Language Models in Plain English
Humans Understanding How Machines Understand Language

Austin Eovito & Marina Danilevsky

REPORT
The world is going hybrid with IBM.

Today, your business needs to keep things moving—and be prepared for what’s next. With IBM, you can do both. By bringing your data together across clouds, you can modernize applications, automate IT processes, predict and manage operational issues, and help keep private data secured. That’s why so many businesses are working with IBM.

ibm.com
Language Models in Plain English
Humans Understanding How Machines Understand Language

Austin Eovito and Marina Danilevsky
# Table of Contents

1. **Introduction: What Is It like to Be a Language Model?** ............ 1  
   Why Do You Need to Read This Report? .......................... 2  
   What Is a Language Model? ....................................... 4  
   Road Map of the Rest of the Report ............................. 12

2. **Meet the Neural Model Family.** .................................. 13  
   What Do Humans Want to Remember? ............................ 14  
   Neural Networks for Language Modeling ......................... 17  
   Considerations on the Use of RNNs .............................. 28  
   Key Takeaways ...................................................... 29

3. **Two Heads Are Better than One: Encoder-Decoder Architecture.** 31  
   How Do Humans Summarize? ...................................... 31  
   Encoder-Decoders for Language Modeling ....................... 33  
   Considerations on the Use of Encoder-Decoder Architecture 35  
   Key Takeaways ...................................................... 36

4. **Choosing What to Care About: Attention and Transformers.** ..... 39  
   Communicating Across Languages ................................ 39  
   Attention for Language Modeling: Concentrating on Only the Relevant Input ............................................. 42  
   Transformer Architecture for Language Modeling ................ 44  
   Considerations on the Use of Transformer Architecture ........ 48  
   Key Takeaways ...................................................... 49
What is it like to be a bat?

The philosopher Thomas Nagel asks this question in his 1974 essay on consciousness.¹ His position is that the answer is unknowable. If I imagine that I have webbed arms and poor vision, perceive the world by sonar, subsist on a diet of insects, and spend the day hanging upside down, “it tells me only what it would be like for me to behave as a bat behaves.” But if I try to imagine what it’s like for a bat to be a bat, my restrictions to the limited range of my own mind and experiences render this impossible.

Humans and bats, at least at the time of this writing, have no shared language. On the other hand, countless AI models exist in the world—many of which have been created specifically to communicate something to us in our own language.

The recent explosion in advances in machine learning has brought a myriad of interesting, powerful, and increasingly opaque models. Simultaneously, the recent movement toward democratization of AI has lowered the barriers for being a data scientist and using machine learning models. It is simple to deploy a model in the real world without being concerned about explaining the output or without

---

exploring the ethical implications of decisions that will be made on the basis of that output. The ease of creating and using machine learning models is going up; the ease of understanding what machine learning models are doing is going down (Figure 1-1).

![Figure 1-1. Recent advancements in the field of machine learning have meant that creating and running models is becoming much easier, while simultaneously, understanding what models are doing is becoming harder.](image)

**Why Do You Need to Read This Report?**

We use a myriad of technological advances every day without wanting or needing to understand the details of how they work. You may not know exactly how your car, your toaster, or even your house was put together, but you have at least a general intuition of motion, electricity, and architecture. You (mostly) do not anthropomorphize them to the extent of accusing your toaster of being biased toward burning your bread, or suspecting your car’s air conditioner of deliberately refusing to work on especially hot days. When you observe such effects, you know to change your toaster settings or to take your car in for a checkup.

The human race is in a much earlier stage of its relationship with AI than it is with cars and appliances. We use AI every day without wanting or needing to understand the details of how it works—search, communication, vision, automation, the list goes on.
However, we have not yet settled into a comfortable intuition of the general functionality. Unlike with toasters, we struggle to keep from anthropomorphizing AI—and language models (LM) are one of the more challenging examples of this, as language output is made to be interpreted.

This report is not going to teach, or even get into, how to run and deploy language models. Many wonderful practical resources are available for this, at various levels of granularity. You may have never personally run a language model (or a system with a language model component). Alternatively, you may be creating, training, and running language models on a regular basis, or consuming them in downstream applications. Wherever you fall in that range, we suppose the following:

- You have at least some surface familiarity with probability, matrices, and machine learning tasks such as classification, summarization, and translation (you are not reading these terms for the first time here). You likewise have at least heard of language models, though you don’t need to be clear on what exactly they are or how they work.
- You want to be able to take a step back from the logistical details and have a thoughtful conversation with the various stakeholders in your business about high-level language model concepts (as the primary focus of this report is on neural network language models).
- You want to have the same comfortable handle on the models you come in contact with as you do with toasters; you need to know that electricity heats the filaments and the generated heat cooks the bread, but not the atomic-level physics and chemistry involved in either process.
- And finally, you want to have reasonable confidence in how to think critically about applying language models to your business. (You can toast a waffle but not a slice of cheese—or rather, you technically can, but you will have a mess on your hands!)

---

2 Here’s one good guide to NLP systems in a business setting: Practical Natural Language Processing by Sowmya Vajjala et al. (O’Reilly).
Let’s begin by turning a critical eye to what a language model is, what it does, and how it approaches language.

**What Is a Language Model?**

Language models are at the root of many of our experiences with technology. Among many, many examples, you are interacting with a language model when:

- You read the summary of a large set of product reviews.
- You ping the HR chatbot to find out this year’s company holidays.
- Your phone offers weirdly prescient next-word suggestions.
- And for us, when we enter “language” in our Google search box and the top autocomplete suggestion is “language model”—because we have been working on this report!

A *language model* is a technique for calculating the probability of a particular sequence of words occurring. It tries to emulate certain human linguistic capabilities, learning a myriad of associations between words to represent, depending on the task at hand, your own language patterns, the patterns of a set of other people, the patterns of English in general, and so on.

But remember, a human will not know what it is like to be a bat through imitating bat behavior. Likewise, an LM will not know what it is like to be a human through imitating human interaction with language. An LM learns only what it would be like if it behaved like a human, within the model’s capabilities. And since this report is written for a human audience (not for LMs), we will spend the rest of the report providing some perspective on how humans should think about what an LM is doing when it is trying to behave like a human.

**What Does a Language Model Do?**

So what do LMs do? Consider a simple game between two people, popular in improv comedy. One person begins by throwing out a single word, the second person gives the next word to follow, the first person gives the third word, and so on: “I,” “went,” “to,” “the,” “mall,” “and,” “found,” “a,” “stupendous,” “llama,” “that,” “was,” “purple.” The object is often to see who is first to laugh or to find
themselves unable to continue the game. The direction the generated story takes depends on the background knowledge of the players, their previous experience with the game, their goals (to keep going for as long as possible? To cause the other person to laugh?), and many other subtle and variable aspects.

In the simplest terms, each time a trained language model is invoked, it’s playing the next round of this game. The prompt may be something like a part of a sentence to be continued (“I went to the”), or something to be translated (speech or text in a different language). The object is to output the best possible word(s) for the given prompt, and what the model chooses as the best output depends on how it gets trained.

The model is trained by playing the game of “predict the next word,” where all the best guesses are already known (the sentences or translations already exist). In each round, the model competes with itself, trying to guess the next word correctly, and tweaking its point of view to get closer to the existing text.

A language model (LM) is a technique for calculating the probability of a particular sequence of words occurring. In plain terms, the model is always playing the game of “predict the next word.”

**Are Language Models like Humans?**

To understand how an LM approaches language, let’s first consider how people think about language.

A fluent English speaker expects to come across “red book” rather than “book red,” and “hot dog” rather than “cold mouse,” and “early bird” rather than “late bird.” When asked why this is, a human may possibly be able to articulate the reasons for these examples: in English, adjectives usually precede nouns, and a “hot dog” is a phrase describing a popular food and not just a synonym for “overheated canine,” and an “early bird” is a description of early risers or early arrivers, as well as part of a well-known proverb. However, a human’s simplest answer would be that you “just see” the common phrases, and “just don’t see” the uncommon ones. This hodgepodge of human responses draws on the two components of language: structure and meaning.
However, LMs are aware of only structure. They are entirely focused on outputting something grammatically correct—without the notion that such a thing as grammar exists. A good language model (of English) will also correctly rank “red book” as more likely than “book red,” and so on, because it will have “read” enough English text to learn what is common and expected.

But the model can never know what “hot dog” actually means. For that matter, the model does not know what “hot” and “dog” mean separately, nor is it able to grasp the concept of words as having meaning. In the same way, an LM cannot intentionally tell lies, obfuscate facts, or spare your feelings. When we encounter language, we experience the illusion of meaning because we must. LMs do not because they cannot.

Language models don’t make judgments; they make predictions. Language models do not mean what they say. Language models generate well-formed language, and humans experience it as an illusion of meaning.

How Does a Language Model Learn?

Language models (and AI models more generally) are constructed through the framing of “becoming as good as possible” at a specific task. An LM “understands” a task only in terms of the content encountered during training for it: for input that is this, the output is that. To further generalize the statement: for input that is like this, desired output is like that.

The representation of all the necessary and sufficient information to calculate like is usually referred to as the feature space. Perfecting the calculation for weighing this information in preparation for unseen input is the process of learning, or training. This discussion is a simplification, but it applies to everything else discussed in this report as well as to artificial intelligence more broadly.

---

3 “Read” is in quotes to emphasize that language models don’t read in the same way humans do.
The process of creating a trained model is the journey from for input that is this, the output is that to for input that is like this, desired output is like that by representing all the necessary and sufficient information to calculate like.

As we described, the most straightforward task for an LM is learning to predict the next word: for an input sequence of $K$ words, the output is the word $W$ that the language model deems the most likely to come next. The model is exposed to training data comprising many input and output examples. (Examples just from the preceding sentence include the input “model” with the output “is,” the input “exposed to” with the output “training,” and so on.) For more advanced LMs, the input is more complicated than the few words preceding the output, incorporating additional contextual information from the rest of the sentence or from even further away.

During this learning process, the model constructs its representation to “understand” the language of the training data, and adjusts that representation until it comes as close as it can to matching the examples in the training data. Think of your own experience training for a test by using flash cards, with a prompt on one side and the answer on the other. You check your mastery of the material by how closely your answer to a prompt matches the other side of the card. The language model is learning from many (many!) such flash cards. But what exactly is this representation that is being learned?

---

4 LM inputs and outputs, of course, can include punctuation.

5 For a large language model, many training examples are necessary.
How Does an LM Represent Language?

Language models use three general approaches to encode the information that represents language:

Linguistic

The model manually encodes the grammar structures, which is difficult and time-consuming to do, but is also explicit and fully explainable. *The model represents language as a set of rules.* One such rule for English would be that an adjective usually comes before the noun it is modifying (“red book,” not “book red”), though there are exceptions, such as when used with some verbs (“the cake tastes great,” not “the great cake tastes” or “the cake great tastes”).

Probabilistic/statistical

The model, using a reference text, counts word occurrences, and relies on those counts when playing “predict the next word.” As an overly simple example, a bigram (two-word) language model counts five instances of the word “red” in the reference text, of which two are “red book” and three are “red shoes.” When asked to predict the next word after a prompt of “red,” the model will answer “book” with two-fifths probability and “shoes” with three-fifths probability. An LM can learn these probabilities for a word sequence of any size (one word, two words, three words, etc.). Additional statistical tricks can address issues such as dealing with previously unseen sequences. *The model represents language as a set of word sequences and their associated probabilities.* Compared to modern neural network language models, statistical language models are both far better understood and, at the current time, waning in popularity.

Embeddings

The model represents every word in the language as a vector in a large dimensional space. This is the representation primarily used by deep neural network language models.
The third approach, embeddings, is both the most difficult one to understand and the one used by most state-of-the-art language models, so we will now spend a little extra time with it.

Consider the following (well-known) analogy: man:woman::king:?  
Or, “man” is to “woman” as “king” is to what word? With reasonable fluency in English, you will have little difficulty coming up with the answer of “queen.” You might explain your reasoning in a couple of ways:

- The definition of “king” is (simplistically) “royal man.” What word is defined as “royal woman”? The answer is “queen.”
- The difference between “man” and “woman” is a change in gender from “male” to “female” (again, simplistically, and fully acknowledging the existence of nuances we are ignoring for the sake of the example). What do you get when you take “king” and change its gender from “male” to “female”? The answer is “queen.”

In your mind, you have a concept of the difference between “man” and “woman” as being like the difference between “king” and “queen”; similarly, “man” and “king” are different, like “woman” and “queen” are different. We have words to represent these differences: “gender” and “royalty.” Furthermore, adding any other words does not change the relationships: a “tall man” is like a “tall king” in the exact same way that a “tall woman” is like a “tall queen”—that is to say, the difference between them is still only the concept of “royalty.”

Taking a critical eye to the previous sentences, it is clear that we as humans define all these words in a circular way, relative to each other (all words are, of course, defined by other words). We hold some representation of these words in our mind, and when probed, use other words to describe the meaning of that representation—the concepts of royalty, gender, and so on.

---

6 For the academic origins of the analogy, see “Efficient Estimation of Word Representations in Vector Space” by Tomas Mikolov et al. If you’re interested in further academic digging on word analogies, see “Word Embeddings, Analogies, and Machine Learning: Beyond King - Man + Woman = Queen” by Aleksandr Drozd et al.
LMs, as we have discussed, do not grasp the idea of meaning. But in the space of word embeddings, they have a crisp and quantifiable idea of differences and of objects being like other objects:

A word
A point in a multidimensional embedding space, with specific coordinates

A difference between two words
The distance between their coordinates, represented by a vector (which has a size and direction)

Determining to what extent a relationship between one pair of words (“man” → “woman”) is like the one between a second pair of words (“king” → “queen”) is equivalent to calculating how similar the differences (size and direction of movement in the embedding space) are between the words of the first pair and the words of the second.

Figure 1-2 illustrates how an LM would see the four words of our analogy as points in an embedding space.

Figure 1-2. A simplified representation of the four words “man,” “woman,” “king,” and “queen” in an LM embedding space, with the dotted lines representing the differences between them. A real LM has far more than three dimensions (this is just impossible to visualize in a helpful way).
We can now loosely translate our human rationales into two potential paths to take to arrive at the answer to the analogy:

- Start at “king.” Move to “man.” Move to “woman.” Move the exact same distance, but in the opposite direction, as when you moved from “king” to “man.” What word is closest to the point at which you have ended up? The answer is “queen.”
- Start at “woman.” Move to “man.” Move to “king.” Move the exact same distance, but in the opposite direction, as when you moved from “woman” to “man.” What word is closest to the point at which you have ended up? The answer is “queen.”

The dimensions in the $n$-dimensional space of word embeddings do not have any inherent meaning. Similarly, the model cannot put any meaningful interpretation on the distance between word coordinates.

A few final words on embeddings. It should come as no surprise that the real world is messier than our simplified example with four words. The word embeddings that an LM learns are based on the text it is trained on, and the resulting encoded relationships will include, sometimes in strange and hard-to-track ways, all the biases and idiosyncrasies of that text (recall that LMs make predictions, not judgments).7 We further discuss these extremely important considerations toward the end of the report.

In addition, many words have multiple meanings, often encompassing multiple parts of speech, such as both verb and noun,8 rendering their vector representations potentially ambiguous and certainly not as clean as in the preceding example. And the dimensions of that embedding space do not have any inherent meaning (remember, LMs work only with structure, not meaning). Word descriptions of this purely mathematical space have the same effect as word descriptions of being a bat—we can discuss hanging upside down, but we're really talking about a whole different animal.

---

7 This is especially true when learning on many examples.
8 For just a few examples, words that can be either a verb or a noun include “run,” “walk,” “bat,” and “throw.”
Road Map of the Rest of the Report

As we’ve mentioned, LMs use three general approaches: (1) manual, based on linguistics; (2) statistical, based on probabilities; and (3) neural, based on embeddings. This report focuses on the last of these. The current state-of-the-art LMs are either entirely composed of neural networks or have a significant neural architecture component. At the same time, neural network–based LMs are far more difficult to get a comfortable sense of, compared to manual or statistical LMs. Our goal for this report is for you to gain this intuition without getting tangled in the mathematics of the inner workings of these models.

In the following chapters, we explain how neural LMs are used in completing a few of the most common and important text tasks, including text summarization, translation, and reading comprehension. We discuss the practical differences between how humans and LMs approach these tasks. LMs are seeking to emulate the behavior of a human who is generating language, in the best way they can, given how they are constructed. Therefore, throughout this report, we compare how humans and LMs approach these tasks, so that we, as humans, can better understand how machines understand language. The rest of the report is organized as follows:

- First, we dive deeper into the basic task of an LM, to predict the next word, and use it as a lens to understand neural network language—specifically, Recurrent Neural Networks and their close relatives.
- We build on this to consider the task of abstractive text summarization as a lens to understand Encoder-Decoder architecture (also referred to as Sequence-to-Sequence).
- Then, we discuss machine translation and use it as a lens to understand the attention mechanism and Transformer architecture.
- Finally, we’ll gather the knowledge gained from the previous sections to explore the current state of machine language understanding, and focus on what LMs are good at as well as on their risks and weaknesses.
If you have read about neural networks, you are probably aware that they are so called because their general structure is inspired by the observed behavior of biological neurons. However, this is not the same as saying that a neural network constructed to perform machine learning tasks works in the same way as a human brain.

An artificial neural network is intended to mimic certain things the human brain does. But in the same way that eating bugs and using sonar cannot let a human know what it is like to be a bat, learning to predict the next word does not allow an LM to know what it is like to be a human. This goes both ways: the fact that our behavior inspires neural network architecture, and there is convergence in the observed outcome (a word is, indeed, predicted), does not mean that we know what it is like to be an LM.

We therefore introduce neural network language models, and all subsequent techniques in this report, by first making critical observations of how humans approach language, memory, and communication. It is important to develop good intuition about the basic building blocks of a neural network language model so that we may take on the increasingly complex models that are coming to the forefront of language modeling.

To communicate these underlying concepts, we will begin with the very human process of learning to bake through trial and error. In our upcoming example, we make certain assumptions to better understand the act of learning, rather than the underlying process that defines learning. We highlight some concrete lessons to
motivate the technical introduction to neural networks (with more bats!), with a focus on getting a handle specifically on Recurrent Neural Networks and their variations.

**What Do Humans Want to Remember?**

An essential aspect of what it is to be human is to eat. Eating is an act of necessity at worst and an act of pleasure at best. Humans, as we would have it, have passed down information for thousands of years through the use of symbols and language. This information persisted orally for some time and ultimately was written down. We now have written records communicating information from hundreds to thousands of years ago about all kinds of things, but for brevity, let’s focus on cooking.¹

In this scenario, we imagine ourselves in our most basic form: we have limited capacity to hold information in memory, we lack experience, and we are unable to embed outside information into our knowledge base. In this limited form, we must find a way of gaining experience and carrying it forward to accomplish our goal, and to increase our ability to accomplish this goal well. To explore the methods of learning in this limited state, we’ll follow a recipe. To reiterate, in our current, abstracted form, we have limited memory and limited ability to embed additional information (we know only what we can read), and we need to gain experience. Here is our recipe and ingredient list:

**Ingredients:**
- 1 cup white sugar
- ½ cup butter
- 2 eggs
- 2 teaspoons vanilla extract
- 1 ½ cups all-purpose flour
- 1 ¾ teaspoons baking powder
- ½ cup milk

---

¹ I hope I’m not alone in confessing that I have definitely accidentally set things on fire, misread recipes, and engaged in poorly thought-out ingredient substitutions.
Instructions:

0. Grab ingredients.²

1. Preheat the oven to 350°F (175°C). Grease and flour a 9×9-inch pan or line a muffin pan with paper liners.

2. In a medium bowl, cream together the sugar and butter.

3. Beat in the eggs, one at a time; then stir in the vanilla.

4. Combine flour and baking powder, add it to the creamed mixture, and mix well.

5. Finally, stir in the milk until the batter is smooth. Pour or spoon the batter into the prepared pan.

6. Bake for 30 to 40 minutes in the preheated oven. For cupcakes, bake for 20 to 25 minutes. Cake is done when it springs back to the touch.

Although we don’t know it yet, we will end up making multiple attempts, so we’ll refer to each one numerically, starting with Cake 0.

For Cake 0, you begin reading the instructions. You see a total of seven steps, in sequence. You start at step 0, as it appears first. You grab all the ingredients and read the directions and think, “Who needs these? It all ends up in the bowl anyway.” So besides grabbing the ingredients (step 0), you think that you can just do what you want in the order that suits you best.

You think that combining the flour and baking powder (step 4) sounds easiest. You dump them together and go to grab the creamed mixture. To your surprise, you cannot seem to find it. So you read the step before step 4: “3. Beat in the eggs, one at a time; then stir in the vanilla.” You do this, and add it to your flour and baking powder. You think you’re about finished when you notice you have unused ingredients—namely, the sugar, butter, and milk. You decide to just add them all in at once and stir.

You have a final mixture that looks fine and go to put it in the oven. It is not hot, so you read the instructions again, seeing that step 1 tells you to preheat the oven. Furious, you wonder why the instructions didn’t say “Instructions are in order” as step 0! You wait a

² We intentionally added this step zero, as it was not explicitly mentioned and, technically, based on what we said, if this wasn’t here, we’d heat up the oven and be done…
couple of hours for your cake to finally be done, and it sucks. You tell yourself, fine, next time I will follow the instructions in order.

For Cake 1, you decide that you will follow the instructions in explicit order. You do step 0, step 1, step 2, and step 3. You get to step 4: “Combine flour and baking powder, add it to the creamed mixture, and mix well.” You look around for the creamed mixture and still cannot find it. Livid that the instructions lied again, you add the flour and baking powder to the beaten mixture. It’s all the same anyway.

You complete step 4 based on your success from the previous step (without the darned creamed mixture), but at step 5 (“Finally, stir in the milk until the batter is smooth. Pour or spoon the batter into the prepared pan.”), you can’t find the “prepared pan,” and what the heck is the “batter”? You grab your “9×9–inch pan” and pour in everything you’ve been mixing in the bowl. You place it into your “preheated” oven, and 30 minutes later you get a delicious cake!

By following the steps in order, you successfully made the cake! You also conclude that the “beaten mixture” must be the “creamed mixture,” the “stuff in the bowl” is the “batter,” and the “9×9–inch pan” is the same as the “prepared pan.” You feel like there is nothing left to learn now that you can successfully make the cake, so you try to commit everything you need to know to memory. However, some mistakes keep popping up in your head every now and then, reminding you that you didn’t know what things were called, and you wish you knew what was actually important to remember and what wasn’t (who cares that by step 5 it’s called “batter”?). So let’s learn how to forget.

Now, for Cake 2. You decide to make the cake again a week later but, unfortunately, you lost everything but the ingredient list. You try to recall the entire recipe but can remember only three things:

1. There is an order to the recipe that begins with preheating the oven to 350°F and getting a greased pan.
2. The wet ingredients should be mixed before the dry ingredients, finally ending up with the “batter.”
3. The batter goes into the oven until it turns into a cake.

You grab your ingredients and proceed to add the sugar and butter together, then add the eggs and vanilla, then the milk, and finally the
flour and baking powder, producing the “batter” (you ideally should have added the milk last, but this does not cause undue problems). You add the batter to the pan you greased at the beginning, and bake it for 30 minutes. Upon opening the oven, you are greeted by the delicious smell of your cake.

No longer are you limited by the instructions! All you had to remember was the order of a few key aspects, the combination of ingredients, and that by mixing them together, you get batter that makes a cake! You realize that you didn't need to remember the entire recipe; you could use an underlying pattern in combination with the ingredient list to make a cake! With your newfound baking skills, you can have your cake and eat it too.

**Do Machines Dream of Electric Cake?**

As you sit and eat your cake, you reminisce on your baking experience and ponder what it is that you have learned. You see that you can learn information in terms of order over time, because sequences have patterns. You have also learned that relationships between things can remain set over time and that you can retain some pieces of information and forget other pieces with little consequence if you understand the underlying pattern.³

It is now that you think, what if you could teach a machine to make a cake? Wouldn't it need to know only the abstract pattern of baking that you followed? You guess the real question is, how would you even begin to teach a machine how to learn that behavior?

**Neural Networks for Language Modeling**

Let's use this sentence from the US Department of the Interior’s “13 Awesome Facts About Bats” detailing various sizes of bats as our running example:

> Bats range in size from the Kitti’s hog-nosed bat (also called the bumblebee bat) that weighs less than a penny—making it the world’s smallest mammal—to the flying foxes, which can have a wingspan of up to 6 feet.

³ Apologies to science-fiction author Philip K. Dick, who wrote *Do Androids Dream of Electric Sheep?* for the section heading.
Imagine that you have done a lot of reading on this interesting mammal and are going to play the game of predicting the next word. Given the prompt of “Bats range in,” you will think of several reasonable words to follow—“length,” “weight,” “size,” “color,” and so on—and you will choose one of them. We will spend the rest of this chapter discussing how a neural network would mimic this outcome and choose a word to follow a given sequence.

As a reference, recall the mention of probabilistic language models, briefly described in “How Does an LM Represent Language?” on page 8. A trigram model would create a representation for the non-zero probabilities of every word that might follow the phrase “Bats range in” (as well as other trigrams) and output the most likely one, as shown in Figure 2-1.

![Figure 2-1. Running a trained probabilistic trigram language model.](image)

So, how does a neural network predict the next word?
Predicting the Next Word with Neural Networks

We are not going to walk back all the way to introducing artificial neural networks, discussing how their general structure was inspired by the observed behavior of biological neurons, and so on. Let’s jump straight to discussing the basic structure of a simple neural network, as shown in Figure 2-2.

![Figure 2-2. Artificial neural network.](image)

To be clear, a neural network is not equivalent to an LM. We are discussing language models that are constructed as neural networks—but neural networks can work on many other tasks, such as in the area of image processing. An artificial neural network includes the following:

**Input layer**

Consumes an appropriately structured set of inputs.

**Hidden layer (one or more)**

Applies a set of transformations to the inputs.

**Output layer**

Sometimes also referred to as the activation layer, this converts the result of the preceding transformations into something for producing the final output.
The input layer converts each word (or token) into a vector representation that the rest of the neural network can understand—namely, the word embedding we discussed in the previous chapter! Frequently, people will use an existing set of pretrained word embeddings here, such as Word2Vec, GloVe, or ELMo.

We will simplify matters in this chapter by considering only one hidden layer. In reality, deep neural networks are so called because they stack multiple hidden layers, with different weights learned in each layer. In this step, the network applies a set of weights to the embeddings. The values of these weights are the parameters of the model. This model is trained by consuming a snippet of text as input (“Bats range in”), comparing its generated output to the true value in the text (“size”), and adjusting the weights in its hidden layer depending on how close it got. This process is repeated many times over, slightly altering the state of the hidden layer each time.4 Hopefully, you are now gaining some intuition about just how much text such a model needs to consume, and how many parameters it needs to tweak, in order to construct a decent-quality representation!

Having applied the transformation steps in its hidden layer, the neural network has done most of its work. But the information cannot be directly used for anything in its current form in the hidden layer. The output layer asks the question of “so what?” It does so using an activation function—again, terminology borrowed from biology—whose job is to aggregate the information in the matrices into some kind of transmissible output. To be precise, activation functions are also the mechanism by which one hidden layer passes information forward to the next in a deep neural network. For simplicity, we will bypass that part of the discussion and focus only on the activation function in the output layer.5

---

4 Here is where a more technical discussion would get into terms like gradients and back-propagation, to mathematically describe what is going on during the training.

5 For a deeper technical understanding, you might read about the commonly used softmax function in the output layer, and ReLU between hidden layers, among others; for example, see Deep Learning for Natural Language Processing by Karthikey Reddy Bokka et al. (O’Reilly).
For our task, we need to end up with a probability distribution across all possible words that might follow “Bats range in,” just as we have in the probabilities table for the trigram language model in Figure 2-1. A probability distribution is a finite set of numbers between 0 and 1 that sum to 1. Therefore, a particular mathematical function is applied (e.g., softmax) to yield the probability distribution.

What is “inside” an artificial neural network? Matrices. Figure 2-3 shows a simplified example.

![Diagram of a neural network's layers](image)

**Figure 2-3. The composition of a neural network's layers. The actual numbers are meaningless, and the matrix sizes are significantly truncated for this visualization.**

The values in the activation and aggregation layer are comparable to the values in the probabilities table of the trigram language model (Figure 2-1) shown earlier. The operations performed on the
matrices are implied by the arrows. Thus, for instance, when we speak of weights being applied to the embeddings, this is shorthand for an operation being performed on the matrices in those two layers.

**Note**
What is a neural network made of? Matrices, and operations defined to be performed on the matrices.

It is important to understand that this neural architecture is not limited to creating an LM—a system for predicting the next word. Other tasks that this neural architecture can take on include part-of-speech tagging, named entity recognition, sentence classification (e.g., for sentiment), and so on, not to mention the myriad of tasks for image recognition! Applying the architecture to such tasks means altering the functionality of the output layer and the expected output (and for image tasks, the input as well, of course). We are not going to explore these other aspects in this report (which is specifically about language modeling) but wanted to avoid any potential confusion on this point.

We have up to now discussed a feed-forward, fixed-width neural network (so described because it feeds in previous input to predict the next token, or word, forward in the sequence, and the size of the input is a fixed number of tokens). Returning to our example sentence, the neural network would train on it by “sliding” along, pairing consecutive trigram inputs and unigram outputs:

“Bats range in,” “size”
“range in size,” “from”
“in size from,” “the”

And so on. Although this relieves some of the major problems with probabilistic n-gram models, such as those of sparsity and storage, using a feed-forward, fixed-width neural network for language modeling has some serious drawbacks. Words that appear close together will affect the network together, but relationships between words that appear further apart will not be detected.

---

6 For additional reading on probabilistic n-gram models, see, for example, “N-gram Language Models” by Daniel Jurafsky and James H. Martin.
In our example sentence, the network will forget all about “the Kitti’s
tog-nosed bat” by the time it reaches “weighs less than a penny”
because of the intervening parenthetical “(also called the bumblebee
bat).” In other words, even if we widened its fixed-width window to
say, five tokens, the model would have learned exactly the same
information from the input “weighs less than a penny” whether it
was in our sentence about bat sizes or in a sentence such as “That
whole pile of feathers still weighs less than a penny,” although the
human reader will likely think about the weights of these things
differently. The contextual information that surrounds the text input
is completely ignored by the neural network.

Imagine a little robot bat that has been outfitted with an artificial
neural network for learning about (and surviving in) its environ‐
ment. Using this neural network would be like giving our bat the
proverbial memory of a goldfish. Every five seconds, it would com‐
pletely reevaluate its environment and decide where to look for
food, how to avoid predators, and so on. It would take cues only
from what it could sense in that time period, and then immediately
forget everything (as we did in the baking of Cake 0). The bat might
feel full but have no memory of eating; it might be flying extra fast
but not remember that it is being actively pursued. Our poor bat
would really benefit from having access to information beyond the
current moment.

So in the rest of this chapter, we cover how to give neural networks
(and our robot bat) a better memory. We promise the rest of the dis‐
tussion will go more quickly!

**RNN: Adapting Neural Architecture to
Language Modeling**

When training an artificial neural network, all we tell it each time is,
“Here is what is happening now.” What if we could add, “and here is
what you just learned” to the input? We can. Meet Recurrent Neural
Networks (RNNs), illustrated in Figure 2-4.

Compared to an artificial neural network, the main difference in the
structure of an RNN is that it takes in two inputs as it trains: the cur‐
rent token, and the hidden layer (roughly, the values of all the
parameters) after consuming the previous token. By comparing
Figure 2-4 with Figure 2-2, you can see how RNNs are trained by
using sequences of inputs, rather than single inputs.
The RNN processes one token at a time, and hence we speak of iterations over the sequence, or time steps. In our example, we have three time steps. At time step 0, the token “bats” is passed to the hidden layer, which is altered to reflect the consumption of the input. At the next iteration, time step 1, both the token “range” and the hidden layer’s state at the end of the previous time step are passed as input. At time step 2, both the token “in” and the hidden layer’s state at the end of time step 1 are passed as input. The token sequence has now ended, and the hidden layer’s state at the end of time step 2 is passed through the output layer, as shown in Figure 2-5.
Note that although we are currently interested in (and showing) the output of only the last time step, every time step is fully ready to output its best guess for the next word. Another way to look at it is that the RNN is producing an output sequence that is exactly the same size as the input sequence—we are just ignoring it. This fact will become important in the next chapter.

RNNs always produce an output sequence that is exactly the same size as the input sequence.

One aspect that (hopefully) is immediately obvious is that RNNs can process input sequences of any length, and there is no need to specify a window size, which is certainly an advantage. In theory, an RNN could just keep propagating its hidden layer information forever!

Ah, if only it were that simple. In practice, it turns out that an RNN is not successful at accessing information from too many steps back. Why? Well, the size of the model (the size of the hidden layer) remains exactly the same, no matter how far along in the input sequence the model is. In Figure 2-5, the shaded rectangles representing the hidden layer are not growing in size, even though the model is having to remember information about an increasing amount of text. This means that, returning to our bat sizes, if we allow sequences to be as long as sentences, we are limited to the same structure for predicting the next word after “Bats range in,” as for “Bats range in size…up to 6 feet” (43 tokens). Such computation also turns out to be quite slow, for the same reason: at each training step, the network has to adjust its weights not just once, but over and over as it travels “back in time” to the beginning of the sequence.7

We have now upgraded our bat to an RNN and allowed it to remember things a bit further back, such as that it was recently eating or in the middle of escaping pursuit. But it still can’t remember that it visited a promising feeding ground three nights ago, and to check back, because that information has been crowded out by all the

---

7 Technical discussions might now examine gradients that either “vanish” or “explode,” which refers roughly to losing information from too many steps back, or propagating errors from one step in the sequence that is given too much importance, respectively.
Hyperthymesia is an extremely rare condition that allows people to be able to remember an abnormally large number of their life experiences in vivid detail. Similarly, it is liable to get spooked all out of proportion by a single bad encounter and never return to a good feeding spot again. How can we help our bat learn to remember the important things and forget the unimportant ones?

**LSTM: Adding a Separate Memory Structure to RNN**

*Long Short-Term Memory (LSTM)* networks start with an RNN architecture and add a *memory* state that persists over time (for as long as the network is processing the input sequence). At each time step, the hidden layer both accesses and alters the information in the memory state. Figure 2-6 shows a simplified view of an LSTM network.

![Figure 2-6](image)

*Figure 2-6. As compared to an RNN (see Figure 2-5), an LSTM network has an added memory state that persists over time.*

The interactions between the hidden layer and the memory state are called *gates* because they serve as *gatekeepers of information*, deciding what shall pass and what shall be denied passage. The gates are concerned with two tasks: determining which information to forget from the previous time steps and determining which information to remember going forward. Without the ability to forget, the model suffers from a sort of hyperthymesia. Recall the earlier discussion of what an LM learns: it seeks to be as effective as possible in its “task” that for input that is *like this*, it should produce desired output that is *like that*. Not all the information in the model is going to be relevant to the specific task, and therefore that information would be doing nothing but taking up space and computing time. The ability to forget is a significant aspect of memory, for both humans and LMs.

---

8 *Hyperthymesia* is an extremely rare condition that allows people to be able to remember an abnormally large number of their life experiences in vivid detail.
As each input is processed at each time step, the hidden layer is still working on the predict-the-next-word problem, and the gates are choosing which subsets of information to retain. Without getting into any technical detail (which is not our goal), suffice it to say that these mechanisms are also neural networks, and they get trained in the same way. Given an input sequence, the LSTM unit simultaneously learns how to predict the next word and how to remember important parts of the sequence and forget unimportant ones.

The ability to forget is a significant aspect of memory, for both humans and LMs.

Finally, our bat has an LSTM. It can remember important things, like a particularly fertile feeding grounds and good places to hide from predators. The bat will also not get distracted by trifles, such as the location of plants it can’t eat or the exact behavior of every other bat in its colony.

**On Bidirectionality**

We have spent this chapter presenting both RNNs and LSTMs as *unidirectional*, processing text left to right. However, you may have heard of (or are now thinking about) *bidirectional* RNNs. Information in text can often be presented out of order, but that doesn’t cause problems for people. A person will get pretty much the same understanding from the following two sentences:

I was delighted by the bat, especially when examining its large, translucent ears.

I was delighted when examining the bat’s large, translucent ears.

This is because our brains easily flit (like bats!) back and forth, and incorporate the information we just encountered without having to reread it. But the RNN, which moves resolutely left to right, does not flit. Roughly, a bidirectional RNN (or a variant thereof, such as an LSTM) adds a path going right to left for the hidden layer, simul-

---

9 An RNN can, of course, be set up to move right to left for another language, like Hebrew, but the point to note about a standard RNN is its unidirectionality.
taneously working on the sequence of the original text and the sequence of the reversed text.

An immediate practical consideration is that a bidirectional RNN will work only for tasks for which you have access to the full text! Good examples of this are classification and summarization. In a text-generation task, the mode is creating the sequence as it moves along, and therefore clearly cannot take in the reverse of this sequence, which is not yet completed. However, when a bidirectional model can apply, using it is frequently a good idea as it improves performance (and it is still far less computationally demanding than, for instance, a Transformer-based model, described later in the report).

**Considerations on the Use of RNNs**

Nothing about RNNs, or neural networks in general, constrains their applicability to only language models. Neural networks are developed for all sorts of tasks beyond modeling language, and all sorts of data beyond text, such as image, audio, and many others. In this report, we are concerned with the use of neural network architecture only as language models, but the two are in no way equivalent.

Not all neural networks are language models, and not all language models are neural networks.

Language models with RNN-like architectures can be used for a variety of natural language processing (NLP) tasks, such as classification and text generation. We say **RNN-like** architectures, because we have used LSTMs to illustrate just one variation on the basic architecture of RNNs, and in practice multiple variations exist. While they differ in mathematical structure, the main intuition is the same: an RNN enhanced with neural techniques to maintain a memory state. We will not get into further detail here, and encourage you merely to be aware of the existence of variations and to seek out additional concrete information as needed.

Furthermore, nothing requires us to build word-level neural language models. We just as easily can build (as people have) character-level language models, which similarly predict the next character in
a sequence. And as we’ve mentioned, other tasks that this neural architecture can take on include part-of-speech tagging, named entity recognition, sentence classification (e.g., for sentiment), image recognition, and time-series prediction.

Finally, recall our earlier aside on the potential of having more than one hidden layer, stacking them so that one hidden layer also serves as the input to the next hidden layer (via an activation function). In practice, it has been noticed that stacking a few layers (somewhere in the two-to-four range) can improve performance, but stacking too many layers introduces a version of a familiar problem, as we’ve discussed: information starts to disappear. This should give a hint as to one topic to come later in the report, when we discuss Transformer architecture: more connections in the network, now between layers!

Key Takeaways

This chapter has introduced how neural networks can be used to construct language models and then focused particularly on two architectures for text processing: RNNs and LSTMs. Before moving on to discuss more complex architectures, we would like to leave you with the following key takeaways:

- It is important to develop good intuition about the basic building blocks of a neural network language model, in order to tackle the increasingly complex models that are coming to the forefront of language modeling. A simple way to think about neural network architecture is in three main parts: the input layer (embeddings), the hidden layer (weights and memory), and the output layer (activation function).

- It is important for an LM to have a notion of order, because text is a type of sequence data, where order matters. RNNs are a class of neural networks particularly well suited to processing sequence data, such as text.

- A balanced approach to memory is important, both what to remember and what to forget. Architectures such as LSTMs offer an important improvement over RNNs, with the addition of a memory state and information gates.
• For tasks with access to the full input task, such as translation or summarization, it’s often a good idea to use a bidirectional model.

• Finally, just as an LM does not have to be constructed as a neural network, an RNN (or a variant thereof, such as an LSTM) is not limited to functioning only as an LM. An RNN can be constructed and trained for many other tasks, such as speech tagging, named entity recognition, sentence classification, image processing, and so on. Indeed, neural networks themselves are a more general topic than what we cover in this report.
We have now discussed in detail the neural language model perspective on predicting the next word, without delving too much into why this is such a valuable thing for models to learn to do. Let’s pick a more concrete and practical task to motivate our next exploration of neural network architecture: text summarization. We begin by considering how humans think about summarizing text. The same working example is then used to introduce the Encoder-Decoder architecture, which is also referred to as sequence-to-sequence (seq2seq). Finally, we offer a few considerations on the use of encoder-decoders and finish with the key takeaways.

**How Do Humans Summarize?**

Let’s begin by considering how humans approach the task of summarizing text. We will make the example more concrete by using the game of Pictionary. You play Pictionary in a team of two: you randomly select a word that your partner does not see, and you must then draw a picture of the word to enable your partner to guess the word correctly. Figure 3-1 shows a simple example of the steps in the game with the word “bat.”

Regular Pictionary has a time component: your team has a limited time for drawing and guessing, and if your partner fails to guess correctly, you do not pick up a point for your team. But let’s amend the game slightly and create a Summarization version of Pictionary.
Now, you are given a longer chunk of text (instead of a word), which you must transform into a drawing, and your teammate must then generate a short summary sentence from it—say, no more than 10 words long. What would you put in your drawing? Let’s take the example shown in Figure 3-2.

Consider your own thought process step by step. You keep the task in mind: interpreting an image into a 10-word summary. You want to communicate as much information as possible to your teammate, who will create the summary, while also making sure your teammate is aware of the most important information (this will, of course, be subjective on both your part and theirs).

As you read and draw, you make judgment calls about which information is important and definitely needs to be passed along: there must be a bat, and some sort of desert plant, and the bat should look like it is feeding on the flowers of the plant to suggest both...
pollination and eating nectar. On the other hand, unless you and your teammate are world experts on bats, you decide to ignore the fact that it's a Mexican long-tongued bat, not in the least because you have no idea how to specifically draw that species. That is information that you decide will not make it into your drawing for your teammate. To summarize, you are reading the text, noting some portions as important and others as to be ignored, and finalizing your drawing.

**Encoder-Decoders for Language Modeling**

That preceding sentence was meant to be a bit leading, and, hopefully, to remind you of an RNN or LSTM architecture! An LSTM also takes in a sequence of text, selectively remembers and forgets, and refines a hidden representation (like your drawing). But remember that at each time step, the LSTM is processing the text from left to right and outputs its best guess for the next word. So in some way, it can produce an output sequence of text for a given input sequence of text, but with two problems:

- The output sequence of text is necessarily exactly the same size as the input.
- At every time step, the LSTM is making its best guess for the next word based only on text it had previously encountered. Instead, we want the entire text to be processed first, and only then should the best summary be constructed.

The second problem can be addressed by using a bidirectional architecture, as mentioned in the previous chapter. But this in no way alleviates the first problem: tying the size of the output sequence to that of the input sequence does not make sense for a summarization task (or plenty of other tasks, such as translation).

**NOTE**

Constraining the size of the output sequence to be equal to that of the input sequence does not make sense for many tasks, such as summarization or translation.

How did your partner create the required summary sentence? They looked at your drawing and generated a grammatical sentence that contained the salient information they believed you had chosen to represent. Just as one human can transform another human's draw-
ing into text, when the “drawing” is created by one LSTM, we can get a second LSTM to transform it back into text!

This is the main point of **Encoder-Decoder architecture**: one LSTM to encode the text, linked with a second LSTM to decode it. As discussed in the previous chapter, this isn’t limited to LSTMs. Any neural architecture that is tackling a seq2seq task such as summarization is a candidate here, and we refer only to LSTMs in this chapter for brevity. **Figure 3-3** illustrates the approach.

![Figure 3-3. An Encoder-Decoder architecture for running a text summarization example (both input and output sequences truncated for space).](image)

In this Encoder-Decoder architecture, one LSTM (the **encoder**) consumes the source text to be summarized. The hidden layer of the neural network after processing “cacti” is a representation of all the information gleaned from the input sequence. A second LSTM (the **decoder**) is then initialized with input of the final state of the hidden layer of the encoder as well as a special token (denoted in the figure as \(<\text{START}>\) ) to indicate that it should now begin to produce output.

Going forward, the decoder LSTM acts just as we discussed in the previous chapter, with one modification: the input at any time step is the hidden layer from the previous time step, and its own token output from the previous time step, as illustrated by the dotted lines in **Figure 3-3**. The decoder has nothing else to go on, so it has to assume its best guess is always correct, and uses it as input to continue generating the sequence.

A reasonable question is, when does the decoder stop? Just like the \(<\text{START}>\) token, a special \(<\text{STOP}>\) token is in the vocabulary, and
the decoder learns from its training examples that sometimes the best guess for the next word is to declare the output sequence completed (because all text output should eventually come to an end). Therefore, when the decoder runs, it will, ideally, eventually generate the <STOP> token as its best guess. Alternatively, you might specify the length of the output sequence directly (e.g., stop after 10 tokens).

**Considerations on the Use of Encoder-Decoder Architecture**

We have spent this chapter discussing a simplified, Pictionary-inspired summarization task. Text summarization is usually more practically deployed on much greater volumes of text, such as documents or even document collections. In fact, Encoder-Decoder architecture can be applied to plenty of other tasks, as long as they can be defined as sequence-to-sequence. Examples include (but are not limited to) chatbot conversations, machine translation, and question answering.

Going further, the medium of information is not required to be the same for the input sequence as for the output sequence. Consider the task of automatic image captioning. Here, the Encoder-Decoder architecture would use a Convolutional Neural Network for encoding the image, and a Recurrent Neural Network for decoding the resulting state into a generated sequence of words.

We would like to make one important comment on the text-summarization task used in our example. This type of summarization is called *abstractive text summarization* (ATS) because the summary is created by using a generative model. The other type of text summarization that exists is called *extractive text summarization* (ETS), which directly selects words, phrases, or sentences from the

---

1 This is not to say that <STOP> is always guaranteed to show up. The decoder might dissolve into all sorts of bad behavior, such as generating a loop of one or more words ad infinitum.

2 Just as Recurrent Neural Networks are a class of neural networks particularly suited to processing text, Convolutional Neural Networks are a class well suited to processing images.

3 Can you go the other way? Sure! Text-to-image generation is, likewise, exactly what it sounds like: a sequence of text as input and an image (a sequence of pixels) as output.
original text to construct the summary. ETS uses quite different techniques, creating an intermediate representation of the text to identify the most important snippets. It does not involve any natural language generation. The created summary will remain faithful to the original text but may be quite ungrammatical. ATS, on the other hand, is far more likely to construct a grammatical summary but provides no guarantee of veracity. A factually incorrect (and somewhat extreme, to make the point) but perfectly grammatical summary of our running example text might be, “Some tongues eat cacti from nectar plants.”

Interestingly, it is possible to combine techniques for extractive and abstractive summarization to improve matters. One approach is pointer-generator networks, and we invite you to read up on them if you are interested (but please finish this report first, as that technique involves an attention mechanism, which is introduced in the next chapter).

There are two main types of text summarization. Abstractive text summarization creates the summary by using a generative model. Extractive text summarization creates the summary by copying the most important snippets from the input text.

Key Takeaways

We have now built on the basics of neural networks by introducing encoder-decoders. Before we continue with the next set of innovations in neural architecture for language modeling, we would like to summarize:

- Encoder-Decoder is a modular architecture for arranging two neural networks. It can use RNNs, LSTMs, or any variations thereof.
- Encoder-Decoder architecture is great for seq2seq tasks such as text summarization, translation, question answering, image caption generation, and many others (note, again, that we are not limited to only text data).
- As one improvement on RNNs, the length of the input and the length of the output do not have to match with encoders-decoders.
- The basic Encoder-Decoder architecture has a bottleneck problem: *all* the information from the source text needs to somehow be contained in one single state of the hidden layer (the end of the encoder step). This can significantly degrade performance on long input data. The next chapter discusses how to address this problem.
In this chapter, we explore a fictional example of translating between English and a fantasy language to convey certain underlying concepts surrounding attention mechanisms. We then discuss in more technical terms how attention is achieved, which we will then extrapolate and view through the lens of the Transformer architecture. The task that these architectures will be applied to in this section is machine translation. We will begin with some observations about the human approach to translation. Once the key concepts have been explored, we discuss how attention and the Transformer architecture work from a technical standpoint. Lastly, we discuss key considerations of these models, where they are applicable as a solution, and key takeaways from the chapter.

**Communicating Across Languages**

Imagine that you are on a cruise to the Bahamas. Upon what you believe to be your arrival, the ship’s captain informs you, in English, that it seems that the ship has gotten lost inside the Bermuda Triangle, and you have instead arrived at an uncharted island. Unfortunately, the cruise vessel does not have enough fuel to return to course, so the captain asks you to go and speak with the islanders.

Upon hearing their native language, you are unable to recognize similarities of any language you are familiar with. You come up with a clever idea: why not point to, or mime, various objects and actions,
and both you and the islanders will write the appropriate words in
the sand. To you, their written language looks like a bunch of trian-
gles and circles arranged in different configurations, but it clearly
has the same concept of words and structure.

You point at yourself, write your name, and say it; an islander does
the same. You point at the boat and write “boat,” and so on. Eventu-
ally, you have a mapping of their language to several basic English
words, though you have little understanding of how to say, pro-
nounce, or produce their language yourself, and some misunder-
standings likely occurred during the process. You get to work trying
to translate the English sentence, “The boat needs fuel so it can
leave,” into their language. After much back and forth, you comu-
nicate the basic thought that you must refuel the boat, to which they
happily oblige.

When you finally arrive back home, you are able to find a dictionary
with their language and check what was said to ultimately get you
back home: “Water dragon wants to bloom; it needs food.” 1 This
is not what you meant, but it had the intended effect. You keep looking
through the dictionary and realize a couple of things:

• You were pretty successful communicating about objects.
  “Water dragon” makes sense as a translation for “boat”; and as a
  result, “food” makes sense for “fuel” since, ostensibly, dragons
  need to eat.

• You realize your blunder with “leave” turning into “bloom.” You
  had pointed to leaves on trees, forgetting that it’s only a quirk of
  English that “leaves” is both the plural of “leaf” and a conjuga-
  tion of “to leave.” The islanders thought that you were pointing
  out what things on trees do, which is bloom. They were proba-
  bly puzzled as to why you referred to your “water dragon” as
  blooming but still managed to successfully guess at your
  meaning.

• You had translated your sentence in chunks. You did not try to
  get across the entire representation at once. You started with the
  main topic—the boat—and the two things you wanted to com-
  municate about it: it wanted to leave, and it needed food. You
  started causing confusion by apparently insisting that your boat

1 Yes, this is all reminiscent of an episode of The Twilight Zone.
needed to bloom. But by the time you moved on to explaining that it wanted food, both you and the islanders were able to set that aside and pay attention to only the agreed-upon fact that there was a boat, and, finally, that it was hungry. In translating the second half of your sentence, you were both paying attention (foreshadowing!) to only the “boat” part of the first half of your sentence.

Figure 4-1 illustrates this point extremely simply. When translating “The boat needs fuel” to “Water dragon needs food,” it is easy to realize which words are aligned for the translation.

![Figure 4-1. Simple alignment of words in part of the sample translation.](image)

To summarize, your translation encounter with the islanders had a few notable aspects. First, you focused on only a subset of the text to be translated at a time, so as not to muddle the communication channel with irrelevant information. Second, you used a variation in translation that makes sense: the easiest way to communicate “fuel” was to mime yourself eating, something all creatures do, and so for the translation it did not even matter who was doing the eating, you or the boat. Finally, you used another variation that made sense in only one language but became muddled in the other (the double meaning of “leave”); still, the overall final translation managed to be successful enough.
Attention for Language Modeling: Concentrating on Only the Relevant Input

Let’s now discuss in more technical terms how attention is achieved, specifically through the lens of the Transformer architecture. The motivation for attention is, once again, in the desire to mimic human behavior and our ability to focus on only the relevant context when processing any particular word. How can this be implemented in a neural network? Let’s go back to our Encoder-Decoder architecture in the preceding chapter.

Simply put, an attention mechanism is another layer in the neural network that allows the decoder to consult with both the final hidden layer, produced by the encoder, and a select subset of the original input to the encoder. In other words, attention is a filter for the relevant context. The attention layer is trained to decide what is relevant at the same time as the rest of the network is being trained.

Attention acts as a filter for the relevant context for the current input text. It is trained to decide what is relevant at the same time as the rest of the network.

Figure 4-2 highlights the main architectural differences from the Encoder-Decoder architecture in Figure 3-3. The decoder still gets the last state of the hidden layer of the encoder, but it is now augmented with input from the attention layer, which directs the decoder where to look at each of its time steps. By the time the encoder is finished, the hidden layer holds information about the boat, fuel, and the former’s need of the latter. But to translate “boat,” considering anything else in the sentence is not relevant, and so the attention layer here would bring the focus entirely on the input, “boat,” and block out the rest of the input. The example in Figure 4-2 is extremely short, but it is easy to imagine how useful this becomes when the input text is long and convoluted.
We have chosen machine translation as the task through which we explain the attention mechanism, following in the footsteps of the academic paper that introduced attention to the NLP community via machine translation.\textsuperscript{2}

Is Attention Interpretable?

Although we used a translation task to introduce attention, it is a general technique that can be used in many architectures, not only Encoder-Decoder, and, of course, is useful in many tasks beyond machine translation.

An interesting ongoing discussion is whether attention is interpretable, in the sense of providing an explanation for the model’s behavior. Attention does give a sort of soft alignment between input and output. Examining the attention scores at each time step of the decoder shows a particular interpretation of what the model had identified as the relevant context to be in the form of a distribution of scores over the input tokens. Therefore, you might decide that, by consulting this attention distribution, you can understand the decision taken by the decoder to emit that particular token. Alas, as with most aspects of deep learning, things are just not that simple!

First, attention can be quite noisy, and the relative importance of the context can vary across seemingly similar examples. Interestingly, especially with tasks such as classification, different attention distributions can nonetheless yield equivalent final predictions. In fact, it’s possible to directly interfere with the attention layer and force the model to ignore the most “attended to” tokens, and yet still yield the same output.

These points should strengthen our intuition against relying on attention as a robust pointer to tokens that are “responsible for” the output of the model. Once again, no actual reasoning is going on, only associations. At best, attention can be thought of as giving a plausible reconstruction of the model’s decision process (it could have happened this way), though there’s no guarantee of faithfulness (what really happened?).

Is attention interpretable? There are reasons to answer both yes and no, and this important discussion is actively ongoing.

Transformer Architecture for Language Modeling

Adding an attention layer to the Encoder-Decoder architecture definitely improves things. But we still have a strange—from our English-language perspective—constraint. Yes, we are trying to go from one sequence to another, but why do these sequences need to be processed left to right? After all, what if our example sentence was, “The boat wants to leave and so it needs fuel”? Does it really matter very much, for the final meaning of the translation, whether the leaving or the fuel is discussed first?

We would like to have access to the whole sentence when considering the context for a particular word. In fact, we often want to go beyond a single sentence. If our running example were two

---

3 For further details, see “Attention Is Not Explanation” by Sarthak Jain and Byron C. Wallace, NAACL-HT 2019.
4 For further details, see “Learning to Deceive with Attention-Based Explanations,” by Danish Pruthi et al., 2020.
sentences—“The boat needs fuel. It wants to leave.”—we would have to cross sentence boundaries to see that “It” refers to “The boat.”

Yet, recall that when we first introduced neural networks, we pointed out the issues with using word windows, which grab all of the context within a certain number of tokens away. How can we still keep a sequence-to-sequence process, with the clear benefits of working with both long sequences and an attention layer, while breaking free from both the restrictions of fixed-width windows and the forced unidirectional processing of an RNN architecture?

Transformers address these concerns. Here is a broad outline of the Transformer architecture (Figure 4-3): first, stack multiple encoder layers, where each layer is an implementation of the neural network learning to pay attention to itself, by feeding the attention layer back into its own layers (rather than into a decoder layer as in Figure 4-2). The encoder stack is then followed by a stack of decoder layers doing the same thing. To be a little more precise, two attention mechanisms are between every pair of decoder layers, one for paying attention to itself, and one for paying attention to the encoder output. The model has access to the full input text, and the input and output sequences can still be different lengths.

The attention mechanism can now get even trickier. First, self-attention allows the model to understand what information is relevant in specific sequences—no matter their order. Second, instead of one mechanism informing a hidden layer, we can have several (this is called multihead attention). This approach is trying to mimic the fact that there are many reasons for a word to appear as context for another word that go far beyond simple alignment.

The Transformer architecture has three key aspects, in contrast to the architectures we have discussed previously:

**Stacked layers of neural networks**

This is not a new concept by itself, as you may recall the brief comment on stacking RNNs to improve performance earlier in the report. But precisely how the layers are stacked is different. We are no longer just stacking hidden layers on top of each other like a sleeve of crackers; now we’re making sandwiches.
**Self-attention layer**

Every encoder now comprises a layer of self-attention as well as a hidden layer, which is an attention mechanism for focusing the current hidden layer on subsets of the previous layer. Every decoder has a hidden layer and two other components—one for self-attention, and the second for paying attention to the encoder. See Figure 4-3 for a simple visualization.

**Multihead self-attention**

There is now more than one way to focus on word context. Self-attention is now multiheaded.

---

**Multihead Attention**

To explain the third point, multihead self-attention, with a concrete example, let’s pretend to be translating the sentence “I kicked the ball.” Take a look at Figure 4-4, which shows how we might think of attention information traveling from one layer to another.

Imagine three attention mechanisms working in parallel: the red one focuses on actions, the green one focuses on who does the
In a real Transformer architecture, some part of the attention mechanism will care about articles to ensure a grammatical output.


Figure 4-4. A multihead attention mechanism (source: “Self-Attention for Generative Models” by Ashish Vaswani and Anna Huang).

Let’s immediately walk back the preceding paragraph. We cannot actually state that, for example, green attention is looking for the semantics of “who” is doing any particular action in a sentence. Again, LMs don’t understand meaning and have no concepts of semantics. This is helpful shorthand for us to represent the way in which a sufficiently large language model will discover associations that align closely with known linguistic structures.

The original paper introducing the Transformer architecture stacked 6 encoders and 6 decoders, to form a deep neural network of 12 layers.

---

5 In a real Transformer architecture, some part of the attention mechanism will care about articles to ensure a grammatical output.

Considerations on the Use of Transformer Architecture

We’ve explained the Transformer architecture by using the example of machine translation and Encoder-Decoder architecture, and things have already gotten fairly complicated. But we have just one more step to go: how is it that we have also been able to shift from a model that handles seq2seq tasks to the current state of the art: a pretrained language model (you are likely to have heard of BERT, Bidirectional Encoder Representations from Transformers) that can be fine-tuned (tweaked a bit with more training data) to all sorts of other tasks?

What part of the Transformer architecture is still playing “predict the next word”? The decoder. So what happens if we throw out half of the structure and create an architecture of only decoders? Generative Pretrained Transformer (GPT) models, introduced by OpenAI, are providing increasingly interesting results in text generation. But now we’ve gone back to only a unidirectional model! How can we train a Transformer-based model that looks at the full context of a word, both to the left and to the right?

Hold my beer, says BERT.7

Instead of trying to predict the next word while training its encoders, BERT masks, or hides, a small subset of the input text from itself and tries to guess it. BERT is playing fill in the blank. Now the model can look quite far to the left and right of each word as it tries to guess, and it repeats this game millions and millions of times. Having completed this (large-scale, in every sense) training process and built a complex representation of language, the model is now ready to be put to work on other tasks.

We will give a brief overview of how fine-tuning works on one example task: sentence classification. The training data for this fine-tuning step looks just like data for traditional machine learning classifiers, comprising pairs of sentences and labels. The input sequence

---

7 A huge hat tip to Jay Alammar and his blog post, “The Illustrated BERT, ELMo, and Co.”, in particular. If this expression is new to you, you can read about “Hold My Beer” on Know Your Meme.
is constructed from a special token called `<CLASS>`, followed by the sentence, and the output is the label. BERT is tasked with guessing the value of `<CLASS>`.

The amazing thing is that this approach is quite effective when using the exact same pretrained BERT model across multiple tasks. Furthermore, this approach is frequently more successful than previous (differently structured) models that were constructed and trained to perform specifically on those tasks—while needing significantly less labeled training data (for fine-tuning). The language representation that this Transformer architecture has learned is both useful and flexible. It has transferred its knowledge of English generally from its original fill-in-the-blank training task to the new task (sentence classification or other). This is referred to as transfer learning.

Transfer learning is pretraining one large language model, such as BERT, on a huge amount of unlabeled data, and fine-tuning this single model to many tasks with much smaller sets of labeled data.

Finally, we have come full circle to our first technical topic: embeddings. A pretrained BERT architecture model can be used just like GloVe or ELMo to create contextualized word embeddings, and you can feed these embeddings into your model! This is another way to make use of BERT, in contrast to using it as a model directly and fine-tuning it to your task.

We have arrived at the current state of the art in LMs, and they are large, powerful, and resource hungry. In the next (and last) chapter of this report, following our review of key takeaways, we get into the implications of this state of affairs.

**Key Takeaways**

This chapter has introduced the most recent advancements in neural network architectures for language modeling. We conclude by reemphasizing some important takeaways around the attention mechanism and Transformer architecture:

---

8 Recall the `<START>` and `<STOP>` tokens mentioned in the previous chapter.
9 Of course, there are variations of embeddings; researchers have been busy!
• Attention usually improves performance on sequence-to-sequence tasks by acting as a filter for only the relevant context for the input text.

• Attention mechanisms have become an expected component in LMs.

• Attention may seem to provide a convenient answer to the lack of interpretability of deep learning models. Alas, as with most aspects of deep learning, things are just not that simple. We must be careful to remember that everything is still an association, not reasoning.

• There are three key aspects to Transformer architecture:
  — Multiple layers of encoders and decoders are stacked, with each decoder paying individual attention to the output from the encoder stack.
  — A self-attention layer is sandwiched between each pair in the encoder stack and each layer in the decoder stack. Self-attention allows the model to understand what information is relevant in specific sequences, no matter the order.
  — Inside these self-attention layers, multihead self-attention provides parallel attention mechanisms for focusing on word context in different ways.

• BERT is a powerful Transformer-based language model that doesn’t play “predict the next word” to train; it plays “fill in the blank.”

• Transfer learning is the use of a single pretrained model, such as BERT, on many NLP tasks (such as classification, question answering, translation, and summarization) by fine-tuning. This approach can outperform other models constructed to perform specifically on those tasks, while needing a significantly smaller amount of labeled data (as BERT is pretrained entirely on unlabeled data, only the fine-tuning step requires labeled data).
Now that you better understand a variety of language model architectures and associated tasks, you are probably thinking, “Time to collect data and get to training!” Unfortunately, this is not the case. As we have noted, LMs learn about syntax and structure of language and their associations, but not the underlying meaning. So how would machines read and comprehend? Let’s use an observation of human behavior to explore this concept.

Imagine that you are watching a young child who is just now learning to speak. To learn how to speak, the child must practice. This requires them to hear words, say words, read words, and engage in conversation. They, of course, will receive guidance from you to inform the direction of their learning.

Now imagine that you are not there to help direct them. Instead, they are taken from you and are raised in a mall for one year. There they learn language within the context of the mall, where they pick up the vocabulary to describe their sensory inputs, like what they see and smell. They come back to you, now capable of speaking quite well. Unfortunately, they seem to have little insight into what their words mean. They had picked up only how to string words together in the context of a mall, and now they are home.

You ask the child how they are doing, to which you receive the response, “What’s good?” Puzzled, you say you are doing well; however, you are hungry, so you’d like their input on what they want to eat, to which they respond, “A number 5.” You inquire further, “So you want chicken wings?,” to which the child responds, “No, a
number 5.” At this point, you realize that the child learned to order food at a mall but does not understand when prompted at home. It appears that the child has picked up the ability to string words together, but their communication does not make a lot of sense in this new context. Worse yet, they talk like a 15-year-old…who hangs out at the mall.

Now you realize that you don’t know what the child comprehends, other than that, when you speak to them, they should respond with words that technically make sense. But now that the context has shifted, they sometimes say sensible things and other times say nonsense. You ask, “Why can’t you always talk properly at home?,” and they look at you and say, “What’s the difference?”

You need a lot of data. But you also need the right data.

The child from the preceding example went out into the world with no previous context and learned to speak English. They were able to observe and extract enough information from their environment to make their way through the world as they knew it. Unfortunately, the things they learned to say at the mall do not always translate well when speaking at home. For your part, you have a long struggle ahead of you to change the child’s language. It is impossible to trace all of the influences that the time in the mall had on their understanding of the structure and meaning of English, and retraining them will take significant time and effort. The child is likely to continue issuing strange non sequiturs for a long time to come.

An LM would be happy to train on a year’s worth of spoken and written language of the mall and would perform effectively so long as it remained “at the mall.” But when asked to work in a different context, the language model’s performance may suffer, and—just like the child—it will not be able to explain itself. The large LMs that have become popular today are extremely hungry: they need a lot of training data to perform, and they do not care what that data is. So how do we keep a grounded perspective on what LMs are able to

---

1 This isn’t a stretch of the imagination! Large language models are trained on text available on the web, and we know what a mixed bag of quality that can be (like the mall).
accomplish? In this chapter, we finally dive into understanding, or in our case natural language understanding.

**What Do Language Models Understand?**

For humans, the process of understanding language, or text, generally has three parts: process the text, understand its meaning, and integrate with existing knowledge. LMs, even the complex ones, are good at the first part and sometimes mimic some of the behavior observed during the second and third parts, though without being able to intentionally reason about meaning.

Although LMs do not perform any reasoning, they frequently can appear as if they have done so.

Let’s take a concrete example, returning to the generative task of predicting the next word. We found an available demo online of a GPT model. We tried the following short experiment:

| Our provided prompt: I love Paris, France. My favorite city in Italy is Rome. I like the food in both of them. I like the weather in the two of them. I like the culture in the two of them. I like the art in the two of them. I like the food in the two of them. I like the lifestyle in the two of them. I like the people in the two of them. I like the architecture in the two of them. I like the history in the two of them. I like the architecture in both of them. I like the people in both of them. I like the food in both of them. | Output from model: I love Paris, France. My favorite city in Italy is Rome. I like the food in both of them. I like the weather in the two of them. I like the culture in the two of them. I like the art in the two of them. I like the food in the two of them. I like the lifestyle in the two of them. I like the people in the two of them. I like the architecture in the two of them. I like the history in the two of them. I like the architecture in both of them. I like the people in both of them. I like the food in both of them. |

Let’s start with the cool part. The GPT language model completed an incomplete sentence by listing a city; the city listed, Rome, is indeed a city in Italy; and Rome has the same relationship to Italy as Paris does to France (they are both capitals). All this with no reasoning or external knowledge! Next, the rest of the sentences in the output are all sentences that could reasonably be things to say about cities.

---

² The demo is provided at [EleutherAI](https://eleuther.ai). The model is called GPT-J-6B, and, according to its [GitHub repo](https://github.com/EleutherAI/gpt-j), is a version with six billion parameters.
which have food, weather, and culture. The model did not output something such as, “I like the bedroom in both of them.” The model listed only aspects that make sense for cities. And, of course, the rest of the sentences are complete, grammatically correct, and remain in the first person, as the prompt did.

On the other hand, the output is not very stimulating and quickly starts to repeat itself. The model does not go anywhere interesting with this topic of favorite cities beyond monotonously listing aspects that are liked in each of the two of them or in both of them. We might hypothesize all sorts of things: that this prompt was too short, or, because the sentences in the prompt were short, the model took a cue to also prefer short sentences for the output. But we can’t figure out what happened with the associations formed by the millions—or billions—of parameters.

We may want to think that, since the model correctly identified Rome as the parallel to Paris, it would be able to handle analogies correctly. OK, let’s try.

Our provided prompt: Paris is to France as Rome is to

Output from model: Paris is to France as Rome is to Italy and New York is to the USA. This is an accepted truism, with many of us knowing that without a doubt. Now, what if it were reversed? What if France were the capital city of Italy and Rome were the capital of France? There is an argument to be made that this might be the case; but, in reality, there is little to no historical evidence to suggest this is actually the case.

The Roman Empire fell in the fifth century, after the collapse of the Western Roman Empire. France, however, did not become a unified state until the seventh century. The first capital city of France was Paris.

Once again, the Rome-Italy relationship comes up correctly in the beginning. The next relationship may or may not be OK: New York is not the capital of the United States but is definitely one of the largest and most culturally significant cities (like Paris and Rome). But going further, the text gets really trippy, first calling countries having capitals a truism (huh?), wondering what would happen if one country were the capital of another (not possible), and winding up with some absolute nonsense with a historical flavor. And it’s all
A much more technical discussion can be had here. Basically, sometimes a parameter can be tweaked to “freeze” the model so that identical input will yield identical output. Some outputs will appear to be of better quality to you, and some of worse, and we strongly encourage you to play around with any available demos until you have enough anecdotal data points to satisfy your own intuition. But to the model, as to the child in the mall, it will all be a case of “what’s the difference?” As far as it’s concerned, it all makes the same amount of sense.

**Language Models Are Not Always Enough**

By now you’re hopefully quite comfortable with the fact that LMs are good at producing text that reads well—but it absolutely may not be factually correct, and the combination of sentences may appear to be nonsense. LMs do not perform any sort of reasoning and do not by themselves have any idea of common sense or anything that they have not encountered in their training data. Therefore, a robust end-to-end system may seek to augment LMs with other components that can either perform some sort of reasoning or incorporate additional knowledge (or both).

Let’s consider a conversation system (a chatbot) for making travel plans as an example. For every iteration in the conversation, the steps the system takes are roughly as follows: process the input from the human user, identify the salient parts of the text, gather the information to form the response (including from earlier in the conversation), and return an answer in the form of well-structured natural language. LMs are great at the first and last steps and can give helpful information for the middle steps. But when left entirely to their own devices, they must fudge the middle steps as best they can, going entirely off associations they have formed during their training, which may be fairly mysterious and inconsistent, especially for a complex task like conversation.

---

3 A much more technical discussion can be had here. Basically, sometimes a parameter can be tweaked to “freeze” the model so that identical input will yield identical output, but frequently large models have an element of randomness in the process of choosing the final generated text. The specific model demo we played around with certainly does.
What can be added to the system are components that work with the
LM to perform reasoning or consult with outside knowledge; note
that the type of reasoning and knowledge that is, or is not, deemed
to be necessary depends on your use case. Many interesting vari‐
tions are possible, so we will just briefly mention two and encour‐
you to look into more.

One is the approach of entity linking.⁴ If we have a large knowledge
base with many known entities and relationships (and we some‐
times do, depending on the task domain), which is well structured
and fully understandable, identifying the presence of those entities
and relationships in the text can go a long way. Taking the sample
user utterance, “I’ d like to go to Italy. Are there any flights to the
capital?,” the process of entity linking would identify the entity
“Italy” and the relationship “capitalOf.” A reasoning leap would then
query the knowledge base for the missing component of the triple
(?, “capitalOf,” “Italy”) and get the result (“Rome”), thus determin‐
ing the desired destination.

Another interesting approach is to create a new embedding space of
entities that the language model can consult. The motivation here is
named entity resolution. Consider three phrases that all refer to the
same city: “NYC,” “New York City,” and “New York” (the last one is,
of course, ambiguous without context). A traditional language
model will necessarily have learned somewhat different representa‐
tions of each. A pretrained entity-embedding model could represent
these as exactly identical, to approximate the concept that they refer
to the same entity.⁵ In the conversation with the travel chatbot, if the
human user first mentions “traveling out of NYC” and later asks
about “flights from New York,” this may help the system recognize
them as referring to the same entity.

To reiterate: for humans, text (written or spoken) comprehension
requires processing text, understanding its meaning, and integrating
with knowledge. Language models do the first step, and a model that
is great at it (such as BERT) is a real asset, as it can be constructed
once and fine-tuned with a whole lot less data for successful deploy‐
ment on all sorts of tasks, from question answering to translation to
text generation. From the perspective of BERT, it’s doing the same

---

⁴ We are referring to a named entity, which is generally a person, place, or thing.
⁵ One example is ERNIE.
step of text processing. But LMs cannot perform the second or third steps of reasoning and knowledge integration. Even though they can sometimes give a startling imitation of doing so, they cannot differentiate between correct and incorrect information.

On the other hand, it is important for the person deploying the system to understand whether that matters for the required task. LMs can still be quite effective at particular tasks without any reasoning or knowledge. For example, an LM trained on a sufficiently large number of question-and-answer pairs could do an excellent job at the question-answering task and be quite sufficient for the practical application.

With Great Power Comes Great Responsibility

The current story of LMs is one of scale and adaptability. The basic design of neural networks is not recent, and the underlying mathematics of matrix operations and derivatives significantly predate our current technological era. LMs have become large and powerful for three main reasons: the development of the deep neural network architectures discussed in this report, especially Transformer-based models; the parallel improvements in computation power; and, finally, the availability of more and more data for training these models.

And so we have arrived at an interesting transition point. We are no longer talking only about people using the same types of models for machine learning tasks (SVMs, logistic regressions, even RNNs). We now have more and more examples of people using the same main language model (such as a specific pretrained BERT) and fine-tuning it for all sorts of tasks, both because this approach has been successful across tasks and because of the really high amount of computing and data resources required to construct and train large LMs.

All of these fine-tuned models are then going to inherit both the benefits and the problems of a small number of large LMs.6 We will touch on just some of these:

---

6 For an interesting and timely, although also lengthy and academic, discussion of these points, peruse the inaugural report from the Stanford University Center for Research on Foundation Models (which are basically large language models), “On the Opportunities and Risks of Foundation Models” as well as “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” by Emily M. Bender et al.
Security

Having a single model as the basis runs the risk of it becoming a single point of failure, whether as a result of bad data, robustness failure, or actual attacks.

Regulatory

The transfer learning that is done with models like BERT effectively allows you to train on the web and then fine-tune and deploy models across domains, industries, and clients. The legal and privacy aspects of this process are far from clear, and the regulations certainly have not yet caught up with the technology.

Trust

The needs of large LMs emphasize quantity over quality and opaqueness over transparency. As discussed in the beginning of the report, models are great at producing plausible content but have no ability to care about the veracity of their output. It may be dangerous to make decisions solely based on these models, whose outputs can be hard to explain, and which can fail in unexpected ways. Furthermore, because the output can be so plausible, opportunities exist for misuse, such as creating disinformation, especially with targeted content.

Bias

Intrinsic biases from the training data will make themselves known. Particular communities and perspectives will inevitably be either over- or underrepresented, or associated with particular points of view (as one real example, researchers have found that BERT associates phrases referencing persons with disabilities with more negative sentiment words).7

Ethics

The scale needed to create a large language model such as BERT is large. On the one hand, the financial and environmental impacts of the necessary efforts to train and run the model can be significant. On the other hand, few people will even have access to the resources to do so, and those developers’ choices in design and deployment may have widespread implications. To

7 See section 4.3 of “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?".
put it bluntly, we should continuously be asking “when should a model be built, and when should it not be?”

Final Takeaways

We hope that you have gained an appreciation for the general functionality of LMs, that you will now be able to take a step back from the logistical details and have a thoughtful conversation with the various stakeholders in your business about high-level language model concepts, and that you have reasonable confidence in how to take the next steps toward applying LMs to your business. We would like to leave you with these last few thoughts on humans, machines, and language:

- When humans communicate, we do not assume that the other person is thinking the exact same way to produce a thought. We assume they are able to parse and understand the underlying semantic information of a word, sentence, or phrase, and abstract this and, in turn, respond.

- While humans interpret language to have both structure and meaning, LMs are aware of only structure. They are entirely focused on outputting something grammatically correct, without the notion that there is such a thing as grammar. They are unable to grasp the concept of words having meaning. In the same way, an LM cannot intentionally tell lies, obfuscate facts, or spare your feelings.

- LMs are an augmentation to human interactions, not a replacement for them.

- LMs do not actually perform any reasoning, but they frequently can appear as if they have done so.

- LMs are components in text-processing systems. An LM is not a chatbot; it is the chatbot’s—or any application’s—text processor. For some use cases, using an LM by itself may be sufficient. In other cases, additional techniques for using manual rules, external knowledge, or reasoning may be advantageous.

- LMs are multitask learners to the extent that they learn, in their way, a ton of information about the input text; however, what ultimately decides their usage is the task you put them to work on.
• Current state-of-the-art LMs are extremely interesting and powerful, but the larger these models become, the more carefully we must consider the implications and risks of their usage.

• Bats are super cool (see Figure 5-1).

Figure 5-1. A little brown bat (source: US Department of the Interior).
About the Authors

**Austin Eovito** is a senior data scientist on the Client Engineering Team at IBM. Austin’s current role focuses on understanding and solving complex enterprise AI problems for Segment 1 clients. His role covers areas such as natural language, computer vision, and capacity management. He received his MS in computational science from the Florida State University in 2019, and his BS in applied economics from the Florida State University in 2018. He lives in Tampa, Florida with his wife, two dogs, and two cats.

**Marina Danilevsky** is a research staff member at IBM Research. Her work centers on understanding structured and unstructured text data, with a particular focus on model explainability and human-in-the-loop techniques. She received her PhD in Computer Science from the University of Illinois at Urbana-Champaign in 2014 and her BS in Mathematics from the University of Chicago in 2007. She lives in San Jose, California, with her husband and two children.

Acknowledgements

We are grateful to our family, friends and colleagues for their valuable feedback while writing this report. Special thanks to Benjamin Bengfort, Tony Ojeda, Aya Soffer, Salim Roukos and the amazing team at O'Reilly, especially our editor Nicole Taché.