CHAPTER

# 10 Transformers and Large Language Models

*"How much do we know at any time? Much more, or so I believe, than we know we know."*

Agatha Christie, *The Moving Finger*

Fluent speakers of a language bring an enormous amount of knowledge to bear during comprehension and production. This knowledge is embodied in many forms, perhaps most obviously in the vocabulary, the rich representations we have of words and their meanings and usage. This makes the vocabulary a useful lens to explore the acquisition of knowledge from text, by both people and machines.

Estimates of the size of adult vocabularies vary widely both within and across languages. For example, estimates of the vocabulary size of young adult speakers of American English range from 30,000 to 100,000 depending on the resources used to make the estimate and the definition of what it means to know a word. What is agreed upon is that the vast majority of words that mature speakers use in their day-to-day interactions are acquired early in life through spoken interactions with care givers and peers, usually well before the start of formal schooling. This active vocabulary is extremely limited compared to the size of the adult vocabulary (usually on the order of 2000 words for young speakers) and is quite stable, with very few additional words learned via casual conversation beyond this early stage. Obviously, this leaves a very large number of words to be acquired by other means.

A simple consequence of these facts is that children have to learn about 7 to 10 words a day, *every single day*, to arrive at observed vocabulary levels by the time they are 20 years of age. And indeed empirical estimates of vocabulary growth in late elementary through high school are consistent with this rate. How do children achieve this rate of vocabulary growth? Most of this growth is not happening through direct vocabulary instruction in school, which is not deployed at the rate that would be required to result in sufficient vocabulary growth.

The most likely explanation is that the bulk of this knowledge acquisition happens as a by-product of reading, as part of the rich processing and reasoning that we perform when we read. Research into the average amount of time children spend reading, and the lexical diversity of the texts they read, indicate that it is possible to achieve the desired rate. But the mechanism behind this rate of learning must be remarkable indeed, since at some points during learning the rate of vocabulary growth exceeds the rate at which new words are appearing to the learner!

Many of these facts have motivated approaches to word learning based on the *distributional hypothesis*, introduced in Chapter 6. This is the idea that something about what we're loosely calling word meanings can be learned even without any grounding in the real world, solely based on the content of the texts we encounter over our lives. This knowledge is based on the complex association of words with the words they co-occur with (and with the words that those words occur with).

The crucial insight of the distributional hypothesis is that the knowledge that we acquire through this process can be brought to bear long after its initial acquisition.

Of course, adding grounding from vision or from real-world interaction can help build even more powerful models, but even text alone is remarkably useful.

pretraining

In this chapter we formalize this idea of **pretraining**—learning knowledge about language and the world from vast amounts of text—and call the resulting pretrained language models **large language models**. Large language models exhibit remarkable performance on all sorts of natural language tasks because of the knowledge they learn in pretraining, and they will play a role throughout the rest of this book. They have been especially transformative for tasks where we need to produce text, like summarization, machine translation, question answering, or chatbots.

transformer

The standard architecture for building large language models is the **transformer**. We thus begin this chapter by introducing this architecture in detail. The transformer makes use of a novel mechanism called **self-attention**, which developed out of the idea of **attention** that was introduced for RNNs in Chapter 9. Self-attention can be thought of a way to build contextual representations of a word's meaning that integrate information from surrounding words, helping the model learn how words relate to each other over large spans of text.

We'll then see how to apply the transformer to language modeling, in a setting often called causal or autoregressive language models, in which we iteratively predict words left-to-right from earlier words. These language models, like the feedforward and RNN language models we have already seen, are thus self-trained: given a large corpus of text, we iteratively teach the model to guess the next word in the text from the prior words. In addition to training, we'll introduce algorithms for generating texts, including important methods like **greedy decoding**, **beam search**, and **sampling**. And we'll talk about the components of popular large language models like the GPT family.

Finally, we'll see the great power of language models: almost any NLP task can be modeled as word prediction, if we think about it in the right way. We'll work through an example of using large language models to solve one NLP task of **summarization** (generating a short text that summarizes some larger document). The use of a large language model to generate text is one of the areas in which the impact of the last decade of neural algorithms for NLP has been the largest. Indeed, text generation, along with image generation and code generation, constitute a new area of AI that is often called **generative AI**.

We'll save three more areas of large language models for the next three chapters; Chapter 11 will introduce the **bidirectional transformer** encoder and the method of **masked language modeling**, used for the popular BERT family of models. Chapter 12 will introduce the most powerful way to interact with large language models: **prompting** them to perform other NLP tasks by simply giving directions or instructions in natural language to a transformer that is pretrained on language modeling. And Chapter 13 will introduce the use of the encoder-decoder architecture for transformers in the context of machine translation.

## 10.1 The Transformer: A Self-Attention Network

transformer

In this section we introduce the architecture of the **transformer**, the algorithm that underlies most modern NLP systems. When used for causal language modeling, the input to a transformer is a sequence of words, and the output is a prediction for what word comes next, as well as a sequence of contextual embedding that represents the contextual meaning of each of the input words. Like the LSTMs of Chapter 9,

transformers are a neural architecture that can handle distant information. But unlike LSTMs, transformers are not based on recurrent connections (which can be hard to parallelize), which means that transformers can be more efficient to implement at scale.

Transformers are made up of stacks of transformer **blocks**, each of which is a multilayer network that maps sequences of input vectors $(\mathbf{x}_1, ..., \mathbf{x}_n)$ to sequences of output vectors $(\mathbf{z}_1, ..., \mathbf{z}_n)$ of the same length. These blocks are made by combining simple linear layers, feedforward networks, and **self-attention** layers, the key innovation of transformers. Self-attention allows a network to directly extract and use information from arbitrarily large contexts. We'll start by describing how self-attention works and then return to how it fits into larger transformer blocks. Finally, we'll describe how to use the transformer block together with some input and output mechanisms as a language model, to predict upcoming words from prior words in the context.

self-attention

### 10.1.1 Transformers: the intuition

The intuition of a transformer is that across a series of layers, we build up richer and richer contextualized representations of the meanings of input words or tokens (we will refer to the input as a sequence of words for convenience, although technically the input is first tokenized by an algorithm like BPE, so it is a series of tokens rather than words). At each layer of a transformer, to compute the representation of a word $i$ we combine information from the representation of $i$ at the previous layer with information from the representations of the neighboring words. The goal is to produce a contextualized representation for each word at each position. We can think of these representations as a contextualized version of the static vectors we saw in Chapter 6, which each represented the meaning of a word type. By contrast, our goal in transformers is to produce a contextualized version, something that represents what this word means in the particular context in which it occurs.

We thus need a mechanism that tells us how to weigh and combine the representations of the different words from the context at the prior level in order to compute our representation at this layer. This mechanism must be able to look broadly in the context, since words have rich linguistic relationships with words that can be many sentences away. Even within the sentence, words have important linguistic relationships with contextual words. Consider these examples, each exhibiting linguistic relationships that we'll discuss in more depth in later chapters:

(10.1) The **keys** to the cabinet **are** on the table.

(10.2) **The chicken** crossed the road because **it** wanted to get to the other side.

(10.3) I walked along the **pond**, and noticed that one of the trees along the **bank** had fallen into the **water** after the storm.

In (10.1), the phrase *The keys* is the subject of the sentence, and in English and many languages, must agree in grammatical number with the verb *are*; in this case both are plural. In English we can't use a singular verb like *is* with a plural subject like *keys*; we'll discuss agreement more in Chapter 17. In (10.2), the pronoun *it* corefers to the chicken; it's the chicken that wants to get to the other side. We'll discuss coreference more in Chapter 26. In (10.3), the way we know that *bank* refers to the side of a pond or river and not a financial institution is from the context, including words like *pond* and *water*. We'll discuss word senses more in Chapter 23. These helpful contextual words can be quite far way in the sentence or paragraph,

so we need a mechanism that can look broadly in the context to help compute representations for words.

Self-attention is just such a mechanism: it allows us to look broadly in the context and tells us how to integrate the representation from words in that context from layer $k-1$ to build the representation for words in layer $k$.

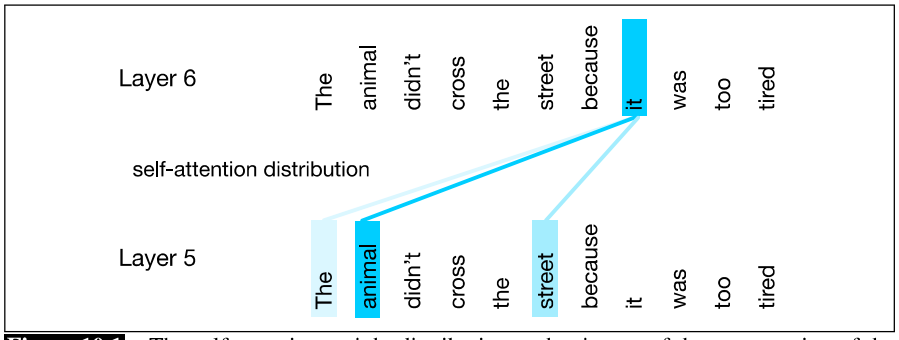


**Figure 10.1** The self-attention weight distribution $\alpha$ that is part of the computation of the representation for the word *it* at layer 6. In computing the representation for *it*, we attend differently to the various words at layer 5, with darker shades indicating higher self-attention values. Note that the transformer is attending highly to *animal*, a sensible result, since in this example *it* corefers with the animal, and so we'd like the representation for *it* to draw on the representation for *animal*. Figure simplified from (Uszkoreit, 2017).

Fig. 10.1 shows an schematic example simplified from a real transformer (Uszkoreit, 2017). Here we want to compute a contextual representation for the word *it*, at layer 6 of the transformer, and we'd like that representation to draw on the representations of all the prior words, from layer 5. The figure uses color to represent the attention distribution over the contextual words: the word *animal* has a high attention weight, meaning that as we are computing the representation for *it*, we will draw most heavily on the representation for *animal*. This will be useful for the model to build a representation that has the correct meaning for *it*, which indeed is coreferent here with the word *animal*. (We say that a pronoun like *it* is coreferent with a noun like *animal* if they both refer to the same thing; we'll return to coreference in Chapter 26.)

### 10.1.2 Causal or backward-looking self-attention

The concept of *context* can be used in two ways in self-attention. In causal, or backward looking self-attention, the context is any of the prior words. In general bidirectional self-attention, the context can include future words. In this chapter we focus on causal, backward looking self-attention; we'll introduce bidirectional self-attention in Chapter 11.

Fig. 10.2 thus illustrates the flow of information in a single causal, or backward looking, self-attention layer. As with the overall transformer, a self-attention layer maps input sequences $(\mathbf{x}_1, ..., \mathbf{x}_n)$ to output sequences of the same length $(\mathbf{a}_1, ..., \mathbf{a}_n)$. When processing each item in the input, the model has access to all of the inputs up to and including the one under consideration, but no access to information about inputs beyond the current one. In addition, the computation performed for each item is independent of all the other computations. The first point ensures that we can use this approach to create language models and use them for autoregressive generation, and the second point means that we can easily parallelize both forward inference and training of such models.

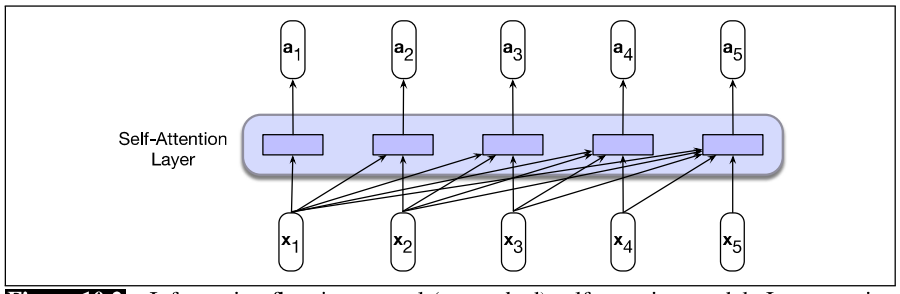


**Figure 10.2** Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

### 10.1.3 Self-attention more formally

We've given the intuition of self-attention (as a way to compute representations of a word at a given layer by integrating information from words at the previous layer) and we've defined context as all the prior words in the input. Let's now introduce the self-attention computation itself.

The core intuition of attention is the idea of *comparing* an item of interest to a collection of other items in a way that reveals their relevance in the current context. In the case of self-attention for language, the set of comparisons are to other words (or tokens) within a given sequence. The result of these comparisons is then used to compute an output sequence for the current input sequence. For example, returning to Fig. 10.2, the computation of $\mathbf{a}_3$ is based on a set of comparisons between the input $\mathbf{x}_3$ and its preceding elements $\mathbf{x}_1$ and $\mathbf{x}_2$, and to $\mathbf{x}_3$ itself.

How shall we compare words to other words? Since our representations for words are vectors, we'll make use of our old friend the **dot product** that we used for computing word similarity in Chapter 6, and also played a role in attention in Chapter 9. Let's refer to the result of this comparison between words $i$ and $j$ as a score (we'll be updating this equation to add attention to the computation of this score):

$$\textit{Verson 1:} \quad \text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \tag{10.4}$$

The result of a dot product is a scalar value ranging from $-\infty$ to $\infty$, the larger the value the more similar the vectors that are being compared. Continuing with our example, the first step in computing $y_3$ would be to compute three scores: $\mathbf{x}_3 \cdot \mathbf{x}_1$, $\mathbf{x}_3 \cdot \mathbf{x}_2$ and $\mathbf{x}_3 \cdot \mathbf{x}_3$. Then to make effective use of these scores, we'll normalize them with a softmax to create a vector of weights, $\alpha_{ij}$, that indicates the proportional relevance of each input to the input element $i$ that is the current focus of attention.

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i \tag{10.5}$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \;\; \forall j \leq i \tag{10.6}$$

Of course, the softmax weight will likely be highest for the current focus element $i$, since $vecx_i$ is very similar to itself, resulting in a high dot product. But other context words may also be similar to $i$, and the softmax will also assign some weight to those words.

Given the proportional scores in $\alpha$, we generate an output value $\mathbf{a}_i$ by summing