

# Assignment 1: Debugging Transformer Models with PyCharm and WSL

43 Snapshots with Shape & Role Explanations (Encoder–Decoder Transformer)

## Objective

Debugger-based walkthrough of a standard encoder–decoder Transformer using **PyCharm + WSL**. We include **43 numbered snapshots**, each with the exact variable, the **shape**, and a short explanation of **what it represents and why the shape is correct**.

## Environment (PyCharm + WSL) — Optional but Recommended

If available, include:

- **Env-1:** PyCharm interpreter set to WSL (Screenshot: env1.png).
- **Env-2:** Debugger hit with Variables pane visible (Screenshot: env2.png).

## Model & Data Compliance

- **Transformer:** Encoder–decoder (Vaswani et al., 2017), **2** encoder layers, **2** decoder layers, **4** heads,  $d_{\text{model}} = 128$ ,  $d_{\text{ff}} = 256$ .
- **Source (6 tokens):** hello from cairo egypt team project
- **Target (5 tokens):** we are debugging transformer today
- **Shapes used throughout:** batch = 1; src\_len = 6; tgt\_len = 5; heads = 4;  $d_{\text{head}} = d_{\text{model}}/4 = 32$ .

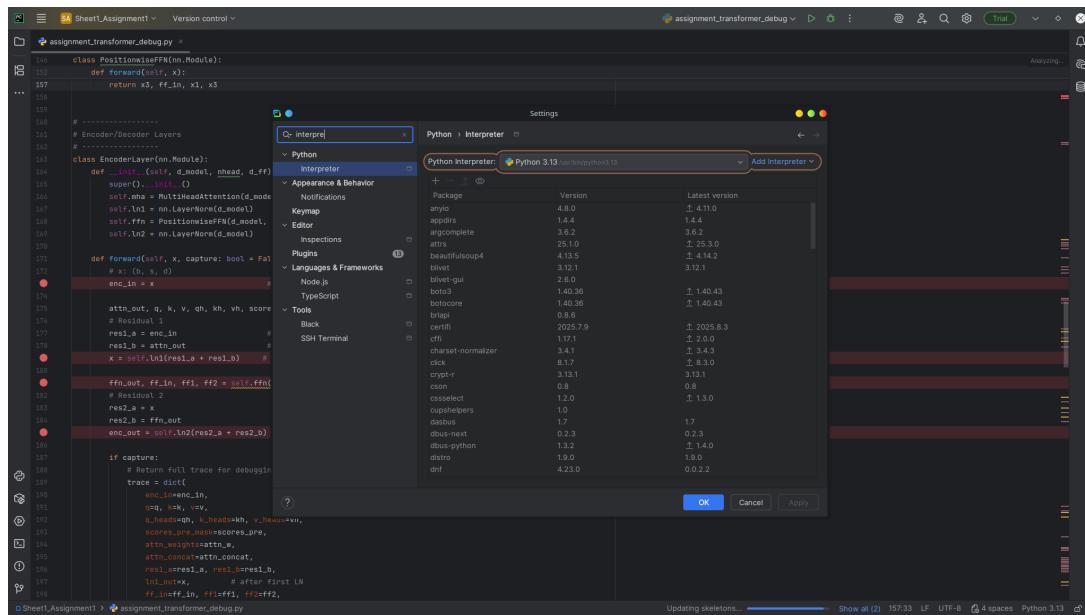


Figure 1: PyCharm interpreter configuration on Linux (instead of WSL).

# Input & Embedding (Snapshots 1–5)

The screenshot shows the PyCharm IDE in debug mode. The code editor displays a Python script named `assignment_transformer_debug.py`. A red highlight bar spans across several lines of code, specifically highlighting the assignment of `src_ids` and `tgt_ids`. The `src_ids` assignment is annotated with a comment: `# Snapshot #1 - Raw input tokens (IDs) src_ids: [3, 2, 0, 1, 5, 4]`. The `tgt_ids` assignment is also annotated with a comment: `# Snapshot #2 - Target tokens (IDs)`. Below the code editor, the debugger's sidebar shows the current stack frame: `<module>, assignment_transformer_debug.py:51`. In the bottom right corner of the debugger pane, there is a status bar with the text "5:1 LF UFT-8 4 spaces Python 3.13".

Snapshot #1 — Raw input tokens (IDs)

## Explanation & Shape

**Variable (as shown in PyCharm):** `src_ids`  
**Observed shape:** `(6,)`

Integer IDs for each source token; length equals the source sequence length (6). This is the discrete input to the embedding lookup.

```

41     src_vocab = build_vocab_from_sentences(SRC_SENTENCE)    src_vocab: {'cairo': 0, 'egypt': 1, 'from': 2, 'hello': 3, 'project': 4, 'team': 5}
42     tgt_vocab = build_vocab_from_sentences(TGT_SENTENCE)   tgt_vocab: {'are': 0, 'debugging': 1, 'today': 2, 'transformer': 3, 'we': 4}
...
44     def encode(sentence, vocab):  2 usages
45         return [vocab[w] for w in sentence.strip().split()]
46
47     def decode(ids, inv_vocab):
48         return " ".join(inv_vocab[i] for i in ids)
49
50     src_ids = encode(SRC_SENTENCE, src_vocab)      # SNAPSHOT #1 - Raw input tokens (IDs)  src_ids: [3, 2, 0, 1, 5, 4]
51     tgt_ids = encode(TGT_SENTENCE, tgt_vocab)       # SNAPSHOT #2 - Target tokens (IDs)  tgt_ids: [4, 0, 1, 3, 2]
52
53     src_vocab_size = len(src_vocab)
54     tgt_vocab_size = len(tgt_vocab)
55
56     # Inverse vocabs (optional, handy in debugger)
57     inv_src_vocab = {i:w for w,i in src_vocab.items()}
58     inv_tgt_vocab = {i:w for w,i in tgt_vocab.items()}
59
60     # -----
61     # Hyperparameters
62     # -----
63     @dataclass  2 usages

```

Threads & Variables    Console

MainThread <module>.assignment\_transformer\_debug.py:53

- src\_vocab = {0: 'cairo', 1: 'egypt', 2: 'from', 3: 'hello', 4: 'project', 5: 'team'}
- src\_ids = [3, 2, 0, 1, 5, 4]
- tgt\_ids = [4, 0, 1, 3, 2]
- src\_vocab\_size = 6
- tgt\_vocab\_size = 5
- inv\_src\_vocab = {3: 'cairo', 2: 'egypt', 0: 'from', 1: 'hello', 5: 'project', 4: 'team'}
- inv\_tgt\_vocab = {4: 'are', 0: 'debugging', 1: 'today', 2: 'transformer', 3: 'we'}
- len\_ = 5

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

## Snapshot #2 — Target tokens (IDs)

### Explanation & Shape

**Variable (as shown in PyCharm):** tgt\_ids

**Observed shape:** (5,)

Integer IDs for each target token; length equals the target sequence length (5) used for teacher forcing.

```

265 class TinyTransformer(nn.Module):  1 usage
...
283     def forward(self, src_ids, tgt_ids):  self: TinyTransformer(\n        (src_emb): Embedding(6, 128)\n        (tgt_emb): Embedding(5, 128)\n        (pos): PositionalEncoding()\n        (enc_layers):\n            ...
284         # src_ids, tgt_ids: lists of ints
285         device = next(self.parameters()).device  device: device(type='cpu')
286         src = torch.tensor(src_ids, dtype=torch.long, device=device).unsqueeze(0)  # (1, S)  src: tensor([3, 2, 0, 1, 5, 4])
287         tgt = torch.tensor(tgt_ids, dtype=torch.long, device=device).unsqueeze(0)  # (1, T)  tgt: tensor([4, 0, 1, 3, 2])
288
289         # Embedding weights
290         emb_weight_slice = self.src_emb.weight[:5, :5]  # SNAPSHOT #3 - Embedding weight matrix (slice)  emb_weight_slice: tensor([-0.8201,  0.3956,  0.8989, -1.3884, -0.3671])
291
292         # Lookups
293         src_embed = self.src_emb(src)  # SNAPSHOT #4 - Input embeddings after lookup
294         src_with_pos = self.pos(src_embed)  # SNAPSHOT #5 - After adding positional encoding
295
296         tgt_embed = self.tgt_emb(tgt)
297         tgt_with_pos = self.pos(tgt_embed)
298
299         # Encoder stack
300         x = src_with_pos
301         enc_traces = []
302         for li, enc in enumerate(self.enc_layers):
303             cap = (li == 0)  # capture a full trace for the FIRST encoder block only

```

Threads & Variables    Console

MainThread <module>.forward\_assignment\_transformer\_debug.py:293

- ndim = 2
- output\_nr = 0
- real = Tensor(5, 5) tensor([-0.8201, 0.3956, 0.8989, -1.3884, -0.1670], [ 0.5689, 0.1333, 0.5600, -0.7351, -0.2240], [ 0.0083, -0.0045, 1.0061, -1.7277, -1.4915], [ 0.3811, 0.3953, -...View as Array
- requires\_grad = True
- retains\_grad = False
- shape = (Size: torch.Size(5, 5))
- 0 = (int) 5
- 1 = (int) 5
- 2 = (int) 2
- > Protected Attributes
- > Protected Attributes
- self = (TinyTransformer) TinyTransformer(\n (src\_emb): Embedding(6, 128)\n (tgt\_emb): Embedding(5, 128)\n (pos): PositionalEncoding()\n (enc\_layers): M... LayerNorm(128), eps=1e-05, elementwise\_affine=True)\n View

## Snapshot #3 — Embedding weight matrix (slice)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["emb_weight_slice"]`

**Observed shape:** (5,5) slice of full (src\_vocab\_size, 128)

A small slice of the learnable embedding table. Rows = vocabulary items; columns = 128 embedding features. The full table is (vocab, 128).

## Snapshot #4 — Input embeddings after lookup

## Explanation & Shape

Variable (as shown in PyCharm): `refs["src_embed"]`

Observed shape: (1,6,128)

Each source token ID is mapped to a 128-d vector. Batch size is 1, sequence length is 6, and  $d_{\text{model}} = 128$ .

The screenshot shows the PyCharm debugger interface during the execution of `assignment_transformer.debug.py`. The code being run is the `forward` method of the `TinyTransformer` class. The current line of code is highlighted at line 296: `tgt_embed = self.tgt_emb(tgt)`. The variable `tgt` is highlighted in blue, indicating it is the current focus. The variable `src_with_pos` is also highlighted in blue, showing its value as a tensor of shape (1, 6, 128). The variable `tgt_ids` is shown as a list [4, 0, 1, 3, 2]. The PyCharm debugger's bottom panel displays the current stack trace, which includes frames from `forward`, `assignment_transformer.debug.py:296`, and `<module>`.

## Snapshot #5 — Embeddings after adding positional encoding

### Explanation & Shape

**Variable (as shown in PyCharm):** `refs["src_with_pos"]`

**Observed shape:** (1,6,128)

Sinusoidal positional encodings are added element-wise to embeddings to encode order; shape remains (1, 6, 128).

## Encoder (One Block, Full Trace) — Snapshots 6–19

The screenshot shows the PyCharm IDE interface during a debugging session. The top navigation bar includes tabs for 'SheetAssignment' and 'assignment\_transformer\_debug.py'. The main editor window displays the 'EncoderLayer' class definition and its implementation. A red dot at line 171 indicates the current execution point. The bottom status bar shows the file path 'assignment\_transformer\_debug.py' and the line number '171'. The bottom-left toolbar contains icons for 'File', 'Edit', 'Run', 'Tools', 'View', 'Help', and 'File Structure'. The bottom-right corner shows the Python version 'Python 3.13'.

```
163     class EncoderLayer(nn.Module): 1 usage
164         def __init__(self, d_model, nhead, d_ff):
165             super().__init__()
166             self.mha = MultiHeadAttention(d_model, nhead)
167             self.ln1 = nn.LayerNorm(d_model)
168             self.ffn = PositionwiseFFN(d_model, d_ff)
169             self.ln2 = nn.LayerNorm(d_model)
170
171         def forward(self, x, capture: bool = False):  capture: True    self: EncoderLayer(\n            mha: MultiHeadAttention(\n                W_q: Linear(in_features=128, out_features=128, bias=True)\n                W_k: Linear(in_features=128, out_features=128, bias=True)\n                W_v: Linear(in_features=128, out_features=128, bias=True)\n                W_o: Linear(in_features=128, out_features=128, bias=True)\n            )\n            ln1: LayerNorm(\n                normalized_shape=d_model\n            )\n            ffn: PositionwiseFFN(\n                d_model=d_model,\n                d_ff=d_ff\n            )\n            ln2: LayerNorm(\n                normalized_shape=d_model\n            )\n        )
172
173         # x: (b, s, d)
174
175         enc_in = x
176
177         attn_out, q, k, v, qh, kh, vh, scores_pre, attn_w, attn_concat = self.mha(x, x, x, mask=None)
178
179         # Residual 1
180
181         res1_a = enc_in + attn_out  # for "Residual connection tensors" (branch A)
182         res1_b = attn_out  # branch B (attention output)
183         x = self.ln1(res1_a + res1_b)  # SNAPSHOT #14 - Residual connection tensors; SNAPSHOT #15 - Layer norm output
184
185         ffn_out, ff_in, ff1, ff2 = self.ffn(x)
186
187         # Residual 2
188
189         res2_a = x
190         res2_b = ffn_out
191
192         enc_out = self.ln2(res2_a + res2_b)  # SNAPSHOT #19 - Encoder block final output tensor
193
194
195     
```

## Snapshot #6 — Encoder block input tensor

## Explanation & Shape

Variable (as shown in PyCharm): refs["enc\_traces"][0]["enc\_in"]  
Observed shape: (1,6,128)

Tensor entering the first encoder layer (embeddings + position). Per-token dimension remains 128.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109     # x: (batch, seq, d_model)
110     b, s, _ = x.size()
111     x = x.view(b, s, self.nhead, self.d_head).transpose(dim0=1, dim1=2) # (batch, heads, seq, d_head)
112     return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  K_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00,
115     # q_in, k_in, v_in: (batch, seq, d_model)
116     q = self.W.q(q_in) # SNAPSHOT #7 or #21 or #30 - Q (depending on caller)  q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
117     k = self.W.k(k_in) # SNAPSHOT #8 or #22 or #31 - K
118     v = self.W.v(v_in) # SNAPSHOT #9 or #23 or #32 - V
119
120     q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)
121     k_heads = self._split_heads(k)
122     v_heads = self._split_heads(v)
123
124     # Scaled dot-product attention
125     # pre-softmax scores (no mask yet)
126     scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b, h, tgt_len, src_len)
127     # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
128
129     scores = scores_pre_mask.clone()

```

Threads & Variables    Console

MainThread

- forward\_assignment\_transformer\_debug.py:117
- ...\_wrapped\_call\_impl\_module.py:1734
- forward\_assignment\_transformer\_debug.py:175
- ...\_call\_impl\_module.py:1784
- ...\_wrapped\_call\_impl\_module.py:1773
- forward\_assignment\_transformer\_debug.py:304
- ...\_call\_impl\_module.py:1784
- ...\_wrapped\_call\_impl\_module.py:1773
- main\_assignment\_transformer\_debug.py:340
- <module>.assignment\_transformer\_debug.py:345

Sheet\_Assignment1 > assignment\_transformer\_debug.py

## Snapshot #7 — Self-attention queries (Q)

### Explanation & Shape

**Variable (as shown in PyCharm):** refs["enc\_traces"][0]["q"]

**Observed shape:** (1,6,128)

Q is a linear projection of the encoder input. Q, K, and V share the same shape so dot-product attention is valid in the same representation space.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109     # x: (batch, seq, d_model)
110     b, s, _ = x.size()
111     x = x.view(b, s, self.nhead, self.d_head).transpose(dim0=1, dim1=2) # (batch, heads, seq, d_head)
112     return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  K_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00,
115     # q_in, k_in, v_in: (batch, seq, d_model)
116     q = self.W.q(q_in) # SNAPSHOT #7 or #21 or #30 - Q (depending on caller)  q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
117     k = self.W.k(k_in) # SNAPSHOT #8 or #22 or #31 - K
118     v = self.W.v(v_in) # SNAPSHOT #9 or #23 or #32 - V
119
120     q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)
121     k_heads = self._split_heads(k)
122     v_heads = self._split_heads(v)
123
124     # Scaled dot-product attention
125     # pre-softmax scores (no mask yet)
126     scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b, h, tgt_len, src_len)
127     # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
128
129     scores = scores_pre_mask.clone()

```

Threads & Variables    Console

MainThread

- forward\_assignment\_transformer\_debug.py:118
- ...\_wrapped\_call\_impl\_module.py:1733
- forward\_assignment\_transformer\_debug.py:175
- ...\_call\_impl\_module.py:1784
- ...\_wrapped\_call\_impl\_module.py:1773
- forward\_assignment\_transformer\_debug.py:304
- ...\_call\_impl\_module.py:1784
- ...\_wrapped\_call\_impl\_module.py:1773
- main\_assignment\_transformer\_debug.py:340
- <module>.assignment\_transformer\_debug.py:345

Sheet\_Assignment1 > assignment\_transformer\_debug.py

## Snapshot #8 — Self-attention keys (K)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["k"]`

Observed shape: (1,6,128)

K is another linear projection representing addresses/features compared against Q. Same shape ensures  $QK^\top$  is defined.

The screenshot shows the PyCharm IDE interface with the code editor open to `assignment_transformer_debug.py`. The code implements a `MultiHeadAttention` module. A specific section of the `forward` method is highlighted in blue, showing the computation of multi-head attention scores:

```
    q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - 0 multi-head split (shape: b,h,seq,d_head)
    k_heads = self._split_heads(k)
    v_heads = self._split_heads(v)

    # Scaled dot-product attention
    # pre-softmax scores (no mask yet)
    scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)
```

The variable `scores_pre_mask` is selected in the code editor. Below the editor, the `Threads & Variables` tool window is open, showing the current state of the variable:

Variable	Type	Value
<code>scores_pre_mask</code>	<code>Tensor</code>	(1, 6, 128)
<code>H</code>	<code>Tensor</code>	(1, 6, 128)
<code>T</code>	<code>Tensor</code>	(1, 6, 128)
<code>grad_fn</code>	<code>ViewBackward0</code>	<ViewBackward0 object at 0x7f305217400>
<code>grad_fn_0</code>	<code>NoneType</code>	None
<code>grad_fn_1</code>	<code>ViewBackward0</code>	<ViewBackward0 object at 0x7f305217400>
<code>grad_fn_2</code>	<code>NoneType</code>	None
<code>grad_fn_3</code>	<code>NoneType</code>	None
<code>grad_fn_4</code>	<code>NoneType</code>	None
<code>grad_fn_5</code>	<code>NoneType</code>	None
<code>grad_fn_6</code>	<code>NoneType</code>	None
<code>grad_fn_7</code>	<code>NoneType</code>	None
<code>grad_fn_8</code>	<code>NoneType</code>	None
<code>grad_fn_9</code>	<code>NoneType</code>	None
<code>grad_fn_10</code>	<code>NoneType</code>	None
<code>grad_fn_11</code>	<code>NoneType</code>	None
<code>grad_fn_12</code>	<code>NoneType</code>	None
<code>grad_fn_13</code>	<code>NoneType</code>	None
<code>grad_fn_14</code>	<code>NoneType</code>	None
<code>grad_fn_15</code>	<code>NoneType</code>	None
<code>grad_fn_16</code>	<code>NoneType</code>	None
<code>grad_fn_17</code>	<code>NoneType</code>	None
<code>grad_fn_18</code>	<code>NoneType</code>	None
<code>grad_fn_19</code>	<code>NoneType</code>	None
<code>grad_fn_20</code>	<code>NoneType</code>	None
<code>grad_fn_21</code>	<code>NoneType</code>	None
<code>grad_fn_22</code>	<code>NoneType</code>	None
<code>grad_fn_23</code>	<code>NoneType</code>	None
<code>grad_fn_24</code>	<code>NoneType</code>	None
<code>grad_fn_25</code>	<code>NoneType</code>	None
<code>grad_fn_26</code>	<code>NoneType</code>	None
<code>grad_fn_27</code>	<code>NoneType</code>	None
<code>grad_fn_28</code>	<code>NoneType</code>	None
<code>grad_fn_29</code>	<code>NoneType</code>	None
<code>grad_fn_30</code>	<code>NoneType</code>	None
<code>grad_fn_31</code>	<code>NoneType</code>	None
<code>grad_fn_32</code>	<code>NoneType</code>	None
<code>grad_fn_33</code>	<code>NoneType</code>	None
<code>grad_fn_34</code>	<code>NoneType</code>	None
<code>grad_fn_35</code>	<code>NoneType</code>	None
<code>grad_fn_36</code>	<code>NoneType</code>	None
<code>grad_fn_37</code>	<code>NoneType</code>	None
<code>grad_fn_38</code>	<code>NoneType</code>	None
<code>grad_fn_39</code>	<code>NoneType</code>	None
<code>grad_fn_40</code>	<code>NoneType</code>	None
<code>grad_fn_41</code>	<code>NoneType</code>	None
<code>grad_fn_42</code>	<code>NoneType</code>	None
<code>grad_fn_43</code>	<code>NoneType</code>	None
<code>grad_fn_44</code>	<code>NoneType</code>	None
<code>grad_fn_45</code>	<code>NoneType</code>	None
<code>grad_fn_46</code>	<code>NoneType</code>	None
<code>grad_fn_47</code>	<code>NoneType</code>	None
<code>grad_fn_48</code>	<code>NoneType</code>	None
<code>grad_fn_49</code>	<code>NoneType</code>	None
<code>grad_fn_50</code>	<code>NoneType</code>	None
<code>grad_fn_51</code>	<code>NoneType</code>	None
<code>grad_fn_52</code>	<code>NoneType</code>	None
<code>grad_fn_53</code>	<code>NoneType</code>	None
<code>grad_fn_54</code>	<code>NoneType</code>	None
<code>grad_fn_55</code>	<code>NoneType</code>	None
<code>grad_fn_56</code>	<code>NoneType</code>	None
<code>grad_fn_57</code>	<code>NoneType</code>	None
<code>grad_fn_58</code>	<code>NoneType</code>	None
<code>grad_fn_59</code>	<code>NoneType</code>	None
<code>grad_fn_60</code>	<code>NoneType</code>	None
<code>grad_fn_61</code>	<code>NoneType</code>	None
<code>grad_fn_62</code>	<code>NoneType</code>	None
<code>grad_fn_63</code>	<code>NoneType</code>	None
<code>grad_fn_64</code>	<code>NoneType</code>	None
<code>grad_fn_65</code>	<code>NoneType</code>	None
<code>grad_fn_66</code>	<code>NoneType</code>	None
<code>grad_fn_67</code>	<code>NoneType</code>	None
<code>grad_fn_68</code>	<code>NoneType</code>	None
<code>grad_fn_69</code>	<code>NoneType</code>	None
<code>grad_fn_70</code>	<code>NoneType</code>	None
<code>grad_fn_71</code>	<code>NoneType</code>	None
<code>grad_fn_72</code>	<code>NoneType</code>	None
<code>grad_fn_73</code>	<code>NoneType</code>	None
<code>grad_fn_74</code>	<code>NoneType</code>	None
<code>grad_fn_75</code>	<code>NoneType</code>	None
<code>grad_fn_76</code>	<code>NoneType</code>	None
<code>grad_fn_77</code>	<code>NoneType</code>	None
<code>grad_fn_78</code>	<code>NoneType</code>	None
<code>grad_fn_79</code>	<code>NoneType</code>	None
<code>grad_fn_80</code>	<code>NoneType</code>	None
<code>grad_fn_81</code>	<code>NoneType</code>	None
<code>grad_fn_82</code>	<code>NoneType</code>	None
<code>grad_fn_83</code>	<code>NoneType</code>	None
<code>grad_fn_84</code>	<code>NoneType</code>	None
<code>grad_fn_85</code>	<code>NoneType</code>	None
<code>grad_fn_86</code>	<code>NoneType</code>	None
<code>grad_fn_87</code>	<code>NoneType</code>	None
<code>grad_fn_88</code>	<code>NoneType</code>	None
<code>grad_fn_89</code>	<code>NoneType</code>	None
<code>grad_fn_90</code>	<code>NoneType</code>	None
<code>grad_fn_91</code>	<code>NoneType</code>	None
<code>grad_fn_92</code>	<code>NoneType</code>	None
<code>grad_fn_93</code>	<code>NoneType</code>	None
<code>grad_fn_94</code>	<code>NoneType</code>	None
<code>grad_fn_95</code>	<code>NoneType</code>	None
<code>grad_fn_96</code>	<code>NoneType</code>	None
<code>grad_fn_97</code>	<code>NoneType</code>	None
<code>grad_fn_98</code>	<code>NoneType</code>	None
<code>grad_fn_99</code>	<code>NoneType</code>	None
<code>grad_fn_100</code>	<code>NoneType</code>	None
<code>grad_fn_101</code>	<code>NoneType</code>	None
<code>grad_fn_102</code>	<code>NoneType</code>	None
<code>grad_fn_103</code>	<code>NoneType</code>	None
<code>grad_fn_104</code>	<code>NoneType</code>	None
<code>grad_fn_105</code>	<code>NoneType</code>	None
<code>grad_fn_106</code>	<code>NoneType</code>	None
<code>grad_fn_107</code>	<code>NoneType</code>	None
<code>grad_fn_108</code>	<code>NoneType</code>	None
<code>grad_fn_109</code>	<code>NoneType</code>	None
<code>grad_fn_110</code>	<code>NoneType</code>	None
<code>grad_fn_111</code>	<code>NoneType</code>	None
<code>grad_fn_112</code>	<code>NoneType</code>	None
<code>grad_fn_113</code>	<code>NoneType</code>	None
<code>grad_fn_114</code>	<code>NoneType</code>	None
<code>grad_fn_115</code>	<code>NoneType</code>	None
<code>grad_fn_116</code>	<code>NoneType</code>	None
<code>grad_fn_117</code>	<code>NoneType</code>	None
<code>grad_fn_118</code>	<code>NoneType</code>	None
<code>grad_fn_119</code>	<code>NoneType</code>	None
<code>grad_fn_120</code>	<code>NoneType</code>	None
<code>grad_fn_121</code>	<code>NoneType</code>	None
<code>grad_fn_122</code>	<code>NoneType</code>	None
<code>grad_fn_123</code>	<code>NoneType</code>	None
<code>grad_fn_124</code>	<code>NoneType</code>	None
<code>grad_fn_125</code>	<code>NoneType</code>	None
<code>grad_fn_126</code>	<code>NoneType</code>	None
<code>grad_fn_127</code>	<code>NoneType</code>	None
<code>grad_fn_128</code>	<code>NoneType</code>	None
<code>grad_fn_129</code>	<code>NoneType</code>	None
<code>grad_fn_130</code>	<code>NoneType</code>	None
<code>grad_fn_131</code>	<code>NoneType</code>	None
<code>grad_fn_132</code>	<code>NoneType</code>	None
<code>grad_fn_133</code>	<code>NoneType</code>	None
<code>grad_fn_134</code>	<code>NoneType</code>	None
<code>grad_fn_135</code>	<code>NoneType</code>	None
<code>grad_fn_136</code>	<code>NoneType</code>	None
<code>grad_fn_137</code>	<code>NoneType</code>	None
<code>grad_fn_138</code>	<code>NoneType</code>	None
<code>grad_fn_139</code>	<code>NoneType</code>	None
<code>grad_fn_140</code>	<code>NoneType</code>	None
<code>grad_fn_141</code>	<code>NoneType</code>	None
<code>grad_fn_142</code>	<code>NoneType</code>	None
<code>grad_fn_143</code>	<code>NoneType</code>	None
<code>grad_fn_144</code>	<code>NoneType</code>	None
<code>grad_fn_145</code>	<code>NoneType</code>	None
<code>grad_fn_146</code>	<code>NoneType</code>	None
<code>grad_fn_147</code>	<code>NoneType</code>	None
<code>grad_fn_148</code>	<code>NoneType</code>	None
<code>grad_fn_149</code>	<code>NoneType</code>	None
<code>grad_fn_150</code>	<code>NoneType</code>	None
<code>grad_fn_151</code>	<code>NoneType</code>	None
<code>grad_fn_152</code>	<code>NoneType</code>	None
<code>grad_fn_153</code>	<code>NoneType</code>	None
<code>grad_fn_154</code>	<code>NoneType</code>	None
<code>grad_fn_155</code>	<code>NoneType</code>	None
<code>grad_fn_156</code>	<code>NoneType</code>	None
<code>grad_fn_157</code>	<code>NoneType</code>	None
<code>grad_fn_158</code>	<code>NoneType</code>	None
<code>grad_fn_159</code>	<code>NoneType</code>	None
<code>grad_fn_160</code>	<code>NoneType</code>	None
<code>grad_fn_161</code>	<code>NoneType</code>	None
<code>grad_fn_162</code>	<code>NoneType</code>	None
<code>grad_fn_163</code>	<code>NoneType</code>	None
<code>grad_fn_164</code>	<code>NoneType</code>	None
<code>grad_fn_165</code>	<code>NoneType</code>	None
<code>grad_fn_166</code>	<code>NoneType</code>	None
<code>grad_fn_167</code>	<code>NoneType</code>	None
<code>grad_fn_168</code>	<code>NoneType</code>	None
<code>grad_fn_169</code>	<code>NoneType</code>	None
<code>grad_fn_170</code>	<code>NoneType</code>	None
<code>grad_fn_171</code>	<code>NoneType</code>	None
<code>grad_fn_172</code>	<code>NoneType</code>	None
<code>grad_fn_173</code>	<code>NoneType</code>	None
<code>grad_fn_174</code>	<code>NoneType</code>	None
<code>grad_fn_175</code>	<code>NoneType</code>	None
<code>grad_fn_176</code>	<code>NoneType</code>	None
<code>grad_fn_177</code>	<code>NoneType</code>	None
<code>grad_fn_178</code>	<code>NoneType</code>	None
<code>grad_fn_179</code>	<code>NoneType</code>	None
<code>grad_fn_180</code>	<code>NoneType</code>	None
<code>grad_fn_181</code>	<code>NoneType</code>	None
<code>grad_fn_182</code>	<code>NoneType</code>	None
<code>grad_fn_183</code>	<code>NoneType</code>	None
<code>grad_fn_184</code>	<code>NoneType</code>	None
<code>grad_fn_185</code>	<code>NoneType</code>	None
<code>grad_fn_186</code>	<code>NoneType</code>	None
<code>grad_fn_187</code>	<code>NoneType</code>	None
<code>grad_fn_188</code>	<code>NoneType</code>	None
<code>grad_fn_189</code>	<code>NoneType</code>	None
<code>grad_fn_190</code>	<code>NoneType</code>	None
<code>grad_fn_191</code>	<code>NoneType</code>	None
<code>grad_fn_192</code>	<code>NoneType</code>	None
<code>grad_fn_193</code>	<code>NoneType</code>	None
<code>grad_fn_194</code>	<code>NoneType</code>	None
<code>grad_fn_195</code>	<code>NoneType</code>	None
<code>grad_fn_196</code>	<code>NoneType</code>	None
<code>grad_fn_197</code>	<code>NoneType</code>	None
<code>grad_fn_198</code>	<code>NoneType</code>	None
<code>grad_fn_199</code>	<code>NoneType</code>	None
<code>grad_fn_200</code>	<code>NoneType</code>	None
<code>grad_fn_201</code>	<code>NoneType</code>	None
<code>grad_fn_202</code>	<code>NoneType</code>	None
<code>grad_fn_203</code>	<code>NoneType</code>	None
<code>grad_fn_204</code>	<code>NoneType</code>	None
<code>grad_fn_205</code>	<code>NoneType</code>	None
<code>grad_fn_206</code>	<code>NoneType</code>	None
<code>grad_fn_207</code>	<code>NoneType</code>	None
<code>grad_fn_208</code>	<code>NoneType</code>	None
<code>grad_fn_209</code>	<code>NoneType</code>	None
<code>grad_fn_210</code>	<code>NoneType</code>	None
<code>grad_fn_211</code>	<code>NoneType</code>	None
<code>grad_fn_212</code>	<code>NoneType</code>	None
<code>grad_fn_213</code>	<code>NoneType</code>	None
<code>grad_fn_214</code>	<code>NoneType</code>	None
<code>grad_fn_215</code>	<code>NoneType</code>	None
<code>grad_fn_216</code>	<code>NoneType</code>	None
<code>grad_fn_217</code>	<code>NoneType</code>	None
<code>grad_fn_218</code>	<code>NoneType</code>	None
<code>grad_fn_219</code>	<code>NoneType</code>	None
<code>grad_fn_220</code>	<code>NoneType</code>	None
<code>grad_fn_221</code>	<code>NoneType</code>	None
<code>grad_fn_222</code>	<code>NoneType</code>	None
<code>grad_fn_223</code>	<code>NoneType</code>	None
<code>grad_fn_224</code>	<code>NoneType</code>	None
<code>grad_fn_225</code>	<code>NoneType</code>	None
<code>grad_fn_226</code>	<code>NoneType</code>	None
<code>grad_fn_227</code>	<code>NoneType</code>	None
<code>grad_fn_228</code>	<code>NoneType</code>	None
<code>grad_fn_229</code>	<code>NoneType</code>	None
<code>grad_fn_230</code>	<code>NoneType</code>	None
<code>grad_fn_231</code>	<code>NoneType</code>	None
<code>grad_fn_232</code>	<code>NoneType</code>	None
<code>grad_fn_233</code>	<code>NoneType</code>	None
<code>grad_fn_234</code>	<code>NoneType</code>	None
<code>grad_fn_235</code>	<code>NoneType</code>	None
<code>grad_fn_236</code>	<code>NoneType</code>	None
<code>grad_fn_237</code>	<code>NoneType</code>	None
<code>grad_fn_238</code>	<code>NoneType</code>	None
<code>grad_fn_239</code>	<code>NoneType</code>	None
<code>grad_fn_240</code>	<code>NoneType</code>	None
<code>grad_fn_241</code>	<code>NoneType</code>	None
<code>grad_fn_242</code>	<code>NoneType</code>	None
<code>grad_fn_243</code>	<code>NoneType</code>	None
<code>grad_fn_244</code>	<code>NoneType</code>	None
<code>grad_fn_245</code>	<code>NoneType</code>	None
<code>grad_fn_246</code>	<code>NoneType</code>	None
<code>grad_fn_247</code>	<code>NoneType</code>	None
<code>grad_fn_248</code>	<code>NoneType</code>	None
<code>grad_fn_249</code>	<code>NoneType</code>	None
<code>grad_fn_250</code>	<code>NoneType</code>	None
<code>grad_fn_251</code>	<code>NoneType</code>	None
<code>grad_fn_252</code>	<code>NoneType</code>	None
<code>grad_fn_253</code>	<code>NoneType</code>	None
<code>grad_fn_254</code>	<code>NoneType</code>	None
<code>grad_fn_255</code>	<code>NoneType</code>	None
<code>grad_fn_256</code>	<code>NoneType</code>	None
<code>grad_fn_257</code>	<code>NoneType</code>	None
<code>grad_fn_258</code>	<code>NoneType</code>	None
<code>grad_fn_259</code>	<code>NoneType</code>	None
<code>grad_fn_260</code>	<code>NoneType</code>	None
<code>grad_fn_261</code>	<code>NoneType</code>	None
<code>grad_fn_262</code>	<code>NoneType</code>	None
<code>grad_fn_263</code>	<code>NoneType</code>	None
<code>grad_fn_264</code>	<code>NoneType</code>	None
<code>grad_fn_265</code>	<code>NoneType</code>	None
<code>grad_fn_266</code>	<code>NoneType</code>	None
<code>grad_fn_267</code>	<code>NoneType</code>	None
<code>grad_fn_268</code>	<code>NoneType</code>	None
<code>grad_fn_269</code>	<code>NoneType</code>	None
<code>grad_fn_270</code>	<code>NoneType</code>	None
<code>grad_fn_271</code>	<code>NoneType</code>	None
<code>grad_fn_272</code>	<code>NoneType</code>	None
<code>grad_fn_273</code>	<code>NoneType</code>	None
<code>grad_fn_274</code>	<code>NoneType</code>	None
<code>grad_fn_275</code>	<code>NoneType</code>	None
<code>grad_fn_276</code>	<code>NoneType</code>	None
<code>grad_fn_277</code>	<code>NoneType</code>	None
<code>grad_fn_278</code>	<code>NoneType</code>	None
<code>grad_fn_279</code>	<code>NoneType</code>	None
<code>grad_fn_280</code>	<code>NoneType</code>	None
<code>grad_fn_281</code>	<code>NoneType</code>	None
<code>grad_fn_282</code>	<code>NoneType</code>	None
<code>grad_fn_283</code>	<code>NoneType</code>	None
<code>grad_fn_284</code>	<code>NoneType</code>	None
<code>grad_fn_285</code>	<code>NoneType</code>	None
<code>grad_fn_286</code>	<code>NoneType</code>	None
<code>grad_fn_287</code>	<code>NoneType</code>	None
<code>grad_fn_288</code>	<code>NoneType</code>	None
<code>grad_fn_289</code>	<code>NoneType</code>	None
<code>grad_fn_290</code>	<code>NoneType</code>	None
<code>grad_fn_291</code>	<code>NoneType</code>	None
<code>grad_fn_292</code>	<code>NoneType</code>	None
<code>grad_fn_293</code>	<code>NoneType</code>	None

The screenshot shows a Jupyter Notebook interface with two open cells. The top cell contains Python code for a `MultiHeadAttention` class, which performs multi-head attention operations on input tensors `q`, `k`, and `v`. It includes forward passes, scaling dot-product attention, applying pre-softmax scores as a mask, and calculating final attention weights, output heads, and concatenated results. The bottom cell is titled "assignment\_transformer\_debug" and contains a single line of code: `forward, assignment_transformer_debug.py:129`.

```
97     class MultiHeadAttention(nn.Module): 3 usages
114         def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  k_in: tensor([[[-5.0110e-01, 9.9553e-01
115             q_heads = self._split_heads(q) # SNAPSHOT #2 on #27 - Q multi-head split (shape: b,h,seq,d_head)  q_heads: tensor([[[[ 4.8122e-01, 4.0718e-01, 2.9
120             k_heads = self._split_heads(k)  k_heads: tensor([[[[-9.2225e-01, -3.9407e-01, 1.3553e-01, -9.5973e-02, 6.4861e-02, \n
121             v_heads = self._split_heads(v)  v_heads: tensor([[[[ 6.4838e-01, 8.0307e-01, 1.2474e+00, 1.2914e-01, -9.2871e-01, \n
122
123             # Scaled dot-product attention
124             # pre-softmax scores (no mask yet)
125             scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)  scores_pre_mask: tensor([[[[ 0.1813, -0.16
126             # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
127
128
129             scores = scores_pre_mask.clone()
130             if mask is not None:
131                 # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
132                 scores = scores + mask # SNAPSHOT #25 - Mask tensor gets added here
133
134             attn_weights = F.softmax(scores, dim=-1) # SNAPSHOT #11 or #26 or #34 - After softmax
135             attn_out_heads = attn_weights @ v_heads # (b,h,tgt_len,d_head)
136             attn_concat = attn_out_heads.transpose(dim0=1, dim1=2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)
137             # SNAPSHOT #13 or #28 or #35 - Multi-head output after concatenation
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
427
428
429
429
430
431
432
433
433
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
```

## Snapshot #10 — Attention scores before softmax

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["scores_pre_mask"]`  
Observed shape: (1,4,6,6)

Per-head scaled dot products  $QK^\top / \sqrt{d_{\text{head}}}$ . Square  $6 \times 6$  because encoder self-attends over the same sequence.

### Snapshot #11 — Attention scores after softmax

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["attn_weights"]`  
Observed shape: `(1,4,6,6)`

Row-wise probabilities per head; each row sums to  $\approx 1$ . These weights mix V.

The screenshot shows the PyCharm IDE interface. The top bar displays the project name "Sheet\_Assignment" and the file "assignment\_transformer\_debug.py". The main area is a code editor with syntax highlighting and line numbers. A red box highlights the following code block:

```
    class MultiheadAttention(nn.Module): 3 usages
  97     def __init__(self, d_model):
  98         super().__init__()
  99         self.d_model = d_model
 100        self.nhead = nhead
 101        self.d_head = d_head
 102        self.linear_in = nn.Linear(d_model, nhead * d_head)
 103        self.linear_out = nn.Linear(nhead * d_head, d_model)
 104        self.linear_q = nn.Linear(d_model, nhead * d_head)
 105        self.linear_k = nn.Linear(d_model, nhead * d_head)
 106        self.linear_v = nn.Linear(d_model, nhead * d_head)
 107
 108        self.dropout = nn.Dropout(p=dropout)
 109
 110        self.scale_factor = d_head ** -0.5
 111
 112        self.positionwise_feedforward = nn.Sequential(
 113            nn.Linear(d_model, 4 * d_model),
 114            nn.ReLU(),
 115            nn.Linear(4 * d_model, d_model),
 116            nn.Dropout(p=dropout)
 117        )
 118
 119        self.layer_norm = nn.LayerNorm(d_model)
 120
 121        self._reset_parameters()
 122
 123    def forward(self, x, mask=None):
 124        b, s, _ = x.size()
 125        x = x.view(b, s, self.nhead, self.d_head).transpose(0, 1) # (batch, heads, seq, d_head)
 126        return x
 127
 128    def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  mask: None  k_in: tensor([[ 5.0110e-01,  9.9553e-01,
 129      q_in: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
 130      k_in: tensor([[ 9.2225e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,  4.7361e-01,  1.0521e+00, -3.7656e-01,
 131      v_in: tensor([[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01, -2.6742e-01,  6.1655e-01,  1.1496e+00,
 132
 133      q_heads = self._split_heads(q)  # SNAPSHOT #12 or #37 - 0 multi-head split (shape: b, h, seq, d_head)  q_heads: tensor([[[[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,
 134      k_heads = self._split_heads(k)  k_heads: tensor([[[[ 9.2225e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,  4.7361e-01,  1.0521e+00, -3.7656e-01,
 135      v_heads = self._split_heads(v)  v_heads: tensor([[[[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01, -2.6742e-01,  6.1655e-01,  1.1496e+00,
 136
 137      # Scaled dot-product attention
 138      # pre-softmax scores (no mask yet)
 139      scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head)  # (b, h, tgt_len, src_len)
 140
 141      # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
 142
 143      scores = scores_pre_mask.clone()
 144
 145      scores = scores * self.scale_factor
 146      scores = scores + self.positionwise_feedforward(x)
 147      scores = self.layer_norm(scores)
 148
 149      if mask is not None:
 150          scores = scores.masked_fill(mask == 0, float("-inf"))
 151
 152      scores = F.softmax(scores, dim=-1)
 153
 154      x = scores @ v_heads
 155      x = x.transpose(0, 1).contiguous().view(b, s, self.d_model)
 156
 157      x = self.linear_out(x)
 158
 159      x = self.dropout(x)
 160
 161      x = self.layer_norm(x)
 162
 163      return x
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1217
 1218
 1219
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1297
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1317
 1318
 1319
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1327
 1328
 1329
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1389
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1417
 1418
 1419
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1427
 1428
 1429
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1477
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1489
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1497
 1498
 1499
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1517
 1518
 1519
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1527
 1528
 1529
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1568
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1577
 1578
 1579
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1589
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1597
 1598
 1599
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1617
 1618
 1619
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1627
 1628
 1629
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1658
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1668
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1677
 1678
 1679
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1689
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1697
 1698
 1699
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1717
 1718
 1719
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1727
 1728
 1729
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1738
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1758
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1768
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1777
 1778
 1779
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1789
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1797
 1798
 1799
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1809
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1817
 1818
 1819
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1827
 1828
 1829
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1838
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1848
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1858
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1868
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1877
 1878
 1879
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1889
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1897
 1898
 1899
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1909
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1917
 1918
 1919
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1927
 1928
 1929
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1938
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1948
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 195
```

Snapshot #12 — Multi-head split (Q/K/V)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["q_heads"]` (& `k_heads`, `v_heads`)

**Observed shape:** (1,4,6,32) each

Split  $d_{\text{model}} = 128$  into 4 heads of 32 to capture diverse relations in parallel.

The screenshot shows the PyCharm debugger interface. The top window displays the `MultiHeadAttention` class definition with several code annotations. The bottom window shows the `forward` method and the variable `attn_out_heads` in the watch list.

```

97     class MultiHeadAttention(nn.Module): 3 usages
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):
122         v_heads = self._split_heads(v)  v_heads: tensor([[[[ 6.4638e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01, -2.6742e-01,  6.1655e-01,  1.1495e+00,
123
124         # Scaled dot-product attention
125         # pre-softmax scores (no mask yet)
126         scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head)  # (b,h,tgt_len,src_len)  scores_pre_mask: tensor([[[[ 0.1813, -0.1693, -0.1295,  0.0085, -0.4855, -0.1307], ... [-0.2066, -0.0625, -0.1083,  0.1918,  0.2571, ...
127         # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
128
129         scores = scores_pre_mask.clone()  scores: tensor([[[[ 0.1813, -0.1693, -0.1295,  0.0085, -0.4855, -0.1307], ...
130         if mask is not None:
131             # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
132             scores = scores + mask  # SNAPSHOT #25 - Mask tensor gets added here
133
134         attn_weights = F.softmax(scores, dim=-1)  # SNAPSHOT #11 or #26 or #34 - After softmax attn weights: tensor([[[[ 0.2210,  0.1557,  0.1620,  0.1860,  0.1135,  0.1618], ...
135         attn_out_heads = attn_weights @ v_heads  # (b,h,tgt_len,d_head)  attn_out_heads: tensor([[[[ 1.5819e-01,  2.3866e-01, -2.8876e-01, -6.7833e-01, ...
136         attn_concat = attn_out_heads.transpose(dim0=1, dim1=2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)  attn_concat: tensor([[[[ 1.5819e-01,  2.3866e-01, ...
137         # SNAPSHOT #13 or #28 or #35 - Multi-head output after concatenation
138
139         out = self.W_o(attn_concat)
140
141         return out, q, k, v, q_heads, k_heads, v_heads, scores_pre_mask, attn_weights, attn_concat
142
143     Debug assignment_transformer_debug.x
144
145     MainThread
146     Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
147     forward_assignment_transformer_debug.py:139
148     output, nr = (int) 0
149     > real = (Tensor: (1, 6, 128)) tensor([[1.5819e-01, 2.3866e-01, 1.7305e-01, -2.8876e-01, -6.7833e-01, ...
150     > requires_grad = (bool) True
151     > retains_grad = (bool) False
152     > shape = (Size: torch.Size((1, 6, 128))
153     > Protected Attributes
154     > attn_out_heads = (Tensor: (1, 4, 6, 32)) tensor([[1.5819e-01, 2.3866e-01, 1.7305e-01, -2.8876e-01, -6.7833e-01, ...
155     > attn_weights = (Tensor: (1, 4, 6, 6)) tensor([[1.010e-01, 0.907e-01, 0.897e-01, 0.887e-01, 0.877e-01, 0.867e-01, ...
156     > k_in = (Tensor: (1, 6, 128)) tensor([[9.2225e-01, -3.8407e-01, 1.3559e-01, -3.5973e-02, 6.4046e-02, ...
157     > k_out = (Tensor: (1, 4, 6, 32)) tensor([[9.2225e-01, -3.8407e-01, 1.3559e-01, -3.5973e-02, 6.4046e-02, ...
158     > k_in = (Tensor: (1, 6, 128)) tensor([[5.010e-01, 0.8553e-01, 1.0681e+00, -7.2736e-01, -1.4915e+00, ...
159     > mask = (NoneType) None

```

## Snapshot #13 — Multi-head attention output after concatenation

### Explanation & Shape

**Variable (as shown in PyCharm):** `refs["enc_traces"][0]["attn_concat"]`

**Observed shape:** (1,6,128)

Per-head outputs (1,4,6,32) are concatenated to (1,6,128), restoring model dimension for residual addition.

Snapshot #14 — Residual connection tensors

## Explanation & Shape

Variable (as shown in PyCharm): refs["enc\_traces"][0]["res1\_a"],  
refs["enc\_traces"][0]["res1\_b"]

**Observed shape:** (1,6,128) each

Residual adds attention output to the input tensor element-wise; identical shapes ensure valid addition and stable gradients.

The screenshot shows a PyCharm interface with multiple tabs open. The main editor tab contains Python code for a `TinyTransformer` class. A debugger window is open at the bottom, showing the stack trace and memory dump. The memory dump shows various tensors and their values, such as `src_ids`, `tgt_ids`, `src_embed`, `tgt_with_pos`, and `logits`. The memory dump also includes a large array of floating-point numbers representing the `enc_traces` tensor.

```
265     class TinyTransformer(nn.Module): 1 usage
266         def forward(self, src_ids, tgt_ids):  self: TinyTransformer(\n            (src_emb): Embedding(6, 128)\n            (tgt_emb): Embedding(5, 128)\n            (pos): PositionalEncoding(\n                (proj): Linear(128, 128)\n            )\n        )\n        logits = self.proj(dec_final) # SNAPSHOT #42 - Logits after final Linear projection\n        logits_slice = logits[0, 0, :10] # SNAPSHOT #43 - Logits slice (first few values for one token)\n\n320\n321\n322\n323\n324\n325\n326\n327\n328\n329\n330\n331\n332\n333\n334\n335\n\n    # Useful references for debugger (do not print; inspect in PyCharm)\n    snapshot_refs = dict(\n        # 1 & 2 captured above (src_ids, tgt_ids)\n        emb_weight_slice=emb_weight_slice, # ##5\n        src_embed=src_embed, # ##4\n        src_with_pos=src_with_pos, # ##5\n        enc_traces=enc_traces, # ##6..#19 inside enc_traces[0]\n        tgt_with_pos=tgt_with_pos, # ##20\n        tgt_mask=tgt_mask, # ##25\n        dec_traces=dec_traces, # ##21..#40 inside dec_traces[0]\n        dec_final=dec_final, # ##41\n        logits=logits, # ##42\n        logots_slice=logots_slice # ##43\n    )\n\n    return logits, snapshot_refs\n
```

## Snapshot #15 — Layer normalization output

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["ln1_out"]`  
Observed shape: (1,6,128)

LayerNorm re-scales the residual sum per feature without changing shape, stabilizing training.

## Snapshot #16 — Feed-forward input

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["ff_in"]`

Observed shape: (1,6,128)

Normalized token states feed the position-wise FFN;  $d_{\text{model}} = 128$  is preserved for residuals.

The screenshot shows the PyCharm IDE interface. On the left, the code editor displays the `assignment_transformer_debug.py` file, specifically the `PositionwiseFFN` class. The code defines two linear layers (`fc1` and `fc2`) with  $d_{\text{model}} = 128$  and  $d_{\text{ff}} = 256$ . The `forward` method takes a tensor `x` and applies these layers sequentially, followed by a `relu` activation. On the right, the debugger window is open, showing the variable `ff_in` at snapshot #17. The variable is a tensor of shape (1, 6, 128), containing normalized token states. The debugger also shows other variables like `real`, `fc1`, and `fc2` with their respective shapes and values.

Snapshot #17 — Feed-forward first linear layer output

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["ff1"]`

Observed shape: (1,6,256)

FFN expands to  $d_{\text{ff}} = 256$ , increasing per-token capacity before projecting back.

Snapshot #18 — Feed-forward second linear layer output

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["ff2"]`  
Observed shape: (1,6,128)

Projects back to 128 so it can be added (residual) to the FFN input.

## Snapshot #19 — Encoder block final output tensor

## Explanation & Shape

Variable (as shown in PyCharm): `refs["enc_traces"][0]["enc_out"]`  
Observed shape: (1,6,128)

After FFN residual + LayerNorm, this output feeds later encoder layers and decoder cross-attention.

Decoder (One Block, Full Trace) — Snapshots 20–40

The screenshot shows the PyCharm IDE interface with two tabs open: `assignment_transformer_debug.py` and `module.py`. The `assignment_transformer_debug.py` tab contains the following code:

```
265
266     class TinyTransformer(nn.Module): 1 usage
267
268     def forward(self, src_ids, tgt_ids): self: TinyTransformer(\n        (src_emb): Embedding(6, 128)\n        (tgt_emb): Embedding(5, 128)\n        (pos): PositionalEncoding(\n            (logits = self.proj(dec_final) # SNAPSHOT #42 - Logits after final linear projection\n            logits_slice = logits[0, 0, :10] # SNAPSHOT #43 - Logits slice (first few values for one token)\n            logits_slice: tensor([-0.1160, 0.8900, -0.3211, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000])\n\n320\n321         # Useful references for debugger (do not print; inspect in PyCharm)\n322         snapshot_refs = dict(\n323             # 1 & 2 captured above (src_ids, tgt_ids)\n324             emb_weight_slice=emb_weight_slice, # #3\n325             src_embed=src_embed, # #4\n326             src_with_pos=src_with_pos, # #5\n327             enc_traces=enc_traces, # #6..#19 inside enc_traces[0]\n328             tgt_with_pos=tgt_with_pos, # #20\n329             tgt_mask=tgt_mask, # #25\n330             dec_traces=dec_traces, # #21..#40 inside dec_traces[0]\n331             dec_final=dec_final, # #41\n332             logits=logits, # #42\n333             logits_slice=logits_slice # #43\n334         )\n335\n336     return logits, snapshot_refs
```

The code is annotated with several `# #` comments indicating specific memory snapshots. The `logits` variable is annotated with `# SNAPSHOT #42 - Logits after final linear projection` and `# SNAPSHOT #43 - Logits slice (first few values for one token)`. The `logits_slice` variable is annotated with its tensor value: `tensor([-0.1160, 0.8900, -0.3211, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000])`.

## Snapshot #20 — Decoder block input tensor

## Explanation & Shape

Variable (as shown in PyCharm): refs["tgt\_with\_pos"]  
Observed shape: (1,5,128)

Target embeddings + positional encodings (teacher forcing). Target sequence length is 5.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109         # x: (batch, seq, d_model)
110         b, s, _ = x.size()
111         x = x.view(b, s, self.nhead, self.d_head).transpose(dim0=1, dim1=2) # (batch, heads, seq, d_head)
112         return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None): k_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00,
115         # q_in, k_in, v_in: (batch, seq, d_model)
116         q = self.W.q(q_in) # SNAPSHOT #7 or #21 or #30 - Q (depending on caller) q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
117         k = self.W.k(k_in) # SNAPSHOT #8 or #22 or #31 - K: tensor([[ 9.2256e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,
118         v = self.W.v(v_in) # SNAPSHOT #9 or #23 or #32 - V
119
120         q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)
121         k_heads = self._split_heads(k)
122         v_heads = self._split_heads(v)
123
124         # Scaled dot-product attention
125         # pre-softmax scores (no mask yet)
126         scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)
127         # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
128
129         scores = scores_pre_mask.clone()

```

The screenshot shows the PyCharm debugger interface with the code editor and the debugger tool window. In the debugger, the variable `q` is selected, and its value is displayed in the console. The variable `q` has a shape of `(1, 5, 128)`.

### Snapshot #21 — Masked self-attention queries (Q)

#### Explanation & Shape

**Variable (as shown in PyCharm):** `refs["dec_traces"][0]["q_m"]`  
**Observed shape:**  $(1, 5, 128)$

Decoder Q is a linear projection of decoder inputs; same shape as K,V to enable attention.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109         # x: (batch, seq, d_model)
110         b, s, _ = x.size()
111         x = x.view(b, s, self.nhead, self.d_head).transpose(dim0=1, dim1=2) # (batch, heads, seq, d_head)
112         return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None): k_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00,
115         # q_in, k_in, v_in: (batch, seq, d_model)
116         q = self.W.q(q_in) # SNAPSHOT #7 or #21 or #30 - Q (depending on caller) q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
117         k = self.W.k(k_in) # SNAPSHOT #8 or #22 or #31 - K: tensor([[ 9.2256e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,
118         v = self.W.v(v_in) # SNAPSHOT #9 or #23 or #32 - V
119
120         q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)
121         k_heads = self._split_heads(k)
122         v_heads = self._split_heads(v)
123
124         # Scaled dot-product attention
125         # pre-softmax scores (no mask yet)
126         scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)
127         # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
128
129         scores = scores_pre_mask.clone()

```

The screenshot shows the PyCharm debugger interface with the code editor and the debugger tool window. In the debugger, the variable `k` is selected, and its value is displayed in the console. The variable `k` has a shape of `(1, 5, 128)`.

### Snapshot #22 — Masked self-attention keys (K)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["k_m"]`  
Observed shape: (1,5,128)

Decoder K shares the same shape so  $QK^\top$  is valid across time steps.

The screenshot shows the PyCharm IDE interface with the following details:

- File:** assignment\_transformer\_debug.py
- Toolbars:** Version control, assignment\_transformer\_debug, and a zoomed-in status bar.
- Code Area:** The code is annotated with several colored highlights:
  - Yellow:** Lines 97-113, 118-123, 128-133, 144-150, 155-161, 166-172, 177-183, 188-194, 199-205, 210-216, 221-227, 232-238, 243-249, 254-260, 265-271, 276-282, 287-293, 298-304, 309-315, 320-326, 331-337, 342-348, 353-359, 364-370, 375-381, 386-392, 397-403, 408-414, 419-425, 426-432, 433-439, 440-446, 447-453, 454-460, 461-467, 468-474, 475-481, 482-488, 489-495, 496-502, 503-509, 510-516, 517-523, 524-530, 531-537, 538-544, 545-551, 552-558, 559-565, 566-572, 573-579, 580-586, 587-593, 594-596, 597-599, 600-602, 603-605, 606-608, 609-611, 612-614, 615-617, 618-620, 621-623, 624-626, 627-629, 630-632, 633-635, 636-638, 639-641, 642-644, 645-647, 648-650, 651-653, 654-656, 657-659, 660-662, 663-665, 666-668, 669-671, 672-674, 675-677, 678-679, 680-681, 682-683, 684-685, 686-687, 688-689, 690-691, 692-693, 694-695, 696-697, 698-699, 699-700, 701-702, 703-704, 705-706, 707-708, 709-710, 711-712, 713-714, 715-716, 717-718, 719-720, 721-722, 723-724, 725-726, 727-728, 729-730, 731-732, 733-734, 735-736, 737-738, 739-740, 741-742, 743-744, 745-746, 747-748, 749-750, 751-752, 753-754, 755-756, 757-758, 759-760, 761-762, 763-764, 765-766, 767-768, 769-770, 771-772, 773-774, 775-776, 777-778, 779-780, 781-782, 783-784, 785-786, 787-788, 789-790, 791-792, 793-794, 795-796, 797-798, 799-800, 801-802, 803-804, 805-806, 807-808, 809-810, 811-812, 813-814, 815-816, 817-818, 819-819, 820-821, 822-823, 824-825, 826-827, 828-829, 830-831, 832-833, 834-835, 836-837, 838-839, 839-840, 841-842, 843-844, 845-846, 847-848, 849-849, 850-851, 852-853, 854-855, 856-857, 858-859, 859-860, 861-862, 863-864, 865-866, 867-868, 869-869, 870-871, 872-873, 873-874, 875-876, 877-878, 878-879, 879-880, 880-881, 881-882, 882-883, 883-884, 884-885, 885-886, 886-887, 887-888, 888-889, 889-889, 890-891, 891-892, 892-893, 893-894, 894-895, 895-896, 896-897, 897-898, 898-899, 899-899, 900-901, 901-902, 902-903, 903-904, 904-905, 905-906, 906-907, 907-908, 908-909, 909-910, 910-911, 911-912, 912-913, 913-914, 914-915, 915-916, 916-917, 917-918, 918-919, 919-920, 920-921, 921-922, 922-923, 923-924, 924-925, 925-926, 926-927, 927-928, 928-929, 929-930, 930-931, 931-932, 932-933, 933-934, 934-935, 935-936, 936-937, 937-938, 938-939, 939-940, 940-941, 941-942, 942-943, 943-944, 944-945, 945-946, 946-947, 947-948, 948-949, 949-950, 950-951, 951-952, 952-953, 953-954, 954-955, 955-956, 956-957, 957-958, 958-959, 959-960, 960-961, 961-962, 962-963, 963-964, 964-965, 965-966, 966-967, 967-968, 968-969, 969-970, 970-971, 971-972, 972-973, 973-974, 974-975, 975-976, 976-977, 977-978, 978-979, 979-980, 980-981, 981-982, 982-983, 983-984, 984-985, 985-986, 986-987, 987-988, 988-989, 989-990, 990-991, 991-992, 992-993, 993-994, 994-995, 995-996, 996-997, 997-998, 998-999, 999-999, 1000-1001, 1001-1002, 1002-1003, 1003-1004, 1004-1005, 1005-1006, 1006-1007, 1007-1008, 1008-1009, 1009-1010, 1010-1011, 1011-1012, 1012-1013, 1013-1014, 1014-1015, 1015-1016, 1016-1017, 1017-1018, 1018-1019, 1019-1020, 1020-1021, 1021-1022, 1022-1023, 1023-1024, 1024-1025, 1025-1026, 1026-1027, 1027-1028, 1028-1029, 1029-1030, 1030-1031, 1031-1032, 1032-1033, 1033-1034, 1034-1035, 1035-1036, 1036-1037, 1037-1038, 1038-1039, 1039-1040, 1040-1041, 1041-1042, 1042-1043, 1043-1044, 1044-1045, 1045-1046, 1046-1047, 1047-1048, 1048-1049, 1049-1050, 1050-1051, 1051-1052, 1052-1053, 1053-1054, 1054-1055, 1055-1056, 1056-1057, 1057-1058, 1058-1059, 1059-1060, 1060-1061, 1061-1062, 1062-1063, 1063-1064, 1064-1065, 1065-1066, 1066-1067, 1067-1068, 1068-1069, 1069-1070, 1070-1071, 1071-1072, 1072-1073, 1073-1074, 1074-1075, 1075-1076, 1076-1077, 1077-1078, 1078-1079, 1079-1080, 1080-1081, 1081-1082, 1082-1083, 1083-1084, 1084-1085, 1085-1086, 1086-1087, 1087-1088, 1088-1089, 1089-1090, 1090-1091, 1091-1092, 1092-1093, 1093-1094, 1094-1095, 1095-1096, 1096-1097, 1097-1098, 1098-1099, 1099-1099, 1100-1101, 1101-1102, 1102-1103, 1103-1104, 1104-1105, 1105-1106, 1106-1107, 1107-1108, 1108-1109, 1109-1110, 1110-1111, 1111-1112, 1112-1113, 1113-1114, 1114-1115, 1115-1116, 1116-1117, 1117-1118, 1118-1119, 1119-11110, 11110-11111, 11111-11112, 11112-11113, 11113-11114, 11114-11115, 11115-11116, 11116-11117, 11117-11118, 11118-11119, 11119-111110, 111110-111111, 111111-111112, 111112-111113, 111113-111114, 111114-111115, 111115-111116, 111116-111117, 111117-111118, 111118-111119, 111119-1111110, 1111110-1111111, 1111111-1111112, 1111112-1111113, 1111113-1111114, 1111114-1111115, 1111115-1111116, 1111116-1111117, 1111117-1111118, 1111118-1111119, 1111119-11111110, 11111110-11111111, 11111111-11111112, 11111112-11111113, 11111113-11111114, 11111114-11111115, 11111115-11111116, 11111116-11111117, 11111117-11111118, 11111118-11111119, 11111119-111111110, 111111110-111111111, 111111111-111111112, 111111112-111111113, 111111113-111111114, 111111114-111111115, 111111115-111111116, 111111116-111111117, 111111117-111111118, 111111118-111111119, 111111119-1111111110, 1111111110-1111111111, 1111111111-1111111112, 1111111112-1111111113, 1111111113-1111111114, 1111111114-1111111115, 1111111115-1111111116, 1111111116-1111111117, 1111111117-1111111118, 1111111118-1111111119, 1111111119-11111111110, 11111111110-11111111111, 11111111111-11111111112, 11111111112-11111111113, 11111111113-11111111114, 11111111114-11111111115, 11111111115-11111111116, 11111111116-11111111117, 11111111117-11111111118, 11111111118-11111111119, 11111111119-111111111110, 111111111110-111111111111, 111111111111-111111111112, 111111111112-111111111113, 111111111113-111111111114, 111111111114-111111111115, 111111111115-111111111116, 111111111116-111111111117, 111111111117-111111111118, 111111111118-111111111119, 111111111119-1111111111110, 1111111111110-1111111111111, 1111111111111-1111111111112, 1111111111112-1111111111113, 1111111111113-1111111111114, 1111111111114-1111111111115, 1111111111115-1111111111116, 1111111111116-1111111111117, 1111111111117-1111111111118, 1111111111118-1111111111119, 1111111111119-11111111111110, 11111111111110-11111111111111, 11111111111111-11111111111112, 11111111111112-11111111111113, 11111111111113-11111111111114, 11111111111114-11111111111115, 11111111111115-11111111111116, 11111111111116-11111111111117, 11111111111117-11111111111118, 11111111111118-11111111111119, 11111111111119-111111111111110, 111111111111110-111111111111111, 111111111111111-111111111111112, 111111111111112-111111111111113, 111111111111113-111111111111114, 111111111111114-111111111111115, 111111111111115-111111111111116, 111111111111116-111111111111117, 111111111111117-111111111111118, 111111111111118-111111111111119, 111111111111119-1111111111111110, 1111111111111110-1111111111111111, 1111111111111111-1111111111111112, 1111111111111112-1111111111111113, 1111111111111113-1111111111111114, 1111111111111114-1111111111111115, 1111111111111115-1111111111111116, 1111111111111116-1111111111111117, 1111111111111117-1111111111111118, 1111111111111118-1111111111111119, 1111111111111119-11111111111111110, 11111111111111110-11111111111111111, 11111111111111111-11111111111111112, 11111111111111112-11111111111111113, 11111111111111113-11111111111111114, 11111111111111114-11111111111111115, 11111111111111115-11111111111111116, 11111111111111116-11111111111111117, 11111111111111117-11111111111111118, 11111111111111118-11111111111111119, 11111111111111119-111111111111111110, 111111111111111110-111111111111111111, 111111111111111111-111111111111111112, 111111111111111112-111111111111111113, 111111111111111113-111111111111111114, 111111111111111114-111111111111111115, 111111111111111115-111111111111111116, 111111111111111116-111111111111111117, 111111111111111117-111111111111111118, 111111111111111118-111111111111111119, 111111111111111119-1111111111111111110, 1111111111111111110-1111111111111111111, 1111111111111111111-1111111111111111112, 1111111111111111112-1111111111111111113, 1111111111111111113-1111111111111111114, 1111111111111111114-1111111111111111115, 1111111111111111115-1111111111111111116, 1111111111111111116-1111111111111111117, 1111111111111111117-1111111111111111118, 1111111111111111118-1111111111111111119, 1111111111111111119-11111111111111111110, 11111111111111111110-11111111111111111111, 11111111111111111111-11111111111111111112, 11111111111111111112-11111111111111111113, 11111111111111111113-11111111111111111114, 11111111111111111114-11111111111111111115, 11111111111111111115-11111111111111111116, 11111111111111111116-11111111111111111117, 11111111111111111117-11111111111111111118, 11111111111111111118-11111111111111111119, 11111111111111111119-111111111111111111110, 111111111111111111110-111111111111111111111, 111111111111111111111-111111111111111111112, 111111111111111111112-111111111111111111113, 111111111111111111113-111111111111111111114, 111111111111111111114-111111111111111111115, 111111111111111111115-111111111111111111116, 111111111111111111116-111111111111111111117, 111111111111111111117-111111111111111111118, 111111111111111111118-111111111111111111119, 111111111111111111119-1111111111111111111110, 1111111111111111111110-1111111111111111111111, 1111111111111111111111-1111111111111111111112, 1111111111111111111112-1111111111111111111113, 1111111111111111111113-1111111111111111111114, 1111111111111111111114-1111111111111111111115, 1111111111111111111115-1111111111111111111116, 1111111111111111111116-1111111111111111111117, 1111111111111111111117-1111111111111111111118, 1111111111111111111118-1111111111111111111119, 1111111111111111111119-11111111111111111111110, 11111111111111111111110-11111111111111111111111, 11111111111111111111111-11111111111111111111112, 11111111111111111111112-11111111111111111111113, 11111111111111111111113-11111111111111111111114, 11111111111111111111114-11111111111111111111115, 11111111111111111111115-11111111111111111111116, 11111111111111111111116-11111111111111111111117, 11111111111111111111117-11111111111111111111118, 11111111111111111111118-11111111111111111111119, 11111111111111111111119-111111111111111111111110, 111111111111111111111110-111111111111111111111111, 111111111111111111111111-111111111111111111111112, 111111111111111111111112-111111111111111111111113, 111111111111111111111113-111111111111111111111114, 111111111111111111111114-111111111111111111111115, 111111111111111111111115-111111111111111111111116, 111111111111111111111116-111111111111111111111117, 111111111111111111111117-111111111111111111111118, 111111111111111111111118-111111111111111111111119, 111111111111111111111119-1111111111111111111111110, 1111111111111111111111110-1111111111111111111111111, 1111111111111111111111111-1111111111111111111111112, 1111111111111111111111112-1111111111111111111111113, 1111111111111111111111113-1111111111111111111111114, 1111111111111111111111114-1111111111111111111111115, 1111111111111111111111115-1111111111111111111111116, 1111111111111111111111116-1111111111111111111111117, 1111111111111111111111117-1111111111111111111111118, 1111111111111111111111118-1111111111111111111111119, 1111111111111111111111119-11111111111111111111111110, 11111111111111111111111110-11111111111111111111111111, 11111111111111111111111111-11111111111111111111111112, 11111111111111111111111112-11111111111111111111111113, 11111111111111111111111113-11111111111111111111111114, 11111111111111111111111114-11111111111111111111111115, 11111111111111111111111115-11111111111111111111111116, 11111111111111111111111116-11111111111111111111111117, 11111111111111111111111117-11111111111111111111111118, 11111111111111111111111118-11111111111111111111111119, 11111111111111111111111119-111111111111111111111111110, 111111111111111111111111110-111111111111111111111111111, 111111111111111111111111111-111111111111111111111111112, 111111111111111111111111112-111111111111111111111111113, 111111111111111111111111113-111111111111111111111111114, 111111111111111111111111114-111111111111111111111111115, 111111111111111111111111115-111111111111111111111111116, 111111111111111111111111116-111111111111111111111111117, 111111111111111111111111117-111111111111111111111111118, 111111111111111111111111118-111111111111111111111111119, 111111111111111111111111119-1111111111111111111111111110, 1111111111111111111111111110-1111111111111111111111111111, 1111111111111111111111111111-111

Snapshot #23 — Masked self-attention values (V)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["v_m"]`  
Observed shape: (1,5,128)

$V$  provides features to be aggregated via masked attention weights.

```
97     class MultiHeadAttention(nn.Module): 3 usages
98
99     114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  k_in: tensor([[ [ 5.0110e-01,  9.9553e-01
100    ...      q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)  q_heads: tensor([[[[ 4.8121e-01,  4.0718e-01,  2.6955e-01
101    121      k_heads = self._split_heads(k)  k_heads: tensor([[[[ 9.2225e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02, \n
102    122      v_heads = self._split_heads(v)  v_heads: tensor([[[[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01, \n
103
104      # Scaled dot-product attention
105      # pre-softmax scores (no mask yet)
106      scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)  scores_pre_mask: tensor([[[[ 0.1813, -0.1693,
107
108      # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
109
110
111      scores = scores_pre_mask.clone()
112
113      if mask is not None:
114          # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
115          scores = scores + mask # SNAPSHOT #25 - Mask tensor gets added here
116
117
118          attn_weights = F.softmax(scores, dim=-1) # SNAPSHOT #11 or #26 or #34 - After softmax
119          attn_out_heads = attn_weights @ v_heads # (b,n,tgt_len,d_head)
120
121          attn_concat = attn_out_heads.transpose(dim0=1, dim1=2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)
122
123      # SNAPSHOT #3 or #28 or #35 - Multi-head output after concatenation
124
125
126
127
128
129
130
131
132
133
134
135
136
137
```

## Snapshot #24 — Masked attention scores before mask

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["scores_pre_mask_m"]`  
Observed shape: (1,4,5,5)

Raw per-head scores before masking; square because masked self-attention is over the target sequence.

## Snapshot #25 — Mask tensor

# Explanation & Shape

Variable (as shown in PyCharm): `refs["tgt_mask"]`

**Observed shape:** (1,1,5,5) with 0 and  $-\infty$

Upper-triangular  $-\infty$  above the diagonal prevents attention to future positions, enforcing autoregressive decoding.

## Snapshot #26 — Masked attention scores after mask + softmax

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["attn_weights_m"]`

Observed shape: (1,4,5,5)

After mask + softmax, probabilities at masked positions are  $\approx 0$ ; rows sum to  $\approx 1$ .

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109         # x: (batch, seq, d_model)
110         b, s, _ = x.size()
111         x = x.view(b, s, self.nhead, self.d_head).transpose_(dim0: 1, dim1: 2) # (batch, heads, seq, d_head)
112         return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  mask: None  K_in: tensor([[ 5.0110e-01,  9.9553e-01,
115         # q_in, k_in, v_in: (batch, seq, d_model)
116         q = self.W.q(q_in)  # SNAPSHOT #7 or #21 or #30 - Q (depending on colten)  q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,  4.7361e-01,  1.0521e+00,  1.8521e+00,
117         k = self.W.k(k_in)  # SNAPSHOT #8 or #22 or #31 - K  k: tensor([[ 9.2225e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,  4.7361e-01,  1.0521e+00,  1.8521e+00,
118         v = self.W.v(v_in)  # SNAPSHOT #9 or #23 or #32 - V  v: tensor([[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2721e-01, -9.2871e-01, -2.6742e-01,  6.1655e-01,  1.1496e+00,
119
120         q.heads = self._split_heads(q)  # SNAPSHOT #12 or #27 - q multi-head split (shape: b,h,seq,d_head)  q.heads: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,  4.7361e-01,  1.0521e+00,  1.8521e+00,
121         k.heads = self._split_heads(k)  k.heads: tensor([[ 9.2225e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,  4.7361e-01,  1.0521e+00,  1.8521e+00,
122         v.heads = self._split_heads(v)  v.heads: tensor([[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01, -2.6742e-01,  6.1655e-01,  1.1496e+00,
123
124         # Scaled dot-product attention
125         # pre-softmax scores (no mask yet)
126         scores_pre_mask = (q.heads @ k.heads.transpose(-2, -1)) / math.sqrt(self.d_head)  # (b,h,tgt_len,src_len)
127         # SNAPSHOT #16 or #24 or #33 - Attention score matrix before softmax (pre-mask)
128
129         scores = scores_pre_mask.clone()

```

Threads & Variables    Console

MainThread

```

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> names = (NoneType) None
> names = (NoneType) None
> nbbytes = (int) 3072
> ndim = (int) 4
> output_nr = (int) 0
> real = (Tensor (4, 6, 32)) tensor([[ 9.2225e-01, -3.9407e-01,  1.3553e-01, -9.5973e-02,  6.4861e-02,  4.7361e-01,  1.0521e+00, -3.7656e-01, ...01, -3.7272e-01, -2.6283e-01,  1.9172e+00, -6.904...View as Array
> requires_grad = (bool) True
> shape = (Size (4, 6, 32))
> retains_grad = (bool) False
> protected_attributes
> k_in = (Tensor (1, 6, 128)) tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00, -1.2773e-01, -1.4915e+00,  4.0681e-01, -1.1047e+00,  1.2238e+00, ...-01, -1.7689e+00,  4.1540e-01,  8.8161e-01,  1.5257e+00,...View as Array
> mask = (NoneType) None

```

Sheet\_Assignment1 > assignment\_transformer\_debug.py

## Snapshot #27 — Masked self-attention multi-head split

### Explanation & Shape

**Variable (as shown in PyCharm):** refs["dec\_traces"]的文化["q\_heads\_m"] (& k\_heads\_m, v\_heads\_m)

**Observed shape:** (1,4,5,32) each

Splitting into 4 heads of 32 captures multiple relations per token in parallel.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  mask: None  K_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00, -1.2773e-01, -1.4915e+00,  4.0681e-01, -1.1047e+00,  1.2238e+00, ...-01, -1.7689e+00,  4.1540e-01,  8.8161e-01,  1.5257e+00,...View as Array
99             # q_in, k_in, v_in: (batch, seq, d_model)
100            v_heads = self._split_heads(v_in)  v_heads: tensor([[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01, -2.6742e-01,  6.1655e-01,  1.1496e+00,
101
102            # Scaled dot-product attention
103            # pre-softmax scores (no mask yet)
104            scores_pre_mask = (q_in @ k_in.transpose(-2, -1)) / math.sqrt(self.d_head)  # (batch, 1, tgt_len, src_len)  scores_pre_mask: tensor([[ 0.1813, -0.1693, -0.1295,  0.0085, -0.4855, -0.1307], ...-0.2066, -0.0625, -0.1083,  0.1918,  0.2571,...View as Array
105            # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
106
107            scores = scores_pre_mask.clone()  scores: tensor([[ 0.1813, -0.1693, -0.1295,  0.0085, -0.4855, -0.1307], ...-0.2066, -0.0625, -0.1083,  0.1918,  0.2571,...View as Array
108            if mask is not None:
109                # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
110                scores = scores + mask  # SNAPSHOT #28 - Mask tensor gets added here
111
112            attn_weights = F.softmax(scores, dim=-1)  # SNAPSHOT #11 or #26 or #34 - After softmax onto weights: tensor([[ [0.2210, 0.1557, 0.1620, 0.1860, 0.1135, 0.1618], ...-0.3751e-01, 4.4334e-01, -1.3415e-02, ...-0.10601e+00, -5.9945e-01, -3.9480e-01, -1.0753e-02,...View as Array
113            attn_out_heads = attn_weights @ v_heads  # (batch, 1, tgt_len, d_head)  attn_out_heads: tensor([[ [1.5819e-01, 2.3866e-01, 1.7305e-01, -2.8876e-01, -6.7833e-01, ...-0.3751e-01, 4.4334e-01, -1.3415e-02, ...00, -5.9945e-01, -3.9480e-01, -1.0753e-02,...View as Array
114            attn_concat = attn_out_heads.transpose(dim0: 1, dim1: 2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)  attn_concat: tensor([[ 1.5819e-01, 2.3866e-01, 1.7305e-01, -2.8876e-01, -6.7833e-01, ...-0.3751e-01, 4.4334e-01, -1.3415e-02, ...00, -5.9945e-01, -3.9480e-01, -1.0753e-02,...View as Array
115            # SNAPSHOT #13 or #28 or #35 - Multi-head output after concatenation
116
117            out = self.W_o(attn_concat)
118            return out, q, k, v, q_heads, k_heads, v_heads, scores_pre_mask, attn_weights, attn_concat
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
815
816
817
818
819
819
820
821
822
823
824
825
825
826
827
828
829
829
830
831
832
833
834
835
835
836
837
838
839
839
840
841
842
843
844
844
845
846
847
848
848
849
849
850
851
852
853
854
854
855
856
857
858
858
859
859
860
861
862
863
863
864
865
865
866
867
867
868
868
869
869
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
135
```

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["attn_concat_m"]`  
Observed shape: (1,5,128)

Per-head outputs are concatenated to 128 to preserve residual compatibility.

## Snapshot #29 — Residual + normalization after masked self-attention

## Explanation & Shape

**Variable (as shown in PyCharm):** refs["dec\_traces"][0]["after\_msa\_norm"]  
**Observed shape:** (1,5,128)

Residual (decoder input + masked self-attn output) followed by LayerNorm; shape unchanged.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109         # x: (batch, seq, d_model)
110         b, s, _ = x.size()
111         x = x.view(b, s, self.nhead, self.d_head).transpose(dim0=1, dim1=2) # (batch, heads, seq, d_head)
112         return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None): k_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00,
...
115         # q_in, k_in, v_in: (batch, seq, d_model)
116         q = self.W.q(q_in) # SNAPSHOT #7 or #21 or #30 - Q (depending on column) q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
117         k = self.W.k(k_in) # SNAPSHOT #8 or #22 or #31 - K
118         v = self.W.v(v_in) # SNAPSHOT #9 or #23 or #32 - V
...
119         q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)
120         k_heads = self._split_heads(k)
121         v_heads = self._split_heads(v)
122
123         # Scaled dot-product attention
124         # pre-softmax scores (no mask yet)
125         scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b, h, tgt_len, src_len)
126         # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
127
128         scores = scores_pre_mask.clone()
129
Debug assignment_transformer_debug.x
```

Threads & Variables    Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

MainThread

- forward, assignment\_transformer\_debug.py:117
- ... wrapped\_call\_impl.module.py:1784
- wrapped\_call\_impl.module.py:1773
- forward, assignment\_transformer\_debug.py:175
- ... wrapped\_call\_impl.module.py:1784
- wrapped\_call\_impl.module.py:1773
- forward, assignment\_transformer\_debug.py:304
- ... wrapped\_call\_impl.module.py:1784
- wrapped\_call\_impl.module.py:1773
- main, assignment\_transformer\_debug.py:340
- <module>, assignment\_transformer\_debug.py:345

Sheet\_Assignment > assignment\_transformer\_debug.py

## Snapshot #30 — Cross-attention queries (from decoder)

### Explanation & Shape

**Variable (as shown in PyCharm):** refs["dec\_traces"][0]["q\_c"]  
**Observed shape:** (1,5,128)

Queries derived from decoder states to attend over encoder outputs.

```

97     class MultiHeadAttention(nn.Module): 3 usages
98         def _split_heads(self, x: torch.Tensor) -> torch.Tensor: 3 usages
...
109         # x: (batch, seq, d_model)
110         b, s, _ = x.size()
111         x = x.view(b, s, self.nhead, self.d_head).transpose(dim0=1, dim1=2) # (batch, heads, seq, d_head)
112         return x
113
114     def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None): k_in: tensor([[ 5.0110e-01,  9.9553e-01,  1.0061e+00,
...
115         # q_in, k_in, v_in: (batch, seq, d_model)
116         q = self.W.q(q_in) # SNAPSHOT #7 or #21 or #30 - Q (depending on column) q: tensor([[ 4.8121e-01,  4.0718e-01,  2.9297e-01,  2.6150e-01,  1.7796e-01,
117         k = self.W.k(k_in) # SNAPSHOT #8 or #22 or #31 - K
118         v = self.W.v(v_in) # SNAPSHOT #9 or #23 or #32 - V
...
119         q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - Q multi-head split (shape: b,h,seq,d_head)
120         k_heads = self._split_heads(k)
121         v_heads = self._split_heads(v)
122
123         # Scaled dot-product attention
124         # pre-softmax scores (no mask yet)
125         scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b, h, tgt_len, src_len)
126         # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
127
128         scores = scores_pre_mask.clone()
129
Debug assignment_transformer_debug.x
```

Threads & Variables    Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

MainThread

- forward, assignment\_transformer\_debug.py:117
- ... wrapped\_call\_impl.module.py:1784
- wrapped\_call\_impl.module.py:1773
- forward, assignment\_transformer\_debug.py:175
- ... wrapped\_call\_impl.module.py:1784
- wrapped\_call\_impl.module.py:1773
- forward, assignment\_transformer\_debug.py:304
- ... wrapped\_call\_impl.module.py:1784
- wrapped\_call\_impl.module.py:1773
- main, assignment\_transformer\_debug.py:340
- <module>, assignment\_transformer\_debug.py:345

Sheet\_Assignment > assignment\_transformer\_debug.py

## Snapshot #31 — Cross-attention keys (from encoder)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["k_c"]`  
Observed shape: (1,6,128)

Keys originate from encoder outputs (src length = 6).

The screenshot shows the PyCharm IDE interface with the following details:

- Top Bar:** Shows tabs for "Sheet\_Assignment1" and "assignment\_transformer\_debug.py", along with icons for file operations, search, and help.
- Code Editor:** Displays the `assignment_transformer_debug.py` file. The code implements a MultiHeadAttention module. A memory dump (snapshot) is being analyzed across multiple frames (Frame #7, #8, #9, #10, #21, #22, #23, #31, #32, #33). The current frame (#10) highlights the calculation of attention scores before softmax.
- Bottom Bar:** Shows the "Debug" tab is selected, and the expression `(assignment_transformer_debug.py:120)` is being evaluated.
- Bottom Left:** Shows the stack trace for the current thread, listing calls from `assignment_transformer_debug.py:120` up to the Python runtime.
- Bottom Right:** Shows memory dump details for variable `arr`, including its type (`[Type: Traceback]`), size (`1.5 MB`), and a preview of its contents as a 2D array.

Snapshot #32 — Cross-attention values (from encoder)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["v_c"]`  
Observed shape: (1,6,128)

Values also come from encoder outputs and will be aggregated into decoder states.

```

97     class MultiHeadAttention(nn.Module): 3 usages
114         def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  k_in: tensor([[ 5.0110e-01,  9.9553e-01,  ...
115             q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - 0 multi-head split (shape: b,h,seq,d_head)  q_heads: tensor([[ 4.8121e-01,  4.0718e-01,  2.5 ...
116             k_heads = self._split_heads(k)  k_heads: tensor([[ 9.2225e-01, -3.9407e-01, -1.3553e-01, -9.5973e-02,  6.4881e-02,  ...
117             v_heads = self._split_heads(v)  v_heads: tensor([[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01,  ...
118
119             # Scaled dot-product attention
120             # pre-softmax scores (no mask yet)
121             scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)  scores_pre_mask: tensor([[ 0.1813, -0.169 ...
122             # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
123
124             scores = scores_pre_mask.clone()
125             if mask is not None:
126                 # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
127                 scores = scores + mask # SNAPSHOT #25 - Mask tensor gets added here
128
129             attn_weights = F.softmax(scores, dim=-1) # SNAPSHOT #11 or #28 or #34 - After softmax
130             attn_out_heads = attn_weights @ v_heads # (b,h,tgt_len,d_head)
131             attn_concat = attn_out_heads.transpose(dim0: 1, dim1: 2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)
132
133             # SNAPSHOT #13 or #28 or #35 - Multi-head output after concatenation
134
135             scores = scores_pre_mask.clone()
136             if mask is not None:
137                 # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
138                 scores = scores + mask # SNAPSHOT #25 - Mask tensor gets added here
139
140             attn_weights = F.softmax(scores, dim=-1) # SNAPSHOT #11 or #26 or #36 - After softmax  attn_weights: tensor([[ 0.2210,  0.1557,  0.1620,  0.1860,  0.1135,  0.1618], ...
141             attn_out_heads = attn_weights @ v_heads # (b,h,tgt_len,d_head)
142             attn_concat = attn_out_heads.transpose(dim0: 1, dim1: 2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)
143
144             out = self.W_o(attn_concat)
145             return out, q, k, v, q_heads, k_heads, v_heads, scores_pre_mask, attn_weights, attn_concat
146
147

```

The screenshot shows the PyCharm IDE with the code for the `MultiHeadAttention` class. A debugger window is open, showing the state of the variable `scores_pre_mask`. The tensor has a shape of (1, 4, 5, 6). The values are as follows:

Batch	Head	Query	Key	Value	Score
0	0	0	0	0	0.1813
0	0	0	1	0	-0.1693
0	0	0	2	0	-0.1295
0	0	0	3	0	0.0085
0	0	0	4	0	-0.4855
0	0	0	5	0	-0.1307
0	1	0	0	0	-0.2066
0	1	0	1	0	-0.0625
0	1	0	2	0	-0.1083
0	1	0	3	0	0.1918
0	1	0	4	0	0.2571
0	1	0	5	0	-0.0351
0	2	0	0	0	0.2268
0	2	0	1	0	-0.1117
0	2	0	2	0	-0.0104
0	2	0	3	0	-0.0408
0	2	0	4	0	-0.0
0	2	0	5	0	-0.029

### Snapshot #33 — Cross-attention score matrix before softmax

#### Explanation & Shape

**Variable (as shown in PyCharm):** `refs["dec_traces"][0]["scores_pre_mask_c"]`  
**Observed shape:** (1,4,5,6)

Per-head scores are rectangular (tgt\_len = 5 queries over src\_len = 6 keys).

```

97     class MultiHeadAttention(nn.Module): 3 usages
114         def forward(self, q_in: torch.Tensor, k_in: torch.Tensor, v_in: torch.Tensor, mask: torch.Tensor | None = None):  k_in: tensor([[ 5.0110e-01,  9.9553e-01,  ...
115             q_heads = self._split_heads(q) # SNAPSHOT #12 or #27 - 0 multi-head split (shape: b,h,seq,d_head)  q_heads: tensor([[ 4.8121e-01,  4.0718e-01,  2.5 ...
116             k_heads = self._split_heads(k)  k_heads: tensor([[ 9.2225e-01, -3.9407e-01, -1.3553e-01, -9.5973e-02,  6.4881e-02,  ...
117             v_heads = self._split_heads(v)  v_heads: tensor([[ 6.4838e-01,  8.0307e-01,  1.2474e+00,  1.2914e-01, -9.2871e-01,  ...
118
119             # Scaled dot-product attention
120             # pre-softmax scores (no mask yet)
121             scores_pre_mask = (q_heads @ k_heads.transpose(-2, -1)) / math.sqrt(self.d_head) # (b,h,tgt_len,src_len)  scores_pre_mask: tensor([[ 0.1813, -0.169 ...
122             # SNAPSHOT #10 or #24 or #33 - Attention score matrix before softmax (pre-mask)
123
124             scores = scores_pre_mask.clone()
125             if mask is not None:
126                 # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
127                 scores = scores + mask # SNAPSHOT #25 - Mask tensor gets added here
128
129             attn_weights = F.softmax(scores, dim=-1) # SNAPSHOT #11 or #28 or #34 - After softmax
130             attn_out_heads = attn_weights @ v_heads # (b,h,tgt_len,d_head)
131             attn_concat = attn_out_heads.transpose(dim0: 1, dim1: 2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)
132
133             # SNAPSHOT #13 or #28 or #35 - Multi-head output after concatenation
134
135             scores = scores_pre_mask.clone()
136             if mask is not None:
137                 # mask shape expected: (batch, 1, tgt_len, src_len) with 0 for keep and -inf for block
138                 scores = scores + mask # SNAPSHOT #25 - Mask tensor gets added here
139
140             attn_weights = F.softmax(scores, dim=-1) # SNAPSHOT #11 or #26 or #36 - After softmax  attn_weights: tensor([[ 0.2210,  0.1557,  0.1620,  0.1860,  0.1135,  0.1618], ...
141             attn_out_heads = attn_weights @ v_heads # (b,h,tgt_len,d_head)
142             attn_concat = attn_out_heads.transpose(dim0: 1, dim1: 2).contiguous().view(q_in.size(0), q_in.size(1), self.d_model)
143
144             out = self.W_o(attn_concat)
145             return out, q, k, v, q_heads, k_heads, v_heads, scores_pre_mask, attn_weights, attn_concat
146
147

```

The screenshot shows the PyCharm IDE with the code for the `MultiHeadAttention` class. A debugger window is open, showing the state of the variable `scores`. The tensor has a shape of (1, 4, 5, 6). The values are as follows:

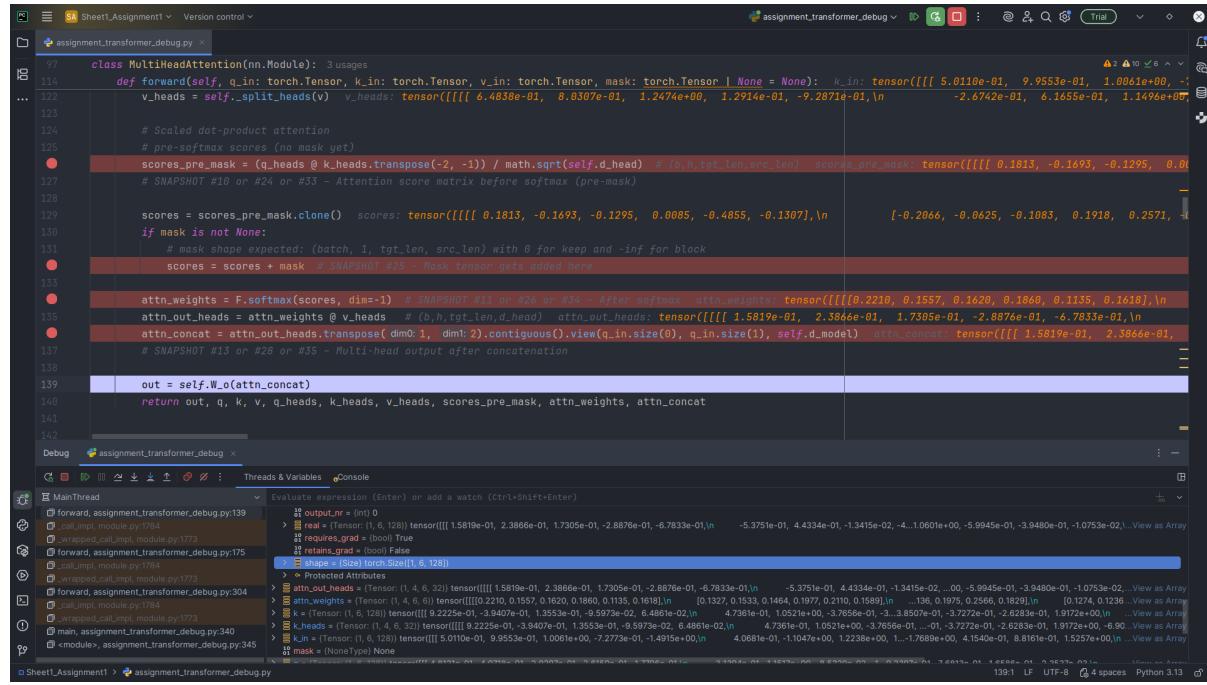
Batch	Head	Query	Key	Value	Score
0	0	0	0	0	0.1813
0	0	0	1	0	-0.1693
0	0	0	2	0	-0.1295
0	0	0	3	0	0.0085
0	0	0	4	0	-0.4855
0	0	0	5	0	-0.1307
0	1	0	0	0	-0.2066
0	1	0	1	0	-0.0625
0	1	0	2	0	-0.1083
0	1	0	3	0	0.1918
0	1	0	4	0	0.2571
0	1	0	5	0	-0.0351
0	2	0	0	0	0.2268
0	2	0	1	0	-0.1117
0	2	0	2	0	-0.0104
0	2	0	3	0	-0.0408
0	2	0	4	0	-0.0
0	2	0	5	0	-0.029
0	3	0	0	0	0.2268
0	3	0	1	0	-0.1117
0	3	0	2	0	-0.0104
0	3	0	3	0	-0.0408
0	3	0	4	0	-0.0
0	3	0	5	0	-0.029
0	4	0	0	0	0.2268
0	4	0	1	0	-0.1117
0	4	0	2	0	-0.0104
0	4	0	3	0	-0.0408
0	4	0	4	0	-0.0
0	4	0	5	0	-0.029
0	5	0	0	0	0.2268
0	5	0	1	0	-0.1117
0	5	0	2	0	-0.0104
0	5	0	3	0	-0.0408
0	5	0	4	0	-0.0
0	5	0	5	0	-0.029

### Snapshot #34 — Cross-attention score matrix after softmax

## Explanation & Shape

**Variable (as shown in PyCharm):** `refs["dec_traces"][0]["attn_weights_c"]`  
**Observed shape:** (1,4,5,6)

Row-wise probabilities over encoder positions; each row sums  $\approx 1$ .



The screenshot shows the PyCharm debugger interface with the code editor and the debugger tool window. In the code editor, the `forward` method of the `MultiHeadAttention` class is being debugged. A red box highlights the assignment of `scores` to `scores + mask`. In the debugger tool window, the variable `out` is selected. The expression `out = self.W_o(attn_concat)` is evaluated, showing its type as `Tensor` and its size as `(1, 6, 128)`. The value of `out` is displayed as a large tensor with numerical values.

Snapshot #35 — Cross-attention output after concatenation

## Explanation & Shape

**Variable (as shown in PyCharm):** `refs["dec_traces"][0]["attn_concat_c"]`  
**Observed shape:** (1,5,128)

Concatenated per-head outputs, restoring  $d_{\text{model}} = 128$  for residual addition.

Snapshot #36 — Residual + normalization after cross-attention

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["after_ca_norm"]`  
Observed shape: (1,5,128)

Residual (decoder state + cross-attn output) then LayerNorm; shape stays (1, 5, 128).

# Feed Forward

```
144 # ---  
145  
146 class PositionwiseFFN(nn.Module): 2 usages  
147     def __init__(self, d_model: int, d_ff: int):  
148         super().__init__()  
149         self.fc1 = nn.Linear(d_model, d_ff)  
150         self.fc2 = nn.Linear(d_ff, d_model)  
151  
152     def forward(self, x): self: PositionwiseFFN(  
153         fc1: Linear(in_features=128, out_features=256, bias=True)  
154         fc2: Linear(in_features=256, out_features=128, bias=True)  
155         ff_in = x  
156         x1 = self.fc1(ff_in) # SNAPSHOT #16 or #37 - FF input ff_in tensor([[ 1.5524e-02, 2.6189e-01, 7.3034e-02, -9.6649e-01, -1.7981e+00, -1.3071e-01, -1.3029e+00, 5.1498e-01, 3.3e-01, 1.4038e-01, 6.5498e-01])  
157         x2 = F.relu(x1)  
158         x3 = self.fc2(x2) # SNAPSHOT #18 or #39 - FF second linear  
159         return x3, ff_in, x1, x3  
160  
161 # Encoder/Decoder Layers  
162 # ---  
163 class EncoderLayer(nn.Module): 1 usage  
164     def __init__(self, d_model, nhead, d_ff):  
165         super().__init__()  
166         self.mha = MultiHeadAttention(d_model, nhead)
```

Snapshot #37 — Decoder feed-forward input

# Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["ff_in"]`  
Observed shape: (1,5,128)

Input to the decoder FFN;  $d_{\text{model}}$  stays constant across blocks to support residuals.

The screenshot shows a PyCharm IDE window with two tabs open: `assignment_transformer_debug.py` and `module.py`. The `assignment_transformer_debug.py` tab contains Python code for a PositionwiseFFN module. A memory dump is being analyzed at line 152, showing tensors `ff_in`, `x1`, `x2`, and `x3` with their respective values and shapes. The `Threads & Variables` tool window is open, showing the current stack trace and variable information. The `Console` tab is also visible.

```
144 # Feed Forward
145 #
146 class PositionwiseFFN(nn.Module): 2 usages
147     def __init__(self, d_model: int, d_ff: int):
148         super().__init__()
149         self.fc1 = nn.Linear(d_model, d_ff)
150         self.fc2 = nn.Linear(d_ff, d_model)
151
152     def forward(self, x): x: tensor([[1.15524e-02, 2.6189e-01, 7.3034e-02, -9.6649e-01, -1.7981e+00, ..., -1.3071e-01, -1.3029e+00, 5.1498e-01, 3...e-01, 1.4038e
153         ff_in = x           # SNAPSHOT #16 on #37 - FF input ff_in: tensor([[1.15524e-02, 2.6189e-01, 7.3034e-02, -9.6649e-01, -1.7981e+00, ..., -1.3071e-01, -1.3029e+00, 5.1498e-01, 3...e-01, 1.4038e
154         x1 = self.fc1(ff_in)          # SNAPSHOT #17 on #38 - FF first linear x1: tensor([[[-0.7218, 0.5992, -0.5013, ..., 0.1686, -0.3616, 0.5029], ...])
155         x2 = F.relu(x1)
156         x3 = self.fc2(x2)           # SNAPSHOT #18 on #37 - FF second linear
157
158         return x3, ff_in, x1, x3
159
160     # -----
161     # Encoder/Decoder Layers
162     #
163
164     class EncoderLayer(nn.Module): 1 usage
165         def __init__(self, d_model, nhead, d_ff):
166             super().__init__()
167             self.mha = MultiheadAttention(d_model, nhead)
```

Snapshot #38 — Feed-forward first linear layer output

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["ff1"]`  
Observed shape: (1,5,256)

Expanded to  $d_{ff} = 256$  for more capacity and non-linearity.

Snapshot #39 — Feed-forward second linear layer output

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["ff2"]`  
Observed shape: (1,5,128)

Projected back to 128 to match the residual branch.

## Snapshot #40 — Decoder block final output tensor

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_traces"][0]["dec_out"]`

Observed shape: (1,5,128)

Output of the decoder block after FFN residual + LayerNorm; feeds the next decoder layer or final projection.

## Final Output (Snapshots 41–43)

Snapshot #41 — Decoder final sequence output (before projection)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["dec_final"]`

**Observed shape:** (1,5,128)

Final decoder hidden states after all decoder layers; still in model space 128.

## Snapshot #42 — Logits after final linear projection

## Explanation & Shape

Variable (as shown in PyCharm): `refs["logits"]`

**Observed shape:** (1,5,*tgt\_vocab\_size*)

A linear map converts  $128 \rightarrow vocab\_size$  per time step, producing unnormalized log-probabilities (logits) over the target vocabulary.

```
265     class TinyTransformer(nn.Module): 1 usage
266
267     def forward(self, src_ids, tgt_ids):  self: TinyTransformer<n
268         (src_emb): Embedding(6, 128)<n
269         (tgt_emb): Embedding(5, 128)<n
270         (pos): PositionalEncoding<n
271
272         # Decoder stack
273
274         y = tgt_with_pos  y: tensor([[[-1.8276e-01,  3.1504e-01, -1.801e+00, -1.5486e+00, -1.1451e+00, ..., -7.7362e-01, -5.6076e-01, -3.4116e-01, 5.6076e-01, -1.8276e-01], [0.0, 0.0, 0.0, 0.0, 0.0, ..., 0.0, 0.0, 0.0, 0.0, 0.0]]])  # View as Array
275
276         tgt_mask = self.make_subsequent_mask(tgt.size(1), batch_size=tgt.size(0))  # SNAPSHOT #25 - Mask tensor  tgt_mask: tensor([[[[0., -inf, -inf, -inf, -inf], [0., 0., -inf, -inf, -inf], [0., 0., 0., -inf, -inf], [0., 0., 0., 0., -inf], [0., 0., 0., 0., 0.]]]])  # View as Array
277
278         dec_traces = []  dec_traces: []
279         for li, dec in enumerate(self.dec_layers):  dec: DecoderLayer<n
280             (self.mha): MultiHeadAttention<n
281             (W_q): Linear(in_features=128, out_features=128)
282             cap = (li == 0)  # capture a full trace for the FIRST decoder block only
283             y, dec_trace = dec(y, enc_out, tgt_mask, capture=cap)  dec_trace: None
284             dec_traces.append(dec_trace)
285
286
287         dec_final = y  # SNAPSHOT #41 - Decoder final sequence output (before projection)  dec_final: tensor([-1.8276e-01, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
288         logits = self.proj(dec_final)  # SNAPSHOT #42 - Logits after final linear projection  logits: tensor([-0.1160, 0.8900, -0.3211, 0.8900, -0.1160, 0.8900, -0.3211, 0.8900, -0.1160, 0.8900, -0.3211, 0.8900])
289         logits_slice = logits[0, 0, :10]  # SNAPSHOT #43 - Logits slice (first few values for one token)  logits_slice: tensor([-0.1160, 0.8900, -0.3211, 0.8900, -0.1160, 0.8900, -0.3211, 0.8900, -0.1160, 0.8900, -0.3211, 0.8900])
290
291
292         # Useful references for debugger (do not print; inspect in PyCharm)
293         snapshot_refs = dict(
294             # 1 & 2 captured above (src_ids, tgt_ids)
295             emb_weight_slice=emb_weight_slice,  # #3
296             src_embed=src_embed,  # #4
297             enc_ids=[3, 2, 0, 1, 5, 4]
298
299         )
300
301     Debug assignment_transformer.debug x
```

### Snapshot #43 — Logits slice (first few values for one token)

## Explanation & Shape

Variable (as shown in PyCharm): `refs["logits_slice"]`

Observed shape: ( $\leq 10$  elements)

A small slice of a token's logits for quick sanity-check: confirms projection and vocab dimension are correct.

## Guiding Questions — Answers

### 1) What do the dimensions represent at embedding, attention, FFN, and output stages?

Embeddings use  $(\text{batch}, \text{seq\_len}, d_{\text{model}})$ . Attention uses  $(\text{batch}, \text{heads}, \text{tgt\_len}, \text{src\_len})$  for scores/weights and  $(\text{batch}, \text{heads}, \text{seq\_len}, d_{\text{head}})$  for per-head Q/K/V and outputs, with  $d_{\text{head}} = d_{\text{model}}/\text{heads}$ . The FFN preserves  $(\cdot, \cdot, d_{\text{model}})$  externally while expanding internally to  $d_{\text{ff}}$  then projecting back. Logits are  $(\text{batch}, \text{seq\_len}, \text{vocab\_size})$ .

### 2) Why do Q, K, V have the same shape, and why split heads?

They're linear projections of the same  $d_{\text{model}}$ -dimensional token states so dot products  $QK^{\top}$  and weighted sums on  $V$  are compatible. Splitting into heads lets the model attend to different subspaces in parallel.

### 3) What do the attention score matrices represent, and why square?

They are scaled dot-product affinities between queries and keys per head. In self-attention, target and source are the same sequence, so the matrix is square ( $\text{seq} \times \text{seq}$ ). In cross-attention, it's rectangular ( $\text{tgt\_len} \times \text{src\_len}$ ).

### 4) Why is masking necessary in the decoder, and how does the mask enforce it?

Masking prevents access to future positions during training. Adding  $-\infty$  above the diagonal before softmax forces those probabilities to 0, ensuring autoregressive behavior.

### 5) How do residuals and LayerNorm ensure consistency of shapes across blocks?

Residuals require identical shapes so sublayers return  $d_{\text{model}}$ ; LayerNorm normalizes features without changing dimensions, stabilizing depth.

### 6) Why keep the embedding dimension constant through layers?

A fixed  $d_{\text{model}}$  keeps interfaces consistent and enables valid residual additions across stacked blocks.

### 7) How does the final projection connect decoder output to vocabulary logits?

A learned linear map applies at each time step:  $W \in \mathbb{R}^{d_{\text{model}} \times \text{vocab\_size}}$ , producing logits over the target vocabulary.