2.0534, -4.3369,  3.2125]],
        [[-2.5745,  0.2893,  1.1454],
          [ 0.9021,  1.5632,  0.6930]]]], dtype=torch.float64)
```

Output shape: (1, 3, 2, 3)

Code:

```
# Unroll last dim: (b, num_tokens, d_out) -> (b, num_tokens, num_heads, head_dim)
keys = keys.view(b, num_tokens, self.num_heads, self.head_dim)
values = values.view(b, num_tokens, self.num_heads, self.head_dim)
queries = queries.view(b, num_tokens, self.num_heads, self.head_dim)
```

## Step 6:

Group matrices by num\_of\_heads

$(\text{batches}, \text{num\_tokens}, \text{num\_heads}, \text{head\_dim}) \rightarrow (\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{head\_dim})$

```
keys = keys.transpose(1, 2)
queries = queries.transpose(1, 2)
values = values.transpose(1, 2)
```

$(1,3,2,3) \rightarrow (1,2,3,3)$

We do this so that we can do the parallel computing of attention scores more effectively.

#### Step 7:

Find attention scores: Queries\*Keys Transpose (2,3)



$(\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{head\_dim}) * (\text{b}, \text{num\_heads}, \text{head\_dim}, \text{num\_tokens}) \rightarrow (\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{num\_tokens})$

#### Step 8:

Find attention weights: Mask attention scores to implement causal attention

Divide by sqrt of head\_dim

```
# Compute scaled dot-product attention (aka self-attention) with a causal mask
attn_scores = queries @ keys.transpose(2, 3) # Dot product for each head

# Original mask truncated to the number of tokens and converted to boolean
mask_bool = self.mask.bool()[:num_tokens, :num_tokens]

# Use the mask to fill attention scores
attn_scores.masked_fill_(mask_bool, -torch.inf)

attn_weights = torch.softmax(attn_scores / keys.shape[-1]**0.5, dim=-1)
attn_weights = self.dropout(attn_weights)
```

#### Step 9:

Context vector = Attention weights \* Values

$= (\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{num\_tokens}) * (\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{head\_dim})$

$= (\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{head\_dim})$

But there's a problem here. We need to merge the num\_heads and head\_dim back together. Because the resultant context vector should have dimension d\_out. The d\_out dimension must be preserved.

#### Step 10:

Reformat context vectors:

$(\text{b}, \text{num\_heads}, \text{num\_tokens}, \text{head\_dim}) \rightarrow (\text{b}, \text{num\_tokens}, \text{num\_heads}, \text{head\_dim})$

```
tensor([[[[-3.6194,  2.0935,  1.3879],
          [ 1.5961, -0.9367, -0.3400]],
        [[-1.8728,  1.6494,  0.7163],
          [ 1.3712,  0.7805,  0.2233]],
        [[-0.3553, -0.5607, -0.1181],
          [ 0.6997, -0.4487, -0.0530]]]])
```

```
# Shape: (b, num_tokens, num_heads, head_dim)
context_vec = (attn_weights @ values).transpose(1, 2)
```

So the final shape = (b, num\_tokens, d\_out)

## DETAILED EXPLANATION OF THE MULTI-HEAD ATTENTION CLASS

---

The splitting of the query, key, and value tensors, is achieved through tensor reshaping and transposing operations using PyTorch's `.view` and `.transpose` methods.

The input is first transformed (via linear layers for queries, keys, and values) and then reshaped to represent multiple heads.

---

The key operation is to split the `d_out` dimension into `num_heads` and `head_dim`, where `head_dim = d_out / num_heads`.

This splitting is then achieved using the `.view` method: a tensor of dimensions (b, num\_tokens, d\_out) is reshaped to dimension (b, num\_tokens, num\_heads, head\_dim).

---

The tensors are then transposed to bring the `num_heads` dimension before the `num_tokens` dimension, resulting in a shape of (b, num\_heads, num\_tokens, head\_dim).

This transposition is crucial for correctly aligning the queries, keys, and values across the different heads and performing batched matrix multiplications efficiently.

---