

4 Dreams About Talking Machines: Current LLMs

The mentally disturbed do not employ the Principle of Scientific Parsimony: the most simple theory to explain a given set of facts. They shoot for the baroque.

—Philip K. Dick, *Valis*, 1981

The inherently contextual nature of words and sentences is at the heart of how LLMs work.

—Yann LeCun, 2022

What Do Language Machines Do?

The range of tasks that computing machinery has thus far been designed for is impressively broad. Notable among these tasks are those we all rely on in our daily toil. The tasks may be divided into three groups:

Group A: *string classification* tasks, notably (i) summarization and classification of texts, as in automated detection (and elimination) of fake news on social media, for which a division of strings into *true* and *false* ones is critical; and (ii) the establishment of logical relations between input strings that represent statements, for the purpose of reasoning (e.g., string B *follows from* string A, string A *contradicts* string B, etc.).¹

Group B: *string generation* tasks, notably (i) machine translation, which, given a text in an input language L_{input} , outputs a text in L_{output} ($L_{\text{input}} \neq L_{\text{output}}$) that amounts to a proper translation (as judged by native speakers of both languages); (ii) question answering, which accepts an input query (let's say, a text that begins with *what*, *which*, *who*, etc. and/or ends with *?*) and, given a relevant database, outputs a proper answer (as judged by experts in the

relevant domain); and (iii) next word prediction: given a context, name the next word in the stream—one that is not yet input.

Group C: *text-internal* tasks, notably coreference assignment, which links referentially dependent expressions to the correct antecedent expression in the context, whether within or across sentences. These may be pronouns (*he, his, their*), reflexives (*himself, themselves*), or other terms (*that man, the bastard*).

These are special tasks. Some have viewed the trend to build such specialized machines as detrimental, leading to a process that the late MIT computer scientist Patrick Winston (one of the greatest first-generation AI thinkers) dubbed at one time “the mechanistic Balkanization of AI”: a collection of scattered communities that developed task-oriented programs. And while all these tasks are required for proper language processing, one of them, the *next word prediction* task, is presently a main benchmark for network functioning. It requires the machine to use prior knowledge for the identification of a word before it becomes known. This task is reminiscent of the 1950s fill-in-the-blank task that we’ve already encountered (see chapter 1), in which the participant is asked to complete the last word of a given sentence, such as *After pouring milk into her empty bowl at breakfast, Jeannie realized that an ingredient was missing and ran to the convenience store to get _____*. What does a good performer—whether human or digital—need to know and do, in order to make a good guess? I can imagine several distinct strategies, each relying on a different aspect of the utterance, part of which the incoming word is expected to be. First, the performer may try to determine that word’s expected *meaning*, given what is known about the sentence and about the world (i.e., what people like Jeannie tend to put in a bowl with milk in the morning); likely, it’s an edible entity, mixed with milk during breakfast, that can moreover be purchased in a convenience store. Second, the performer can try to identify the missing word’s expected *grammatical form* (“part of speech”), given what is known about the structure of English sentences and what categories may follow a verb; likely, it is the name of a product or a mass noun (*Fruit Loops, cereal*), an adjective (*wild red cereal*) or perhaps a pronoun (*her*) that precedes the top candidate noun (*her cereal*). Third, she can try to calculate its expected *lexical identity*: given the counter we may have in our head that records the frequency of occurrence of each word in a context, what word is most likely to be found

there? For now, I note that the natural language processing community has consistently opted almost exclusively for the third, quantitative path.

Readers may now wonder about a fourth possibility, namely, “all of the above.” Wouldn’t it supply the best approximation of how we humans act? Moreover, when engineering a machine, mightn’t this strategy improve the system’s performance? Finally, is there a reason for the primacy given to the next word prediction task? These are very good questions, which will be addressed below. But first, let’s take a step backwards and look at how the field of natural language processing (NLP) got us here.

A Memory from a Distant Past

In prehistoric times, that is, before Windows and Apple’s macOS, I kept wondering why I had to use a nontransparent code to communicate with my computer. It was clear that life would be much easier if I could just tell the machine in plain language what I wanted it to do and have it return a verbal answer. Sure enough, I was not the only one to wonder. Today, Generation Z has at its disposal bots like ChatGPT, thus far the most impressive incarnation of that ancient dream. The engine behind this chatbot belongs of course to the family of LLMs—multilayered, probabilistic, deep neural networks. Language tasks put an extra burden on deep neural networks, and they are therefore set apart from other computational tasks, especially visual ones like object identification and categorization, in which the input typically requires a single presentation, with no temporal unfolding. In the previous chapter, we saw how superfast computers with extreme storage capacity opened the way to programs like AlexNet, which are capable of processing huge image datasets. Here, we’ll see how the special treatment that language materials require is delivered by LLMs. As we all know, these highly successful technologies have seen remarkable growth, both within and beyond the academic world, and they are the topic of this chapter.

Even the most playful of engineers build machines (especially expensive ones) for a purpose. This purpose may be practical, rather than scientific. Indeed, natural language processing research has for the most part been use-guided and has not attempted to understand why and how humans speak, what underlies our language ability, and what properties of the human mind and brain are behind it. Focusing on specific practical tasks and goals was important, as it liberated technology builders from scientific constraints.

In the case of language, this meant the construction of language machines without a grammar. But if grammar is an inherent human property, then the liberty to ignore it may incur a cost—a loss of potential scientific merits that a language machine may have. A lot, then, is at stake here. Let's keep that in mind and turn to the best and most powerful language-related technological inventions.

The Magical Effect of Letters and Numbers

Scientists and mystics have long been fascinated with the magic of letters, words, and texts. In the Middle Ages, it became customary to endow these with numerical properties. Muslim and Jewish numerologists developed mystical numerological methods that assigned values to letters, which was said to unveil “hidden” meanings lying in numerical equivalences between words, based on the value that each letter is assigned by some table. In this system, the Hebrew word *DVAŠ* (דבֿש=honey) is valued at 306, which is also the value of *IŠA* (יִשָּׁא=woman)—a numerical equivalence that supposedly underscores a female quality. (When writing this chapter, I was quite surprised to discover a multiplicity of online calculators for Hebrew numerology.)

In the modern era, a mathematical perspective took somewhat more serious directions. Early twentieth century mathematician Andrey Markov pioneered the first scientific quantitative exploration of language, with a goal that was also related to mysticism: he tried to debunk the notion “free will,” as a way to argue against religion. Markov's theological musings made him launch a quantitative study of vowel sequences in an actual text—Pushkin's novel in verse *Yevgeniy Onegin*. This led to an idea currently known as a Markov chain, a tool for measuring the connection between the probabilities of occurrence of events in a sequence. Markov's invention of a new statistical tool and its application to a literary text constitute the first quantitative linguistic investigation ever.² He thus laid the foundation for much of the current effort to analyze texts, which consists of the application of statistical methods of ever-growing sophistication to questions about language. The torch seems to have then been taken up by George Zipf, who in another foundational study, conducted in the 1930s, counted words in large English texts and rank-ordered them (the most frequent word at the top, then the second most frequent word, etc.). He noticed an

odd regularity: the most frequent word (*the*) occurs about 70,000 times in a million word text, its successor (*of*) occurs half the number of times, the third in line (*and*) has about a third of the frequency of the top word *the*, and so on. This led him to realize that the actual frequency of occurrence of a word in a text is inversely related to its rank relative to other words, which he codified in a principle, now known as Zipf's Law. While there has been little insight about the origin of this Law and while its precise formulation is debated, it is said to be obeyed by many languages, which underscores the importance of quantitative properties of human texts.³ A related effect, concerning frequency and human behavior, was discovered soon thereafter: tasks relating to more frequent words are performed faster than those involving less frequent ones. These two developments helped to aim the limelight at quantitative properties of language materials.

Claude Shannon's eloquent presentation of *information theory* in the late 1940s set the stage for a new era in the quantitative study of language.⁴ This theory is about the transmission of information over a noisy channel. It tries to quantify the degree of certainty of an encoded message being received accurately, given the noisiness of the channel it traverses. If we can successfully calculate the amount of expected error in a transmitted message, we will try to minimize it or will at least be aware of the limited reliability of the channel. For that, Shannon proposed, information needs to be quantified. Over the years, this theory has had immense impacts on a variety of scientific and engineering fields. In the context of human language, the crucial information-theoretic notion is *entropy*, a concept borrowed from thermodynamics. The entropy of a message is the amount of information that the receiver is expected to gain from it. As we move from one symbol in a message to the next, we get a clearer picture about what's coming. Information theory expresses this as a change in entropy. At the start of transmission, entropy is at its peak, with the receiver having no information; at this stage, nothing is expected, and the occurrence of any event would be surprising. Entropy decreases once transmission begins and continues to decrease as it progresses. Figure 4.1 shows this graded process: at the beginning, our ability to guess the next word (in the box) is very limited—entropy is therefore high, as the appearance of most any word would be surprising; when we move on, we become increasingly able to make a reasonable guess about the identity of the word in the box; and at the end of the transmission, the entropy of the message is minimized.

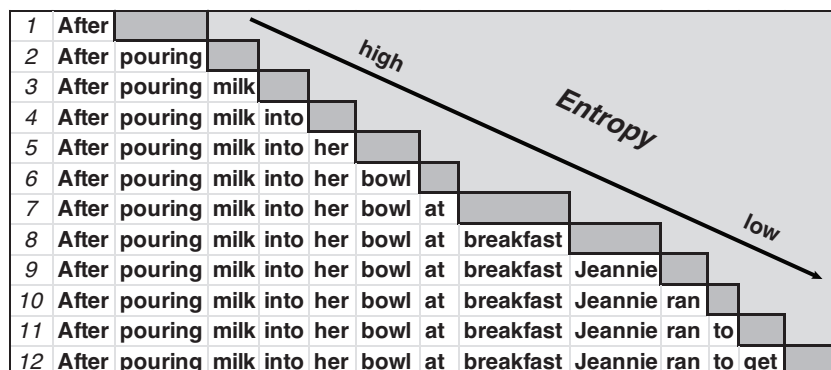


Figure 4.1

Next word prediction: a reduced version of our example “cereal” sentence. As the string becomes longer, the next word becomes less surprising, and easier to predict. Entropy is thus high at the beginning and decreases as the words keep unfolding.

Expected events, whose occurrence would be less surprising, thus have low entropy, while the opposite is true of unexpected events.

The relevance of information theory to communicative acts in language is quite apparent—such acts can be naturally viewed as information exchanges, aimed to reduce uncertainty. Indeed, psychologists of language immediately took notice. Most prominent among them was George Miller, the Harvard psychologist who in the early 1950s pioneered the statistical approach to language and applied information theory to comprehension (see chapter 1). This approach quickly gained prominence. However, soon thereafter, it was criticized by Noam Chomsky, who highlighted its empirical limitations, presenting language phenomena that text statistics were unable to capture. In a remarkable move, George Miller changed his mind (among the first to do so) and quickly became a leading champion of psycholinguistic studies in the new TGG-based framework, which later gained wide acceptance.

The Failure of TGG Implementations

Though TGG was based on a distinction between linguistic *competence* and *performance*, there was a (not unjustified) expectation that it could readily be implemented in a working computer program. Regarding this endeavor, the bottom line is clear: in the engineering battle with information theory, linguistic theory lost badly.

Similar to Frank Rosenblatt's colossal failure to build a working perceptron, the first language processing programs were unsuccessful: a 1950s machine translation project involving engineers, mathematicians, and linguists aimed to automate the translation of large quantities of Russian text—both published and intercepted—into English, initially with the aim of expediting the US intelligence gathering process during the Cold War (indeed, its beginnings were CIA-funded). Machine translation's goal, to build a “fully automatic high-quality translation,” was never reached. A series of failures led to the formation of a special Automatic Language Processing Advisory Committee by the National Science Foundation in 1964. The committee recommended termination, funding was stopped, and the machine translation project was dead.⁵ Its ultimate grave seems to have been dug by Fred Jelinek, a senior language and speech investigator at IBM Research Center, who was rumored to have snapped, “Every time I fire a linguist, the performance of the speech recognizer goes up”—a sentence that keeps being quoted by natural language processing experts. Indeed, later attempts to use symbolic grammars were not particularly successful either.

Later, I will reflect on whether the programmability, or *runnability*, of a linguistic theory is a serious test of its veracity. Yet the failures just described have clearly been held against TGG. A shift was in the making. It was perhaps most pronounced when UPenn's Mitch Marcus, the progenitor of one of the boldest computational implementations of TGG (in his MIT doctoral thesis), became a leading proponent of the statistical approach and the cocreator of the Penn Treebank, potentially the most important database for statistical computational linguistics. This database contains “approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, over 2 million words of text parsed for predicate-argument structure, and 1.6 million words of transcribed spoken text annotated for speech disfluencies.”⁶ The creation of the Penn Treebank laid the foundation for a new generation of statistics-based approaches, which have since prevailed in computer science (though not in linguistics), bringing us to present-day developments.

RNNs as Natural Language Generators and Analyzers

In the previous chapter, the loops in RNNs were equated to a kind of memory, as they enable previous states of the network to participate in the

determination of its current state. Once recurrence is permitted, we can imagine not just one memory but *a multiplicity* of memories: one for the machine's most recent state, one representing the relative contribution of both more recent and less recent states, and so on. In the language domain, the addition of recurrence enabled these programs to relate elements in a sequence to each other during both learning and text generation. Plainly put, if past exposure gets to weigh in, learning is made easier, because it is based on a richer dataset; and text generation becomes possible once the content of your previous state (the identity of the words you've just generated) can assist in optimizing your current state. Indeed, RNNs have therefore featured in text-generating programs, mostly in question answering tasks (a.k.a. chatbots).

How does an RNN carry out its analysis of language input, and what does it require? Input comes in and is evaluated one word at a time, based on the current state of the machine, as determined by its experience with the preceding words in this string (or text). But how is a word represented as digital input for an RNN? For that, *word embedding* is a suitable tool.

What's in a Word?

When you read or hear a completely unfamiliar word, what can you do to reconstruct its meaning? When it appears in isolation, obviously you can do nothing; but when it is flanked by other words, there's a lot of information you may use in order to make an educated guess. This is, in fact, what you do when, in the middle of a conversation, somebody coughs or a car honks and the noise masks a word or two. You try to reconstruct the missing words from the context. Context may be linguistic or situational, but here, we focus on the former. The richer the context is, the more likely you are to reconstruct its missing parts correctly (as a somewhat hearing-impaired person, I do this all the time, with considerable success). From an information-theoretic perspective, the rarer the word, the lower its entropy is and the higher its informativity. Such a word is therefore more difficult to reconstruct. Shannon's idea of quantitatively expressing word informativity thus fits both our intuitions and the need for a useful numerical representation of word meaning.

But how can word informativity be quantified? Based on an idea of linguist John Rupert Firth, "*You shall know a word by the company it keeps*,"⁷

computational linguists invented a system of representation for each word within all its documented contexts (namely, the distribution of other words around it), in the form of a vector in a multidimensional space. The size and dimensionality of this vector, called *word embedding*, may vary depending on the nature of its designated language task. In quantitative terms, we may be able to represent word meaning through the measurement of language use parameters. This is what vector semantics tries to do. This approach quantifies each word's relevance to its context through its absolute frequency, combined with the number of times it appeared in each specific context (of other words to its left and right).

The embeddings of words that we view as similar are expected to be close to one another. Thus words that are close to one another in this vector space keep similar company, up to substitutability: that is, in a context, you may substitute one for the other, with little change in meaning. Measures of proximity are illustrated in figure 4.2, where two-valued vectors represent words along a scale of *empathy* and *intensity*. In this simple representation, the angle between the vectors and their relative length indicate how similar they are. This similarity, a word's relative frequency in the text, and the linear distance between similar words (among other parameters) all express the relevance of a particular word to others and help to establish a context for it.

Needless to say, the graph in figure 4.2 is an extremely simplified version of vector semantics, which as noted is capable of representing vectors in high-dimensional spaces. Dimensionality is determined in part by the

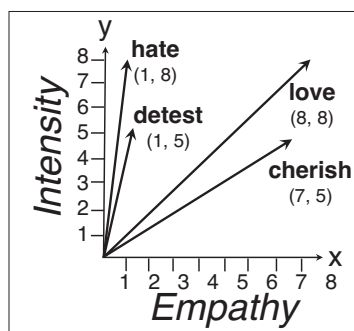


Figure 4.2

Vector-semantic representation of verbs in a two-dimensional space. The vectors for words that are close in meaning are similar.

size of the vocabulary of the training corpus we choose, normally between 10,000 and 50,000 words. But the idea is one and the same: converting meaning—both lexical and contextual—into quantities over which reliable computations can be made in the service of specific tasks. This way, Firth’s oft-cited maxim is materialized.

But not all words are created equal: with grammatical words (*that, if, then, whether, whom, at least, . . .*), embedding becomes much harder. This was realized early on by philosopher Ludwig Wittgenstein, who predated Firth, and in his *Philosophical Investigations* went even further to assert that “*the meaning of a word is its use in the language.*”⁸ The concepts of word embeddings and vector semantics can be thought of as implementations of his ideas as well. But even Wittgenstein qualified his statement and warned that meaning is not always just use: in his discussion of the English inflected copular verb *is*, he suggested that it has one meaning but more than one use. A broader look at the lexicon suggests that the problems actually run deeper. Still, this is how word meaning is represented in current day computational linguistics. Embeddings are the objects over which LLM’s algorithms are computing, supplying the information that reduces entropy as the LLM moves along a string of words. Think again about the case where *cereal* (or *muesli*) appears in the context of a child’s breakfast. When the word *breakfast* appears, the “meaning proximity” between it and *cereal*, hence the high likelihood that the latter would appear can be computed. Word embedding and vector semantics are thus critical to successful next word prediction.

How RNNs Learn with Word Embeddings

Text generation (which has lent it name to the generative AI project) is an important task that RNNs perform. The task is to deliver the next word, which is done with the help of word embeddings. For it to perform well, the RNN requires a word embedding format that is suitable to the task at hand, and a learning phase, materialized though training on a relevant training set.

After proper training as detailed in the previous chapter, the RNN starts processing its first input string, attempting to generate predictions regarding the next word. Its first prediction, of the first word, is not likely to be good: at this stage, entropy is very high, as the only information the RNN has comes from the distribution of words that are in the first position in a

sentence, or more precisely, their ranking according to their likelihood to be in the first position in a sequence (their softmax distribution). We want prediction to improve as the RNN moves on the string, and so entropy has to go down. Recurrence comes to our rescue: input representing the network's state in the previous step is fed to the recurrent loop of the RNN (figure 3.5), which helps the machine to use its previous knowledge in order to narrow down the range of future possibilities. At time t , then, RNNs are capable of combining the word embedding at $t - 1$ into the computation, a capacity that reduces entropy and leads to improved prediction. This process is repeated until the input string reaches its end.

While recurrence is computationally intensive, RNNs are still not powerful enough for certain language tasks, because the memory that they avail a network is not sufficient for the execution of many language tasks. Next, we consider why their performance needs to be enhanced, and how this has been done.

The Need for Memory Enhancement

The discussion so far has looked at input word strings as if they have no grammatical structure, because this is how the neural network approach views language. But the intrasentential dependency relations, which were at the heart of the previous part on language, require a memory which RNNs might not be able to deliver. Salient among these are grammatical agreement relations. Consider these relations in the embedded object relative clause in 1 and the question in 2.

1. The **dogs**_{pl.} ¹*which* ²*the* ³*girl* ⁴*showed* ⁵*her* ⁶*mother*_{Sing.} ⁷*the* **dogs** ⁸*were*_{pl.} ⁹*thin*
2. Which **boy**_{Sing.} ¹*did* ²*John* ³*tell* ⁴*the* ⁵*guards*_{pl.} ⁶*which* **boy** ⁷*was*_{Sing.} ⁸*waiting* ⁹*outside*

In 1, the noun phrase that is closest to the plural verb **were** is singular (*her mother*); this verb agrees with the plural **dogs** that is six words away (but its original *struckthrough* position is adjacent to **were**); in 2, the noun phrase closest to the singular **was** is plural (*the guards*); this verb agrees with the singular **boy** that is five words away (but its original *struckthrough* position is adjacent to **was**). How do these elements agree at a distance? Linguistic theory posits a MOVE relation seems to have separated the overt (heard) noun

phrase and its verb. Agreement is said to hold between the original (silent, hence *struckthrough*) position of this noun phrase and the verb (perforated line). This agreement is “inherited” by the moved noun phrase (full line).

These are frequently used sentence types, whose grammatical agreement is complex. Indeed, RNNs often fail to generate the correct forms.⁹ Interestingly, much of the natural language processing literature on this matter refrains from describing these sentences in grammatical terms, sticking to the measure of linear distance between codependent positions within the string. I follow suit, and I note that from the natural language processing perspective the number of intervening words in 1 and 2 is indeed large and that the absence of grammatical analysis makes the challenge to the network even greater. As seen in 3, the number of intervening words within an intrasentential dependency relation can be even greater, in fact, almost arbitrarily increased.

3. The **boys** *who*¹ *I*² *introduced*³ *to*⁴ *the*⁵ *girl*⁶ *with*⁷ *the*⁸ *dragon*⁹ *tattoo*¹⁰ **were/*was** tall.

RNNs tend to focus on connections that are local, which makes correct prediction in such instances difficult, often unsuccessful. Network engineers, realizing this deficiency and being reluctant to consider the addition of grammatical dimensions into the representation, set themselves to find new solutions.

From RNNs to Long Short-Term Memory

The computational burden of enhanced recurrence is high, and when the distance between linked serial positions in a string was large, performance failures followed. A solution was proposed, to make the memory required for the computations of long-distance dependencies more efficient at expunging or at least fading out irrelevant information, while dynamically highlighting relevant elements. This device is long short-term memory (LSTM).

Instead of performing heavy computations blindly, LSTM can be thought of as managing the context through a new “memory of context” layer and the continuous assignment of new “relevance” weights to the input (through a component called an “input gate”), such that at time $t - 1$, words that are

deemed to have a lesser contribution (relative to others) to the current state at time t , receive low weights by the “forget gate” (or even zero, in which case they are “flushed out”), while those with a larger contribution receive a greater weight (through an “output gate”). The first and last of these three components constitute the long-term aspects of the system: the input gate decides what information from state $t-1$ would be stored, and the output gate decides what information to be passed on as output. The “forget” aspect is the short-term one, hence the (intentionally oxymoronic) label *long short term memory*. LSTM gets fed with the previous state’s contextual information C_{t-1} , as well as that in the current state X_t .

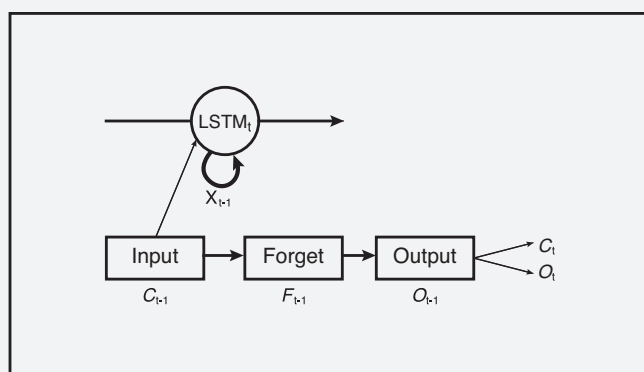


Figure 4.3

LSTM architecture that consists of input, output and forget “gates.”

The decisions LSTM makes are always statistical in nature: as it is trained by the backpropagation algorithm, its weights are adjusted to identify connections in a string, such that frequent connections between nonadjacent words are highly weighted and less frequent ones can be flushed out. The result is a representation of the importance of each dependency between serial positions, which assists the network in focusing on the more frequent, hence more important, dependencies.

So where are we now? The advantage of RNNs over feedforward networks was a recurrent loop in hidden units; LSTM augmented these RNNs with a new device, namely a memory system that manages the context according to its relevance (as measured by the similarity of the embeddings of past words to the current word) and feeds the result to the output layer. Note that once again, linear distance, not structure, is what matters for

LSTM: the context is determined by cooccurrence patterns of word pairs in different serial positions in a string. Moreover, as LSTM moves along a word string, recent inputs are more relevant to the upcoming word than more distant ones. Still, the resulting memory, together with a significant increase of training data, takes us to a new level of performance in language processing, which comes with an enhanced capacity to process serial data.

Attention and Transformers: Handling Dependency Relations

We started off by talking about the distributional perspective on meaning, by which a word is to be recognized “by the company it keeps.” We also discussed how word embeddings implement this idea. We then proceeded to problems that RNNs encounter in handling intrasentential dependency relations, once the codependent words are distant from one another in the string. When these problems were recognized, effort focused on increasing networks’ ability to handle contextual connections properly, while reducing computation time, through the removal of costly recurrent processes. This resulted in an additional burden on memory systems, which increased in size, but enhanced the range of linguistic problems that were solved, and also increased networks’ training capacity. These moves cleared the path for LLMs.

What is context? I have been using this word as a vague cover term for a variety of relations between positions in a string (that can be smaller than, equal to, or larger than a sentence). Earlier, we saw some types of dependency relations within a sentence (MOVE, pronoun-antecedent, and agreement dependencies), but there are others, and complex relations between sentences also exist. Identifying and representing these must be at the heart of any successful model for language. While linguists analyze dependency types one by one, LLM designers proposed a solution in one big sweep: a new network architecture that consists of two parts, a receiving input end (“encoder”) and a transmitting output end (“decoder”). Initially designed for machine translation tasks (also known as *seq2seq* tasks) by aligning, for example, pairs of sequences from different languages, it is even capable of aligning pairs of strings that are not of equal length, which is a must in translation: expressions in different languages may mean the same thing (hence be good input–output machine translation pairs), but each may contain a different number of words, which makes their alignment and ultimately translation difficult. This enhanced encoder-decoder architecture facilitates the network’s alignment task, making the establishment

of dependencies easier. These capacities can be used for the establishment of dependency relations in a string, especially with the help of *attention*, a dynamic memory. Attention can represent extremely long strings, and dynamically highlight their more important aspects (similar to LSTM), so as to indicate which positions in the string should be codependent. This machine, moreover, encodes not only the text that precedes a word but also a sizeable part of the text that follows it. The result is a bidirectional machine, the *transformer*.

Let's look a bit into how a transformer processes data: the entire input string is compressed into a single vector of fixed length. This encoder vector is combined with the output of attention: for each word w in the string, an attention mechanism encodes all the words around w across the entire context (which in current LLMs can be as long as 2^{12} tokens, or several thousand English words). At each w position, the attention mechanism evaluates all the elements in the string along three relevance dimensions (called *query*, *key*, and *value*); the obtained values constitute an *attention score*. This score then combines with the initial string vector to produce a matrix representation, which can be thought of as a profile such that any word w in this string, the score highlights all those elements in the string that are of relevance to w . The profile of the position of w in the string can then be compared to that of other positions, allowing the detection of dependency relations within the string. This massive process leads to a large number of attention scores, which are now rank-ordered (by their attention score). A high score at a given position can help to detect a special relation between it and other positions. For example, in figure 4.4, each of the words *robber* and *policeman* maintains a number relation with a verb (solid line) and a gender relation with a pronoun (dotted line). These pairings of positions are therefore salient due to their high attention scores, as indicated in the figure.¹⁰

Transformers lack RNNs (no recurrence), which makes them more efficient, but they still require massive computation, as they link up all position pairs in a very long string. The combined information for the whole string is fed into the last part of the machine, the decoder. Still, this intense effort seems to pay off, as the encoder-decoder+attention architecture greatly enhances the performance of transformers. It fixes a deficiency that was detected in the functioning of LSTM, whose *modus operandi* gives precedence and saliency to recent inputs: as its most recent state encompasses a string whose latest element is the current one, a "bottleneck" is created, in which the most recent context is overrepresented, because its relevance is based on its more recent appearance than the earlier parts. The overrepresentation-of-the-recent problem has brought about performance errors. In the transformer, attention successfully ignores precedence relations of this type.¹¹

(continued)

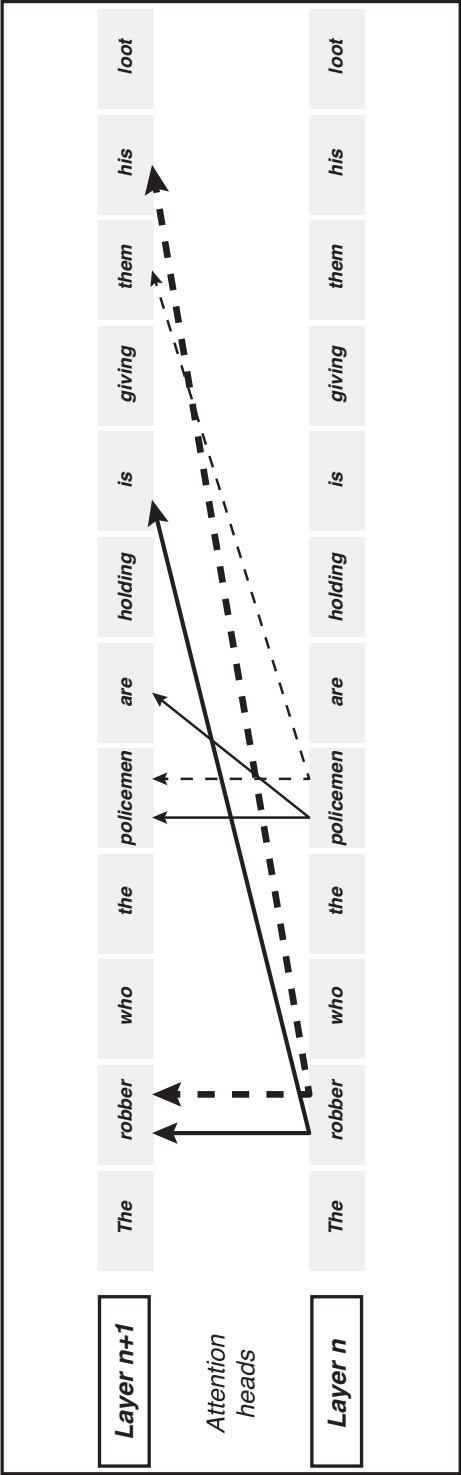


Figure 4.4
The connection that attention heads establish in a sentence that contains two agreement dependency relations.

A flagship language model using the transformer architecture has been BERT (Bidirectional Encoder Representations from Transformers)—an LLM with 110 million parameters in one version, 340 million parameters in another version.¹² Its learning phase (which I did not discuss) trains it to predict the next word. Its capacity can be scaled up by training on billions of words, affording it with more power.

Indeed, BERT's performance (as well as its offspring's) is stellar, which is what has made it extremely popular and has led to claims that it is an actual model of human language processing. This is the claim we will explore in detail in the next chapter—whether a running system with compellingly human-like performance and multiple applications for a rich variety of tasks is to be considered a scientific theory; if so, how we can know it; and how we can consider whether it meets fairly standard criteria that a valid theory of human language must satisfy.

The Interpretability of LLMs

The rapid increase in the size of neural networks has advanced their problem solving abilities but created new problems. The obvious increase in materials and energy that such gigantic systems consume, conjoined with the harmful environmental effects of this increase, has recently given rise to calls for more “frugal computing,” which will likely not be heeded anytime soon.¹³ In the present context, growth seems to have caused two problems. The first problem (to be discussed in the next chapter) is a nearly complete lack of constraints on LLMs' descriptive power, which sends them in a direction that is diametrically opposed to that of an explanatory theoretical science. A second problem is that our vision is blurred by opaqueness, which is due to the size of networks, their nonlinearity, the tens of billions of parameters each has (a number that's growing), and their massive connectivity. No human can conceivably approach these machines analytically, whether manually or through an interpretation algorithm. The resulting products are working machines and a lack of understanding of how and why they do what they do so well.

The problem of interpretability manifests as “the inability to effectively interpret these models,” which “has debilitated their use in high-stakes applications such as medicine and raised issues related to regulatory pressure, safety, and alignment.”¹⁴ We thus end this chapter on a sweet-and-sour

note, to the effect that while LLMs' operation is excellent, they are mystery black boxes. This lowers even more their chances of becoming scientific theories.

Taking Stock

This chapter updated you on present-day neural networks. We saw:

- The development of statistical methods in the analysis of texts
- The shift from structural to statistical approaches in natural language processing
- Tasks that LLMs are supposed to carry out
- RNNs
- Word embeddings and LSTM
- Attention and transformers
- The problem of interpretability

This is a section of [doi:10.7551/mitpress/15932.001.0001](https://doi.org/10.7551/mitpress/15932.001.0001)

How Deeply Human Is Language?

Chomsky, the Brain, and the AI Fantasy

By: Yosef Grodzinsky

Citation:

How Deeply Human Is Language?: Chomsky, the Brain, and the AI Fantasy

By: Yosef Grodzinsky

DOI: 10.7551/mitpress/15932.001.0001

ISBN (electronic): 9780262052023

Publisher: The MIT Press

Published: 2026



The MIT Press

The MIT Press
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
mitpress.mit.edu

© 2026 Massachusetts Institute of Technology

All rights reserved. No part of this book may be used to train artificial intelligence systems or reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data is available.

ISBN: 978-0-262-05200-9

EU Authorised Representative: Easy Access System Europe, Mustamäe tee 50, 10621 Tallinn, Estonia | Email: gpsr.requests@easproject.com