3 Coding attention mechanisms

This chapter covers

- The reasons for using attention mechanisms in neural networks
- A basic self-attention framework, progressing to an enhanced self-attention mechanism
- A causal attention module that allows LLMs to generate one token at a time
- Masking randomly selected attention weights with dropout to reduce overfitting
- Stacking multiple causal attention modules into a multi-head attention module

At this point, you know how to prepare the input text for training LLMs by splitting text into individual word and subword tokens, which can be encoded into vector representations, embeddings, for the LLM.

Now, we will look at an integral part of the LLM architecture itself, attention mechanisms, as illustrated in figure 3.1. We will largely look at attention mechanisms in isolation and focus on them at a mechanistic level. Then we will code the remaining

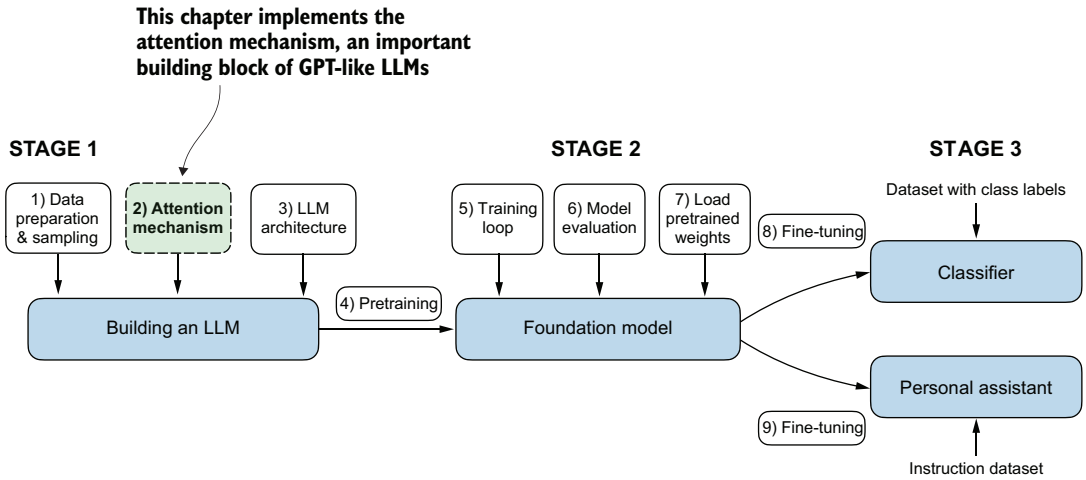

Figure 3.1 The three main stages of coding an LLM. This chapter focuses on step 2 of stage 1: implementing attention mechanisms, which are an integral part of the LLM architecture.

parts of the LLM surrounding the self-attention mechanism to see it in action and to create a model to generate text.

We will implement four different variants of attention mechanisms, as illustrated in figure 3.2. These different attention variants build on each other, and the goal is to

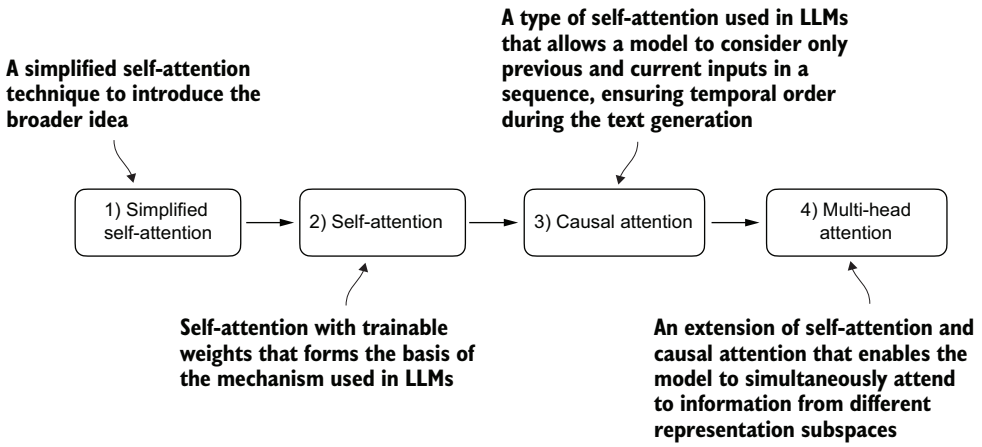

Figure 3.2 The figure depicts different attention mechanisms we will code in this chapter, starting with a simplified version of self-attention before adding the trainable weights. The causal attention mechanism adds a mask to self-attention that allows the LLM to generate one word at a time. Finally, multi-head attention organizes the attention mechanism into multiple heads, allowing the model to capture various aspects of the input data in parallel.

arrive at a compact and efficient implementation of multi-head attention that we can then plug into the LLM architecture we will code in the next chapter.

3.1 The problem with modeling long sequences

Before we dive into the *self-attention* mechanism at the heart of LLMs, let's consider the problem with pre-LLM architectures that do not include attention mechanisms. Suppose we want to develop a language translation model that translates text from one language into another. As shown in figure 3.3, we can't simply translate a text word by word due to the grammatical structures in the source and target language.

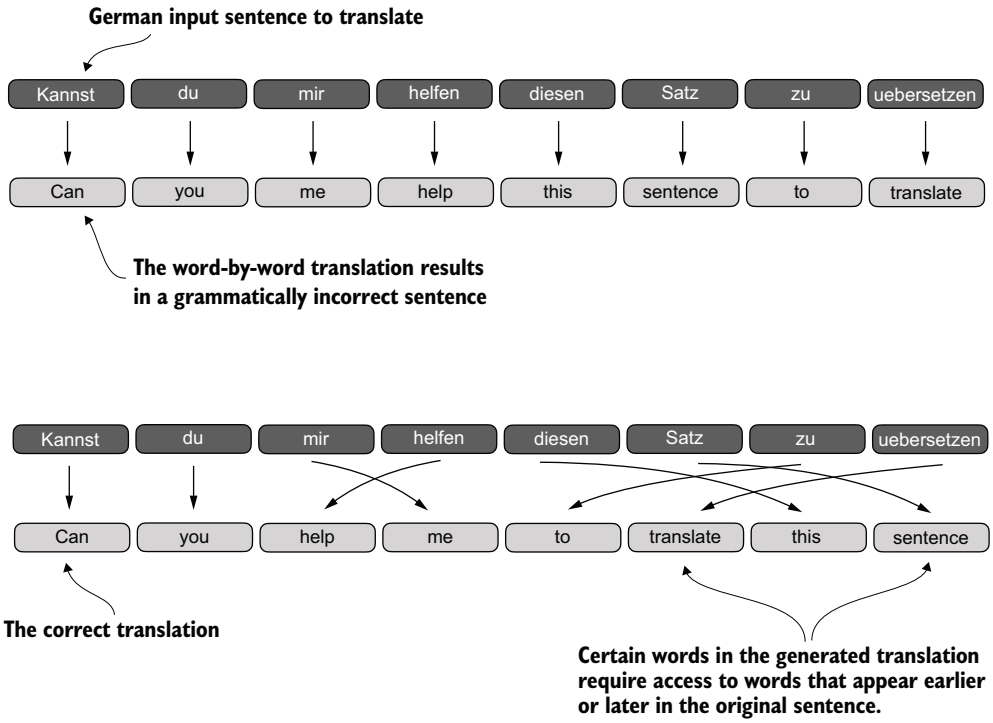

Figure 3.3 When translating text from one language to another, such as German to English, it's not possible to merely translate word by word. Instead, the translation process requires contextual understanding and grammatical alignment.

To address this problem, it is common to use a deep neural network with two submodules, an *encoder* and a *decoder*. The job of the encoder is to first read in and process the entire text, and the decoder then produces the translated text.

Before the advent of transformers, *recurrent neural networks* (RNNs) were the most popular encoder–decoder architecture for language translation. An RNN is a type of neural network where outputs from previous steps are fed as inputs to the current