Lecture 16: Causal Attention

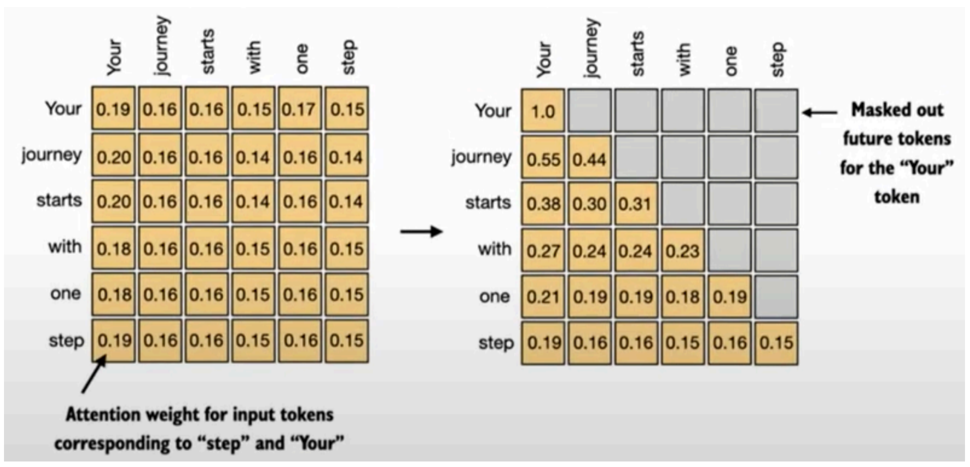
Causal attention, also known as **masked attention** is a special form of self attention.

It restricts the model to only consider **previous and current inputs in a sequence**, when processing any given token.

This is in contrast to the self attention mechanism, which allows access to the entire input sequence at once.

When computing attention scores, the causal attention mechanism ensures that the model only factors in tokens that occur at or before the current token in the sequence.

To achieve this is GPT like LLMs, for each token processed, we mask out the future tokens, which come after the current token in the input text.



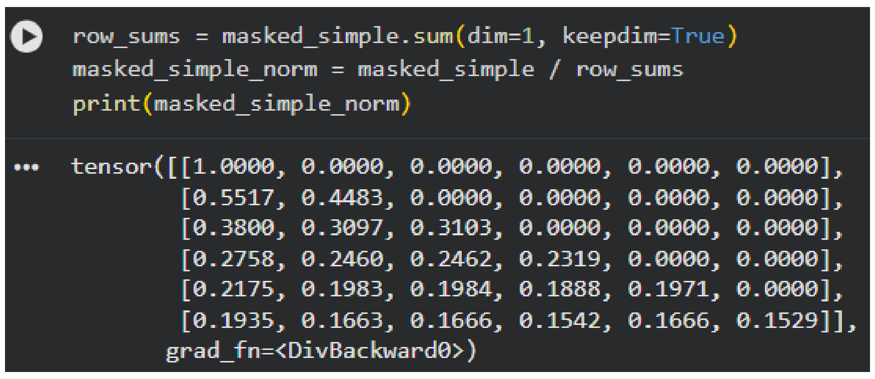
	Your	journey	starts	with	one	step
Your	0.19	0.16	0.16	0.15	0.17	0.15
journey	0.20	0.16	0.16	0.14	0.16	0.14
starts	0.20	0.16	0.16	0.14	0.16	0.14
with	0.18	0.16	0.16	0.15	0.16	0.15
one	0.18	0.16	0.16	0.15	0.16	0.15
step	0.19	0.16	0.16	0.15	0.16	0.15

	Your	journey	starts	with	one	step
Your	1.0					
journey	0.55	0.44				
starts	0.38	0.30	0.31			
with	0.27	0.24	0.24	0.23		
one	0.21	0.19	0.19	0.18	0.19	
step	0.19	0.16	0.16	0.15	0.16	0.15

Attention weight for input tokens corresponding to “step” and “Your”
Masked out future tokens for the “Your” token

We mask out the attention weights above the diagonal, and we normalize the non-masked attention weights, such that the attention weights sum upto 1 in each row.

Applying a Causal Attention Mask



```
row_sums = masked_simple.sum(dim=1, keepdim=True)
masked_simple_norm = masked_simple / row_sums
print(masked_simple_norm)
```

```
tensor([[1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.5517, 0.4483, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.3800, 0.3097, 0.3103, 0.0000, 0.0000, 0.0000],
        [0.2758, 0.2460, 0.2462, 0.2319, 0.0000, 0.0000],
        [0.2175, 0.1983, 0.1984, 0.1888, 0.1971, 0.0000],
        [0.1935, 0.1663, 0.1666, 0.1542, 0.1666, 0.1529]],
       grad_fn=<DivBackward0>)
```

Even though we got this causal attention matrix, there are still some issues.

When we first calculated the attention scores, the first inputs were already influenced by the future tokens. So even though we are zeroing out all the future tokens, it’s not cancelling out the influence of the future tokens. After this, we are again dividing by the sum of all the token weights to normalize. So the influence remains.

This leads to a **Data Leakage Problem.**

The causal attention mask produces an attention weight matrix where all entries **above the diagonal are zero**, and each row forms a **valid probability distribution** (summing to 1).

We could stop here, but there’s a handy mathematical shortcut we can use. Softmax has a useful property:

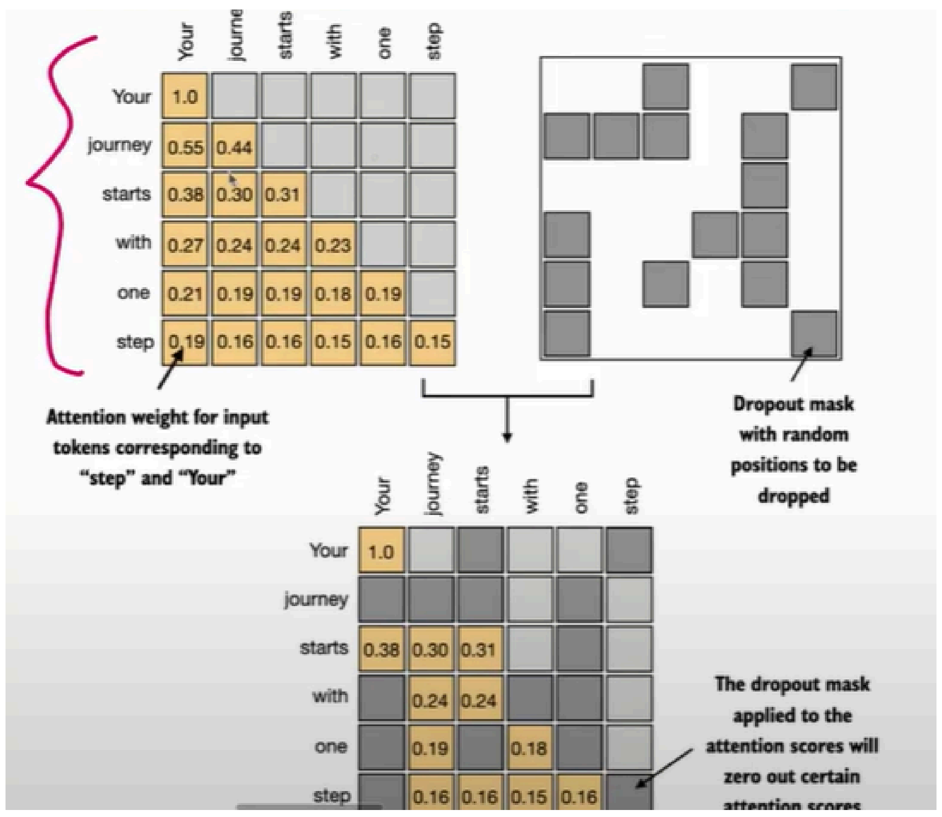
any value that is $-\infty$ becomes 0 after softmax.

So instead of zeroing out the upper-triangle *after* softmax, we can mask *before* softmax by replacing all positions above the diagonal with $-\infty$. That way, softmax **automatically handles the “ignore these positions”** part for us.

To do this efficiently, we build a mask filled with 1’s above the diagonal, then replace those 1’s with $-\infty$ and add it to the attention scores. The softmax then naturally produces the correct causal attention weights.

Dropout in Causal Attention

In causal attention, **dropout** randomly zeroes out some attention weights during training. This prevents the model from relying too heavily on specific tokens, helping **regularize** the network and reduce overfitting, while still respecting the causal (past-only) masking.


Your
journe
starts
with
one
step
Attention weight for input tokens corresponding to "step" and "Your"
Dropout mask with random positions to be dropped
The dropout mask applied to the attention scores will zero out certain attention scores