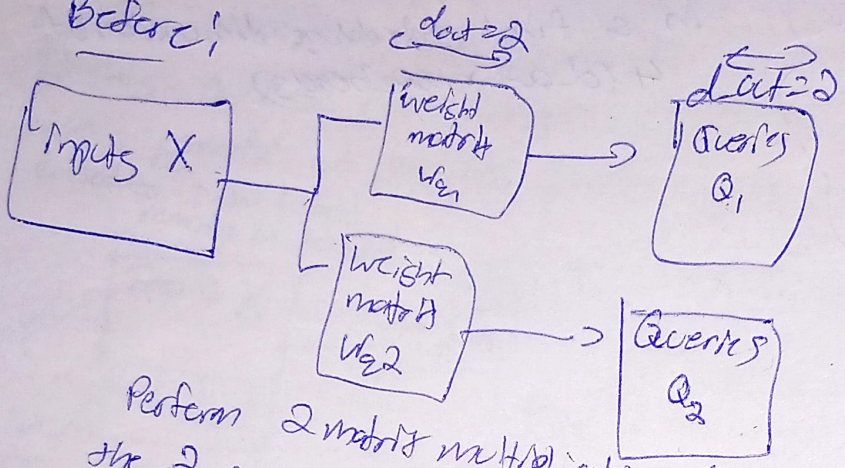
MULTI-Head Attention part 2

b) Implementing multi-head attention w/ weight splits

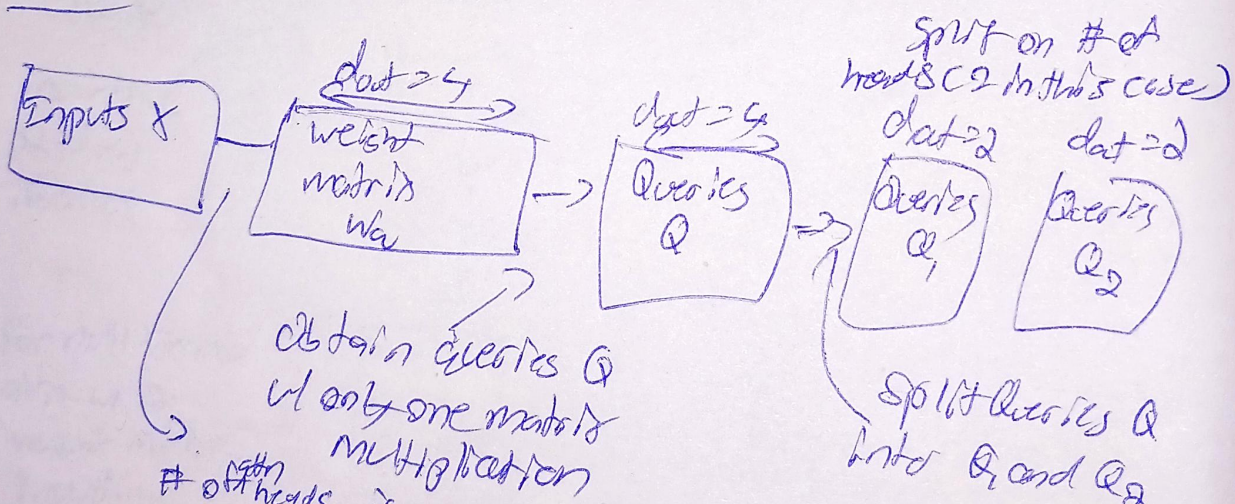
c) Instead of maintaining 2 separate classes; MultiHead Attention Wrapper and Casual Attention, we combine both of these into a single MultiAttention class

Before:



Perform 2 matrix multiplications to obtain the 2 query matrices, Q_1 and Q_2

After



# of attn heads is specified

Head dimension = $\frac{d_{out}(4)}{n\_heads(2)}$

* Let us start w/ simple example;

Step ①: Start w/ input

b, num_tokens, d_in = (1, 3, 6)

the ~ 6 (first row)
cat ~ 6 (second row)
sleeps ~ 6 (third row)

In GPT models d_in = d_out typically

x = torch.tensor([[[...]]])
↳ batch = 1, num_tokens = 3, d_in = 6

(1 × 3 × 6)

CV

[ ] 3 × d_out

Step ② Decide d_out, num_heads → head_dim = d_out / num_heads = 6/2 = 3

Step ③: Initialize trainable weight matrices

for keys, queries, value (W_q, W_k, W_v)

last dim of input (6 × 6) d_in × d_out

W_q = [ ] (6 × 6)

W_k = [ ] (6 × 6)

W_v = [ ] (6 × 6)

Step ④: Calculate keys, queries, value matrix (input @ W_q, input @ W_k, input @ W_v)

(1 × 3 × 6) → (6 × 6)

[ ] keys (1 × 3 × 6)
b, num_tokens, d_out

[ ] Queries (1 × 3 × 6)

[ ] Values (1 × 3 × 6)

Multiplying

(1 × 3 × 6) @ (6 × 6)

compare

(3 × 6) @ (6 × 6) → (3 × 6)

Add batch,

(1 × 3 × 6)

Step ⑤: Unroll last dimension of keys, queries, and values to include num_heads and head_dim

head_dim = d_out / num_heads = 6/2 = 3

(b, num_tokens, d_out) -> (b, num_tokens, num_heads, head_dim)

(1, 3, 6) -> (1, 3, 2, 3) "Essentially splitting that 6 dim d_out into 2 attn heads, each with 3 dimensions (head_dim)"

Reshaped Queries matrix

Queries matrix -> (1, 3, 2, 3)

```
            3 head dim
tensor([[[[ — — — ]      } token 1
head 1
head 2       [ — — — ]],

head 1      [[ — — — ]   } token 2
head 2       [ — — — ]],

head 1      [[ — — — ]   } token 3
head 2       [ — — — ]]]],
              3 dim head
```

Queries, keys, values all have same dim.

Step ⑥: Group matrices by "number of heads"

(b, num_tokens, num_heads, head_dim) -> (b, num_heads, num_tokens, head_dim)

(1, 3, 2, 3) -> (1, 2, 3, 3)

Groupings w.r.t # of tokens — Groupings w.r.t # of heads

Transposed Queries matrix

```
[[[[ — — — ],   -> head 1
   [ — — — ],    } Each row here represents a token
   [ — — — ],

  [[ — — — ],   -> head 2
   [ — — — ],
   [ — — — ]]]]
```

Queries, keys, values all have same dimensions

Since we can now group w.r.t to heads, we can compute attn score for each head separately

Step 7: Find attention scores

Queries @ keys.Transpose(2,3) ← Transpose the last 2 dimensions (num_tokens and head_dim)

keys (old)

[[[[1 2 3],
[4 5 6],
[7 8 9]],

[[1 2 3],
[4 5 6],
[7 8 9]]]]

New keys.transpose(2,3) =

[[[[1 4 7],
[2 5 8],
[3 6 9]], } head 1

[[1 4 7],
[2 5 8],
[3 6 9]]]] } head 2

(row from old becomes column)

Queries × keys.T
(1, 2, 3, 3) & (1, 2, 3, 3)

~~(b, #de~~

(b, #head, #tok, head_dim) & (b, num_head, head_dim, num_tokens)

↳ (1, 2, 3, 3)

Queries @ keys.Transpose(2,3) yields attn scores matrix (b, num_heads, num_tokens, num_tokens)

matrix mul. on last 2 dims is:
(num_tokens, head_dim) × (head_dim, num_tokens)
↓
(num_tokens, num_tokens)

the cat sleeps

the [[[[- - -],
cat [- - -],
sleeps [- - -]], } head 1

first row is attn score of first word between all other ones

the cat sleeps

the [[- - -],
cat [- - -],
sleeps [- - -]]]] } head 2

Step 8: Find attention weights

↳ Mask attention scores to implement causal attention

[[[[- -inf -inf],
[- - -inf],
[- - -]],

[- -inf -inf],
[- - -inf],
[- - -]]]]

↳ Divide by $\sqrt{head\_dim}$ $= \sqrt{d_{out}/num\_heads}$ $= \sqrt{6/2} = \sqrt{3}$

Make sure that variance scales up w/ # of dimensions and divide to prevent variance from blowing up → stable gradients

Divide by $\sqrt{3}$ ←

[[[[$\frac{-}{\sqrt{3}}$, -inf, -inf],
[$\frac{-}{\sqrt{3}}$, $\frac{-}{\sqrt{3}}$, -inf],
[$\frac{-}{\sqrt{3}}$, $\frac{-}{\sqrt{3}}$, $\frac{-}{\sqrt{3}}$]],

" "

↳ apply softmax on next page

softmax

[[[[1, 0, 0],
[0.9641, 0.0359, 0],
[0.0417, 0.2603, 0.6980]]]]

attention weights
(1, 3, 3, 3)
(b, num_heads, num_tokens, num_tokens)

[1, 0, 0],
[.1722, .8278, 0],
[.3887, .4627, .1566]]]]

can also implement dropout after softmax.

Step (3): context vector = Attention (2) values
weights
(b, num_heads, num_tokens, num_tokens)

(b, num_tokens, num_heads, head_dim)

Attn_weights
Attn_scores
(1, 2, 3, 3)

values
(1, 2, 3, 3)

(num_tokens, num_tokens)

(3)
(num_tokens, head_dim)
↓
(num_tokens, head_dim)

↓
(1, 2, 3, 3)

(b, num_heads, num_tokens, head_dim)

[[[[- - -],
[- - -],
[- - -]],

[[- - -],
[- - -],
[- - -]]]]

Each row corresponds to one token, and represents context_vec for that particular token, dimensions = to head_dim

we want d_out as resultant dim. but context_vec, so need to flip back num_tokens and num_heads dim transpose

Step (10): Reformat context vectors

(b, num_heads, num_tokens, head_dim) -> (b, num_tokens, num_heads, head_dim)

[[[[1 2 3]
  [4 5 6]]]] token 1

[[[7 8 9]
  [10 11 12]]] token 2

[[[13 14 15]
  [16 17 18]]] token 3

(1, 3, 2, 3) → (b, #tokens, #heads, head_dim)

Step (11): Combine heads → Flatten each token output into each row

First row consists of merging the 2 heads, for first token

Second row consists of merging the 2 heads, for second token

same for 3rd token/row.

[[[[1 2 3 4 5 6]
  [7 8 9 10 11 12]
  [13 14 15 16 17 18]]]]

(b, num_tokens, d_out)

(1, 3, 6)

Final result!