Next-Word Prediction, Good-Turing Estimation, and Expectations of Sums

Ben Bavar • 2 Apr 2025

Next-Word Prediction

I open a new chat at chatgpt.com. I send the prompt “Which word is most likely to come next?” GPT promptly responds, “Sure! Please provide the sentence or sequence of words you want me to complete.” (See the full chat here.) What’s going on when I have this interaction with GPT? How does it know what to say to me?

You may have heard someone sum up how GPT works along the following lines: “It predicts the next word in a sequence, that sequence being the text input the LLM receives from the user.” As a sketch of the mechanism, this is serviceable. But it doesn’t capture how GPT differs from a simple n-gram model (not that I’m presently able to articulate that difference adequately myself; I just know transformers like GPT are much more advanced, complex technologies than n-gram models). The explanation is rather confusing in certain respects as well. And no, I’m not referring primarily to the use of the word “word” instead of the technically correct term “token.” Here’s what I mean.

Where does a sequence of words begin and end? You might think a sequence of words begins when a user starts typing a prompt and ends when they hit send. But obviously GPT doesn’t predict that there is no next word in the sequence just because the user has finished typing the message (and it expresses a complete thought). Or, if GPT does predict that, that predicted absence of a word differs markedly from GPT’s typical, notoriously lengthy responses.

A natural thought is, the sequence whose next word GPT predicts is the conversation the user initiates, not a single message. So the sequence is not forever limited to the prompt GPT is currently answering; it will include all the prompts and answers there ever will be in the current chat. But if the sequence is that exact conversation, something seems to be missing from our account of how GPT works. For if GPT possessed autonomy relevantly analogous to that of a normal interlocutor, GPT could make its job easy by opting out of continuing the conversation, and, if necessary, preventing the user from sending further messages in the chat. That way GPT could always predict no word will come

after the prompt. We might expect GPT to sometimes predict that anyway, because, in human interactions, it's normal, in the right context, not to respond to certain sequences of words. So why does GPT invariably predict words *will* come after the prompt?[1]

I suspect one reason GPT does not end the conversation of its own accord is because it's trained to say the sort of thing that tends to be said in a conversational context similar to the one it's in, not just to say what it expects will be said (or, rather, what it expects to say) in the conversation it's having right now. So the sequence about which it's making predictions is a hypothetical, normal conversational sequence comparable to the current chat, not this latter chat itself. But, again, in some conversational contexts or contexts where attempts are made to initiate conversations, people tend to say nothing even if others have shown interest in dialogue with them. So there's still a question as to why GPT never mimics that undesired silence. I'd venture to say its behavior of responding to every prompt (unless it's been prompted not to respond, in which case it still seems reluctant, in my experience, to comply) is hard-coded, though of course the form that response takes depends on what GPT has learned from its training data and interactions with users.

Sequences of words occur in a wide variety of contexts. Conversations seem to be the most relevant type of context, since GPT plays the role of an interlocutor. But then again, it's virtually certain that GPT was trained, in part, on non-conversational linguistic data. If it were simply predicting the next word based on the frequencies with which, in its training data, various words follow the words just input to it, sometimes it would likely do nothing but continue the user's own train of thought, instead of appropriately responding, imparting knowledge, and offering help. So why does GPT mimic conversational, helpful behavior rather than other behaviors involving stringing words together? For now I'll let the mystery be. But perhaps the biggest mystery of all, which I'm not content to leave unsolved, is the subject of the next section.

The Curse of Dimensionality

One of the most mathematically and computationally useful ways to understand the probability of a word w's coming next is in terms of the actual frequency with which w follows a given word sequence s. That frequency is the number of times w follows s in GPT's training data, divided by the total number of times s occurs in those data. But for reasons I'll explain later on in this section, as useful as this straightforward approach to calculating probabilities is, it doesn't seem good enough for the purposes of language modeling.

Let c be the training corpus of our language model (henceforth "LM"), ℓ a natural language like English (as opposed to a formal language like Java, Python, or first-order logic), V the vocabulary of ℓ (i.e., the set of distinct words[2] in ℓ), and S the set of all sequences consisting solely of words in V. To be clear, neither c nor ℓ is meant to be a set like V or S, even though we shall use the locution "d in c" to speak of some datum d in our LM's training sample. For all $s \in S$ and $w \in V$, let the sequence $s + w$ be s followed by w. I shall call a subsequence s_{sub} of a sequence $s_{\text{full}} \in S$ an "unbroken subsequence" if and only if any two consecutive terms of s_{sub} are consecutive terms of s_{full} (lying between the positions in s_{full} corresponding to s_{sub}'s endpoints). Assume, unrealistically, that there's a proportionality constant k such that, for every word $w \in V$ and every sequence s in c, how often w has followed s in the use of ℓ in recent years exactly equals k times how often w follows s in c. Let w_1 and w_2 be words in c. These assumptions entail the following. If, in the contemporary use of ℓ, $s + w_1$ is more common than $s + w_2$, it's more common in c as well. Relatedly, it follows that the ratio between the two words' frequencies is the same in both c and the contemporary use of ℓ. So the training data accurately represent the relative probabilities of w_1 and w_2, conditional on s's having come before[3]—that is, the ratio $P(w_1|s)/P(w_2|s)$ deducible from the data is the actual ratio, despite the limitations of the training sample. The sample is thus, in large measure, representative.

Even so, the LM may not be prepared to respond to any old prompt a user might enter, in anything like the way your average human speaker of ℓ can respond to what another speaker says. If the LM is to be ready for that, it needs to be disposed to respond appropriately to novel sentences, paragraphs, etc., i.e., sequences in S which either have never occurred in the recent use of ℓ or whose instances are unrepresented in the training data c. But how can the LM be so disposed? If, for all $w \in V$ and $s \in S$, the LM equates $P(w|s)$ with

$$\frac{\text{number of times } s + w \text{ occurs in } c}{\text{number of times } s \text{ occurs in } c},$$

then the LM will conclude $P(w|s)$ is undefined if s never shows up in c. So the model won't be able to assign a higher probability to any possible next word than to any other. In other words, the LM won't be able to guess what the next word is, much less make a *good* prediction; all continuations of s will effectively be equally likely from the LM's standpoint. That's a problem. More to the point, it's a problem that we know the developers of LMs have already solved, considering how reliably the best LMs today give proper responses to our queries. The question is how the developers accomplished this, if LMs truly are just complicated next-word predictors.

Admittedly, a number of authors influential in the AI literature seem to have a subtly different notion than I do of the probabilistic formula n-grams are based on. If you understand the probabilities like they do, you might not see the issue yet. (As I shall discuss momentarily, some of these authors have noticed how the issue I raise crops up, in a slightly different form, even in their statistical framework.) Let's say s is a sequence of $n-1$ words. These researchers talk as though n-gram LMs equate $P(w|s)$ with

$$\frac{\text{number of times } s+w \text{ occurs in } c}{\text{number of instances in } c \text{ of all sequences in } c \text{ of } n \text{ words}}.$$

Of course, if s never occurs in c, but some sequence of n words does, then the denominator here is nonzero. But the numerator is zero; it's also zero when s occurs in c but $s+w$ doesn't. So we end up estimating that the probability is zero, not undefined. But that's still a problem, one which Bengio et al. dub "the curse of dimensionality" in their 2003 article "A Neural Probabilistic Language Model." The paper is largely above my pay grade. Nonetheless, I'll share a quote from p. 1138, in which they touch on the matter at hand:

> n-gram models construct tables of conditional probabilities for the next word, for each one of a large number of contexts, i.e. combinations of the last $n-1$ words ... What happens when a new combination of n words appears that was not seen in the training corpus? We do not want to assign zero probability to such cases, because such new combinations are likely to occur, and they will occur even more frequently for larger context sizes.

In "A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams," Church and Gale describe the curse of dimensionality as "[t]he failure of the maximum likelihood estimator (MLE)" (p. 21). They expound on pp. 21–22:

> Suppose that a particular pair of words occurred r times in a sample of N pairs of the language. The obvious estimator of the probability of the pair is r/N ... The problem is that most possible pairs will not occur in a given sample ... Therefore, most of the possible pairs have an observed frequency $r=0$. However, as our example shows, we wish to take products and ratios of the estimated probabilities, and it will often

> happen that neither of two alternative bigrams has been seen in the training text. In this case, the MLE would not determine a likelihood ratio, as both probabilities would be estimated at zero.

Something I'd like to mention, in passing: At first blush, it strikes me as misguided to use, in a next-word prediction algorithm, the formula Bengio et al., Church, and Gale appear to say n-gram models depend on. That is, even if that formula and mine are equally cursed dimensionality-wise, the former seems to track the wrong probability distribution altogether. What we should be predicting is the word that makes the most sense in the context, regardless of how rarely that context arises. But if s is an unusual context, whereas n-grams are ubiquitous, then dividing the number of occurrences of $s + w$ by the number of n-gram occurrences will yield a vanishingly small probability estimate–*even if* w follows s 100% of the times that s occurs! Maybe this doesn't skew the probability distribution, because the number of instances of *every* n-gram is divided by the total number of n-gram instances. Perhaps everything evens out in the final analysis. Just seems fishy to me. Feels like this approach would cause the LM, when faced with an atypical context, to predict the next word based on less unusual subsequences of the unusual context, or even based on the frequencies of words considered individually, apart from context. And a lot of predictively useful contextual information could be ignored in the process.

Anyway, back to the topic of this section. The fundamental difficulty is ensuring our LM can aptly answer queries that don't match any of the word sequences it's seen before. Human speakers of ℓ reliably comprehend and properly reply to novel sentences because they grasp exceedingly well the rules governing both the construction of sentences in ℓ and the use of individual words in V. But to my knowledge, all LMs have to work with are the frequencies with which words have been used in various contexts in their training data. They cannot infer from these statistical data the syntactic and semantic rules (in their full generality) that speakers of a language follow or how those rules apply in contexts unobserved in training.

You might think it's so absurdly improbable that anyone would say something an LM like GPT has never heard before that, from a practical standpoint, there's hardly any point in fighting this so-called "curse." But a simple mathematical argument can be given that the sentences in S vastly outnumber those in c. What's more, the overwhelming majority of sentences in S are not in c. Here's the argument.

Let our LM be GPT and ℓ be English. A conservative estimate of the number of words in V is $171{,}000$. By calculating the number of *ten-word* sequences in S, we see that the *total* number of sequences in S must be at least $171{,}000^{10} \approx 2.1377706 * 10^{52}$, an enormous number.[4,5] Though OpenAI has not officially revealed the size of GPT-4's training sample, GPT-3 is estimated to have trained on 570 GB of text from Common Crawl. Some give similar estimates of the size of GPT-4's corpus; 570 GB is the largest figure I've seen. About half of Common Crawl's dataset is written in English (which is one reason GPT tends to communicate better in English than in other natural languages). That means English text would make up roughly 285 of the 570 GB of textual training data.

Five letters is the average length of an English word, but for the sake of argument we'll make the generous simplifying assumption that every English word is a mere four letters long (just long enough that there are over $171{,}000$ distinct letter sequences/possible words of that length, given a 26-letter alphabet[6]). How many ten-word sequences can you fit in 285 GB, if your $171{,}000$-word vocabulary consists entirely of four-letter words? Well, the number of *words* you can fit in 285 GB is

$$(285\ \cancel{\text{GB}}) \left(\frac{10^9\ \cancel{B}}{1\ \cancel{\text{GB}}}\right) \left(\frac{1\ \cancel{\text{letter}}}{1\ \cancel{B}}\right) \left(\frac{1\ \text{word}}{4\ \cancel{\text{letters}}}\right) = 7.125 * 10^{10}\ \text{words}.$$

Now, if a single sequence s of words takes up n GB, then (i) those n GB can simultaneously contain all the unbroken[7] subsequences of s, and (ii) the number of unbroken subsequences of s is the maximum number of sequences (of the relevant, unbroken sort) that n GB can contain. Consequently, 285 GB can contain as many ten-word sequences as there are unbroken, ten-word subsequences of a sequence of $7.125 * 10^{10}$ words. Of course, a sequence of one or more words inevitably contains more words than unbroken, ten-word subsequences. Therefore, seeing as the number $2.1377706 * 10^{52}$ of ten-word sequences in S is far larger than the estimated maximum number $7.125 * 10^{10}$ of words in GPT's training data, it is likewise far larger than the number of unbroken, ten-word subsequences in those data. Indeed, $2.1377706 * 10^{52}$ is well over twice as big as either of the two numbers I just compared it to. Thus, far fewer ten-word sequences are in c than are in S but not in c.

By the same logic, no matter what sequence length we choose, the sequences of that length in S greatly outnumber those of that length in c.

Good-Turing Frequency Estimation

So the formulas above aren't good enough for programming an LM to reply suitably to input that differs, however slightly, from every sample input—every bit of the training data. How can an LM handle contexts of this kind, if not via the MLE or any formula given thus far?

One idea is to attune the LM to how frequently words have various frequencies, not merely to what frequencies words have. This is the idea behind Good-Turing estimation, a technique Good originally proposed in the 1953 paper "The Population Frequencies of Species and the Estimation of Population Parameters." Allow me to state more precisely the basic procedure, illustrating in broad brush strokes, as I go, how it might be applied in language modeling.

For any frequency r with which some word follows some sequence in training corpus c, we can identify n_r, the number of words that each follow that sequence r times in c.[8,9] We may describe n_r as "the frequency of the frequency r." Once we determine n_r for each r, we may proceed to line up all the frequencies of frequencies, forming a series of numbers that will be immensely useful for approximating the probability that a word w will follow an unobserved sequence s in the use of language ℓ (if and when some user of ℓ forms s out of words in vocabulary V). The series enables us to estimate r^*, the number of times a given word that follows a sequence r times in c would follow that sequence in c, were c a maximally representative sample of text written in ℓ. Note that r^* is not merely a variable but a variable paired with an operator: *. Unlike the familiar $*$ operator, however, this one takes only one argument, namely r. The equations below reveal how the * operation works—how it modifies its input to produce an output.

Once we have r^* and the number N of instances in c of a sequence s of words, we can obtain the approximate value of q_r, the frequency with which a word w follows s in the recent use of ℓ (not just in c, a mere sample of the contemporary use of ℓ). We do so by estimating the expectation of q_r.

Here's the gist of the concept of the expectation of q_r, as I understand it. If you were to turn back time to the very beginning of the "contemporary" use of ℓ, whatever random word choices people had made since that initial time would have to be made again (though the discursive contexts in which some of those choices were made might not arise again, due to changes in prior word choices). They might choose different words that time around, so we might see a change in the ratio q_r of the number of cases of $s + w$ to that of cases of s. Were you then to iterate this reversal of the final segment of ℓ's history infinitely many times, you could basically just average the values q_r takes in the infinitely many

iterations to get a decent sense of the value q_r most likely takes in any given iteration. (Analogy: Were you to flip a coin infinitely many times, you'd expect the ratio of heads outcomes to total outcomes to approach $1/2$ in the long run. So if you were to guess how frequently a coin would come up heads in some sequence of coin flips, the most reasonable prediction would be half the total number of flips, if you didn't know the number of flips in advance.) That (weighted) average is known as the expectation of q_r.

We can take this average to be the probability, given the occurrence of s, that word w will follow in the present and near-future use of ℓ. And fortunately, with r^* and N at our disposal, we have a way to estimate this average. For Good proved Turing's formula to the effect that

$$\mathscr{E}(q_r) = r^*/(N+1) \approx r^*/N,$$

where $\mathscr{E}(v)$ denotes the expectation, or expected value, of a random variable v such as q_r, and where

$$\begin{aligned} r^* &= (r+1)\frac{\mathscr{E}_{N+1}(n_{r+1}|H)}{\mathscr{E}_N(n_r|H)} \\ &\approx (r+1)\frac{n_{r+1}}{n_r}. \end{aligned}$$

H is some hypothesis stating the frequencies with which, in the contemporary use of ℓ, all words in vocabulary V follow all sequences in S.

Good was interested in estimating how common various species are in a population p. In that domain of inquiry, the number n_0 tends to be unknown, making it challenging to calculate 0^* and, by extension, $\mathscr{E}(q_0)$. You *cannot* tell, just by looking at a (limited) sample of p, how many species in p occur zero times in that sample. To infer how many are unrepresented, you would need to be aware of how many species are in p, not just in the sample. But (depending on the population) neither the sample nor common knowledge provides you with that information. Population frequencies of species differ from frequencies of words in that respect. That is, we *do* know roughly how many words there are in our language.[10] Thus, we can find out how many words in our language

(a) are unrepresented in a sample or

(b) never follow sequence s in the sample,

by subtracting

(a') the number of words in the sample or

(b') the number of words that follow s in the sample

from the number of words in the language. We subtract (a') to get the number of words that meet condition (a). If instead we want the number that meet (b), we subtract (b'). Whichever of those two numbers we're after, it makes sense to call it n_0, depending on how we define the subscript r (for which 0 is substituted). If we define r as the number of times a word occurs in a sample, n_0 is the number of words of which (a) is true. If we define r as the number of times a word follows s in the sample, n_0 is the number of words of which (b) is true. I favor the latter definition.

It must be said that the Good-Turing method is not, to my knowledge, used in the development of *large* language models. (Not today, at any rate). It's one of the first approaches ever taken to predicting words in unfamiliar contexts—one that was successful and widely used in language modeling for a time, and one that has left an indelible mark on the AI literature. That is why I have taken an interest in it; at the moment I care more about how, in principle, it is feasible to predict words in unfamiliar contexts than about how this is accomplished in practice nowadays. And I believe anything we can learn about the humble origins of LLMs is bound to shed light on the way LLMs work today. We may even discover, eventually, that the rational basis for the Good-Turing method underlies the methodology of contemporary language modeling.

But what is that basis? To find out, we can read and reproduce Good's derivation of Turing's formula. I have no intention of laying out the full proof in this post. I only wish to establish one theorem the proof presupposes, which caught my attention.

One Theorem We'll Need

As Good begins his proof of Turing's formula, near the top of p. 240 of "The Population Frequencies of Species and the Estimation of Population Parameters," Good states, "We recall the theorem that an expectation of a sum is the sum of the expectations." No, Good, I do *not* recall that theorem, nor do I see why it would hold. So let's prove it, at least for an arbitrary pair of discrete random variables (r.v.'s) X and Y—prove that $\mathscr{E}(X+Y) = \mathscr{E}(X) + \mathscr{E}(Y)$.

Below I have made some minor revisions to a proof of $\mathscr{E}(X+Y) = \mathscr{E}(X) + \mathscr{E}(Y)$, which I found on Mathematics Stack Exchange. Despite how small the edits are, coming up with them made a world of difference to my comprehension of the flow of reasoning. My version of the proof is broken down into steps numbered (1)–(10).

The expectation of $X + Y$… What could that be? Expectations are only defined for functions, to my knowledge. But isn't $X + Y$ just the sum of two variables —well, r.v.'s? How is that sum a function? Technically, r.v.'s, including X and Y, are functions. The question though is what makes the sum of two such functions a further function. We'll answer this shortly.

$$\mathscr{E}(X + Y) = \mathscr{E}(+(x, y)) \tag{1}$$
$$= \sum_{x,y} [+(x, y)p(x, y)] \tag{2}$$

The expectation $\mathscr{E}(f(x_1, x_2))$ of a function f in two discrete variables x_1 and x_2 is defined as $\sum_{x_1,x_2} f(x_1, x_2)p(x_1, x_2)$. This licenses the inference from (1) to (2), given the fact that $+(x, y)$ is a function in discrete variables x and y.

The notation $+(x, y)$ may puzzle you. This is just my unconventional translation of $x + y$ into function notation. It may be an abuse of notation, but it makes explicit the fact that the (binary) addition operator $+$ is a function, as is any operator. It's specifically a function that takes two inputs—no more, no fewer—hence the term "binary." The inputs in question are the addends, typically written to the left and right of the + sign. But I've decided to move these inputs, x and y, to the positions typically occupied, in function notation, by the independent variables of a binary function: between parentheses to the right of the function symbol. Normally a function is expressed as something like $f(x, y)$, with a letter (in this case, f) denoting the function itself. $+$ is obviously not a letter. Nonetheless, you can understand $+(x, y)$ as an expression of the form $f(x, y)$ (with $+$ being our f) that specifically stands for the addition function.

This is no mere notational or verbal trickery. Kripkensteinian doubts notwithstanding, $+$ truly does denote a function, as it receives inputs, and every time it takes in the same set of numbers, it spits out the same number. $1 + 1$ equals 2 and nothing else, $30 + 78$ equals 108 and nothing else, and so on ad infinitum. Indeed, we can even write an equation $f(x, y) = x + y$ which all but explicitly states that $+$ is a function with inputs x and y. One might interpret the equation as saying the output of f for the particular inputs x and y is the sum of x and y. But there's nothing stopping us from using x and y in this equation to represent any two values X and Y can take, rather than a particular pair of numbers. Better yet, we can replace x and y with X and Y to refer to the r.v.'s themselves rather than any specific values they may take. When we do that, we get the expression $X + Y$ with which we began, and we see it does indeed signify a binary function $f(X, Y)$. If the equation is to capture the function,

something on the right side has to be equivalent to $f()$ on the left side, and it's clearly not X and/or Y alone. One possible counterpart remains: $+$, which, like $f()$, unites X and Y, determining how they're combined to yield a new value.

$p(x, y)$ may be a bit confusing as well. According to Wolfram MathWorld's definition of expected value, $p(x, y)$ is a probability mass function (PMF).[11] Presumably it outputs a probability, but the probability of what? The probability of x and y? What would that even mean? Perhaps the proof on Mathematics Stack Exchange can shed some light on the subject. There, in place of $p(x, y)$, we find the function notation $P(X = x, Y = y)$. It would appear, then, that $p(x, y)$ and $P(X = x, Y = y)$ are interchangeable in the formula for expectation ($\mathscr{E}$). If that's right, the probability that $p(x, y)$ represents must be the probability that $X = x, Y = y$—where $X = x$ means X takes value x, and $Y = y$ means Y takes value y. But what does the comma between $X = x$ and $Y = y$ represent? One can't help but wonder as well how the two functions can be interchangeable despite the evident differences between the types of input they take: a list of comma-separated numbers in the case of $p(x, y)$ and a list of comma-separated equations in the case of $P(X = x, Y = y)$.

There are two possibilities. One is that, despite appearances, the functions are the same. The types of input they take do not differ. The other possibility is that the functions are different, but p's output for any given pair of inputs x and y equals P's output for the input(s) $X = x, Y = y$. The first possibility may sound absurd now. But it sounds less absurd once you realize that the input $X = x, Y = y$ is a set, that each input to $p(x, y)$ can be understood as an ordered pair, and that ordered pairs are definable as sets.

Before I explain how $X = x, Y = y$ denotes a set in $P(X = x, Y = y)$, I should say a few words about what the simpler input expression $X = x$ represents. It's shorthand for $\{X = x\}$, which is in turn shorthand for $\{\omega \in \Omega | X(\omega) = x\}$. What we have here is a set—a funny-looking set, but a set nonetheless. This is what's known as "set-builder notation." We read $\{\omega \in \Omega | X(\omega) = x\}$ as "the set of all ω in Ω such that X of ω equals x." More generally, $\{\nu \in N | \Phi(\nu)\}$ signifies the set of all things ν in the set N indicated to the left of the vertical bar that satisfy the condition $\Phi(\nu)$ to the right of the bar. In the case at hand, our set N is the sample space Ω—the set of possible outcomes of experiments involving random variable X. Further, $X(\omega) = x$ is our condition $\Phi(\nu)$. So $X = x$ is the set containing every possible outcome ω whose being input to X yields the output x; in measure theory this set is called the *event* that X takes value x.[12] Nota bene: Depending on the context, X denotes either a general variable or a function with domain Ω and range $\mathcal{X} \subseteq \mathbb{R}$. General variable X takes (i.e., equals) value x if and only if function X outputs x.

$P(X = x, Y = y)$ can be read as "the probability that $X = x, Y = y$." And that comma is to be understood the same way as the comma in the sentence "Let $X = x, Y = y$"—as a symbol for conjunction, equivalent in meaning to the word "and." So, the input expression $X = x, Y = y$ is short for $\{X = x \wedge Y = y\}$, which is short for $\{\omega \in \Omega | X(\omega) = x \wedge Y(\omega) = y\}$, where $\wedge$ means "and."

So we've established that $P(X = x, Y = y)$ takes a set as input. It remains to be seen whether the domains of $P(X = x, Y = y)$ and $p(x, y)$ include all and only the same sets, and thus are one and the same domain (as they must be if p and P are to be the same function). $p(x, y)$ denotes a function in two variables, specifically a *joint probability mass function*. One may think of any function in two variables x and y as having either

i. a single domain whose elements are ordered pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the ranges of values which x and y, respectively, can take, or

ii. two domains $\mathcal{X}$ and $\mathcal{Y}$ (defined as above).

For the time being we may set aside case ii, as that directly contradicts the thesis we're entertaining: that $p(x, y)$ has the same domain $P(X = x, Y = y)$ has (and no others), of which there is only one.

If we take $p(x, y)$ to have a domain of type i, we should interpret $p(x, y)$ as shorthand for $p((x, y))$. Moreover, we in that case take the domain of $p(x, y)$ to be some subset, proper or improper, of Cartesian product $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the ranges of our discrete r.v.'s X and Y. The domain could for instance be

$$\{(x, y) \in \mathcal{X} \times \mathcal{Y} | P(X = x, Y = y) > 0\},$$

the set of all and only ordered pairs (x, y) in $\mathcal{X} \times \mathcal{Y}$ such that the probability is nonzero that $X = x, Y = y$. But let's make things easier for ourselves by treating the entire Cartesian product $\mathcal{X} \times \mathcal{Y}$ as our domain. Let's also define Ω as our *sample space*, the set of possible outcomes of experiments with variables X and Y. If $P(X = x, Y = y)$ is to be a probability measure, its domain must be a σ-algebra on Ω, which is a special kind of set of subsets of Ω. Seeing as the subsets of Ω are events, the domain of $P(X = x, Y = y)$ must consequently be a set of events, or *event space*, $\mathfrak{F}$. So as long as our event space $\mathfrak{F}$ can be $\mathcal{X} \times \mathcal{Y}$, the functions $p(x, y)$ and $P(X = x, Y = y)$ can have the same domain.

Recall that $\mathcal{X} \times \mathcal{Y}$ includes all and only ordered pairs (x, y) s.t. $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. It was also mentioned that ordered pairs are definable as sets. Let's take a look at one set-theoretic definition of an ordered pair.

$$(x, y) := \{\{x\}, \{x, y\}\}$$

By this definition, if $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is to be an event in $\mathfrak{F}$, then $\{x\}$ and $\{x, y\}$ must be possible outcomes of experiments involving variables X and Y. I see no reason that such experiments couldn't have those outcomes. So it may be that everything in $\mathcal{X} \times \mathcal{Y}$ is in $\mathfrak{F}$. Is everything in $\mathfrak{F}$ in $\mathcal{X} \times \mathcal{Y}$? Well, $\mathfrak{F}$ must have certain elements to qualify as a σ-algebra on Ω, and $\mathcal{X} \times \mathcal{Y}$ is likely to lack some of those. For example, Ω itself must be in $\mathfrak{F}$. But assuming

1. $\mathcal{X}$ contains at least three values, and
2. every singleton set containing an element of $\mathcal{X}$ is an outcome in Ω,

Ω is a set of three or more elements. But no set-theoretic representation of an ordered pair (conforming to the definition above) has more than two elements. So, given suppositions 1 and 2, $\mathcal{X} \times \mathcal{Y}$ does not include Ω like $\mathfrak{F}$ does. Therefore, very probably, p and P are not the same function.

The same conclusion follows should we choose (as we certainly may) to view $p(x, y)$ as having two domains of numbers rather than one of ordered pairs, assuming P is a probability measure. A probability measure has only one domain: a set of subsets of a sample space. So in that case we must distinguish between the functions p and P.

But regardless of what domain(s) $p(x, y)$ has, it's true that $p(x, y) = P(X = x, Y = y)$. We still have that p is a function wherein $(x, y) \mapsto P(X = x, Y = y)$. That's how the joint probability mass function p is related to the probability measure P. For more on this relationship, see the appendix.

With that out of the way, let's get back to our derivation of the theorem that $\mathscr{E}(X + Y) = \mathscr{E}(X) + \mathscr{E}(Y)$. I consider the next four steps in the proof pretty self-explanatory.

$$\begin{aligned}
\mathscr{E}(X + Y) &= \sum_{x,y} (x + y) p(x, y) && (3)\\
&= \sum_{x} \sum_{y} (x + y) p(x, y) && (4)\\
&= \sum_{x} \sum_{y} [p(x, y)x + p(x, y)y] && (5)\\
&= \sum_{x} \sum_{y} p(x, y)x + \sum_{x} \sum_{y} p(x, y)y && (6)
\end{aligned}$$

I'll explain briefly. We can move from (3) to (4) thanks to the fact that $\sum_{x,y}$ is shorthand for $\sum_x \sum_y$. (4)–(5) is allowed because multiplication is distributive. In (5)–(6) we break a single sum of sums (henceforth "nested sum") of $p(x,y)x$ + $p(x,y)y$ over ranges $\mathcal{X}$ and $\mathcal{Y}$ into two such sums, one for each of the addends $p(x,y)x$ and $p(x,y)y$. This is just an application of the rule that $\sum_x \sum_y (a+b) = \sum_x \sum_y a + \sum_x \sum_y b$, for arbitrary expressions a and b equal to real numbers. The rule holds whether or not expression a or b contains x or y.

Just to be sure this rule's intuitive appeal does not mislead, let's work through a few examples of the rule in action. We'll stipulate beforehand that, in each of these cases, the ranges of r.v.'s X and Y are $\mathcal{X} = \{0,1\}$ and $\mathcal{Y} = \{2,3\}$, respectively.

Example 1: Let $a = x, b = y$. Then we have:

$$\sum_x \sum_y (x+y) = \sum_x \sum_y x + \sum_x \sum_y y$$
$$\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} (x+y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} x + \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} y$$
$$\sum_{x=0}^{1} \sum_{y=2}^{3} (x+y) = \sum_{x=0}^{1} \sum_{y=2}^{3} x + \sum_{x=0}^{1} \sum_{y=2}^{3} y$$
$$\sum_{x=0}^{1} [(x+2)+(x+3)] = \sum_{x=0}^{1} (x+x) + \sum_{x=0}^{1} (2+3)$$
$$(2*0+5)+(2*1+5) = [(0+0)+(1+1)]+[(2+3)+(2+3)]$$
$$12 = 12 \checkmark$$

Example 2: Let $a = x, b = 1$. That gives us:

$$\sum_x \sum_y (x+1) = \sum_x \sum_y x + \sum_x \sum_y 1$$
$$\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} (x+1) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} x + \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} 1$$
$$\sum_{x=0}^{1} \sum_{y=2}^{3} (x+1) = \sum_{x=0}^{1} \sum_{y=2}^{3} x + \sum_{x=0}^{1} \sum_{y=2}^{3} 1$$
$$\sum_{x=0}^{1} [(x+1)+(x+1)] = \sum_{x=0}^{1} (x+x) + \sum_{x=0}^{1} (1+1)$$
$$(2*0+2)+(2*1+2) = [(0+0)+(1+1)]+[(1+1)+(1+1)]$$
$$6 = 6 \checkmark$$

Example 3: Let $a = 1, b = y$. From this we obtain:

$$\sum_x \sum_y (1+y) = \sum_x \sum_y 1 + \sum_x \sum_y y$$
$$\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} (1+y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} 1 + \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} y$$
$$\sum_{x=0}^{1} \sum_{y=2}^{3} (1+y) = \sum_{x=0}^{1} \sum_{y=2}^{3} 1 + \sum_{x=0}^{1} \sum_{y=2}^{3} y$$
$$\sum_{x=0}^{1} [(1+2)+(1+3)] = \sum_{x=0}^{1} (1+1) + \sum_{x=0}^{1} (2+3)$$
$$7+7 = [(1+1)+(1+1)] + [(2+3)+(2+3)]$$
$$14 = 14 \checkmark$$

For the time being I've satisfied myself that our rule has no counterexamples. Let's move on.

Getting from (6) to (7) is a bit trickier, in my opinion. Recall that (6) reads

$$\mathscr{E}(X+Y) = \sum_x \sum_y p(x,y)x + \sum_x \sum_y p(x,y)y.$$

Now, here's (7) and, while we're at it, (8).

$$\mathscr{E}(X+Y) = \sum_x \sum_y p(x,y)x + \sum_y \sum_x p(x,y)y \tag{7}$$
$$= \sum_x \left[x \sum_y p(x,y) \right] + \sum_y \left[y \sum_x p(x,y) \right] \tag{8}$$

Once we've derived (7), (8) will follow as a matter of course. But where does (7) come from, exactly? Recall that (6) reads:

$$\mathscr{E}(X+Y) = \sum_x \sum_y p(x,y)x + \sum_x \sum_y p(x,y)y$$

So to get from (6) to (7), all we need is the principle that $\sum_x \sum_y f(x,y) = \sum_y \sum_x f(x,y)$ for an arbitrary function $f(x,y)$ (two such functions being $p(x,y)x$ and $p(x,y)y$). But to me this is even less intuitive than

the rule we've already proposed and tested against a few examples (not that the two rules are in competition with each other). So this one stands in greater need of justification.

Let's try a reductio ad absurdum, in hopes of achieving full generality. That is, let's assume $\sum_x \sum_y f(x,y) \neq \sum_y \sum_x f(x,y)$, the opposite (negation) of what we want to prove. We'll proceed to show that this assumption entails a contradiction. Then we'll have demonstrated in a roundabout way that, contrary to our supposition, $\sum_x \sum_y f(x,y) = \sum_y \sum_x f(x,y)$. Otherwise, a contradiction would be true, which is absurd/impossible!

$$\sum_x \sum_y f(x,y) \neq \sum_y \sum_x f(x,y)$$

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} f(x,y) \neq \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} f(x,y)$$

$$\sum_{x=x_1}^{x_m} \sum_{y=y_1}^{y_n} f(x,y) \neq \sum_{y=y_1}^{y_n} \sum_{x=x_1}^{x_m} f(x,y)$$

$$\sum_{x=x_1}^{x_m} [f(x,y_1) + \cdots + f(x,y_n)] \neq \sum_{y=y_1}^{y_n} [f(x_1,y) + \cdots + f(x_m,y)]$$

Consequently,

$$[f(x_1,y_1) + \cdots + f(x_1,y_n)] + \cdots + [f(x_m,y_1) + \cdots + f(x_m,y_n)]$$

does not equal

$$[f(x_1,y_1) + \cdots + f(x_m,y_1)] + \cdots + [f(x_1,y_n) + \cdots + f(x_m,y_n)].$$

At this juncture we need to think carefully about these two, supposedly unequal nested sums, particularly about each sequence of addends/terms enclosed in a pair of square brackets. Let σ_1 denote the first nested sum and σ_2 the second. To be clear, we're defining σ_1 and σ_2 such that

$$\sigma_1 = [f(x_1,y_1) + \cdots + f(x_1,y_n)] + \cdots + [f(x_m,y_1) + \cdots + f(x_m,y_n)]$$

and

$$\sigma_2 = [f(x_1,y_1) + \cdots + f(x_m,y_1)] + \cdots + [f(x_1,y_n) + \cdots + f(x_m,y_n)].$$

We know that, if we take the *first* term of each sequence of terms enclosed in square brackets in σ_1, we can use those terms to form the sequence

$$f(x_1,y_1), f(x_2,y_1), f(x_3,y_1), \ldots, f(x_{m-2},y_1), f(x_{m-1},y_1), f(x_m,y_1).$$

Whaddayaknow?!?—that's the first sequence of terms in σ_2! That is, it's what you get when you remove the brackets from

$$[f(x_1, y_1) + \cdots + f(x_m, y_1)],$$

substitute commas for the $+$ signs, and make explicit a few more of the terms.

So at least σ_1 and σ_2 have those m terms in common. Can we find in σ_1 the second sequence from σ_2?

As a matter of fact, we can! The sequence in question is

$$f(x_1, y_2), f(x_2, y_2), f(x_3, y_2), \ldots, f(x_{m-2}, y_2), f(x_{m-1}, y_2), f(x_m, y_2).$$

We can form this same sequence by taking the *second* term of each sequence in σ_1. First we took the first term of each, and now we're taking the second. If we iterate this process $n - 2$ more times, we'll reproduce every sequence of terms in σ_2, exclusively using terms of σ_1. We'll then have established (in fact we already have, using our imagination) that every term of σ_2 is a term of σ_1. But what about vice versa? Is every term of σ_1 a term of σ_2? If so, σ_1 and σ_2 are sums of exactly the same terms. That would mean $\sigma_1 = \sigma_2$—contrary to the conclusion we derived from our assumption for reductio.

It turns out that there is a parallel procedure we can follow to construct each sequence of terms in σ_1, exclusively using terms of σ_2. The first sequence in σ_2 is just the sequence of the first terms of all the sequences in σ_1. The second sequence in σ_2 is the sequence of the second terms of all the sequences in σ_1. And so on, all the way up to the n^{th} and final sequence in σ_2.

In conclusion, σ_1 and σ_2 have exactly the same terms; neither has a term the other lacks. So $\sigma_1 = \sigma_2$. But we had already concluded that $\sigma_1 \neq \sigma_2$. Of course, that $\sigma_1 = \sigma_2$ directly contradicts that $\sigma_1 \neq \sigma_2$. So it must be that in actuality $\sum_x \sum_y f(x, y) = \sum_y \sum_x f(x, y)$, contrary to our initial assumption for the sake of argument. Quod erat demonstrandum! This concludes our subproof.

We can now be certain of the truth of (7):

$$\mathscr{E}(X + Y) = \sum_x \sum_y p(x, y)x + \sum_y \sum_x p(x, y)y$$

And clearly (8), which reads

$$\mathscr{E}(X+Y) = \sum_x \left[x \sum_y p(x,y) \right] + \sum_y \left[y \sum_x p(x,y) \right],$$

follows from (7) by the definition of Σ and the distributivity of multiplication. The remainder of the proof is written below.

$$\begin{aligned} \mathscr{E}(X+Y) &= \sum_x p(x)x + \sum_y p(y)y && (9) \\ &= \mathscr{E}(X) + \mathscr{E}(Y) && (10) \end{aligned}$$

Equation (10) is our theorem, so we could stop here. But I feel I should elaborate on how we got from (8) to (9), in case it's unclear. Consider the first term on the right side of the equation in (8):

$$\begin{aligned} \sum_x \left[x \sum_y p(x,y) \right] &= \sum_x \left[x \sum_{y=y_1}^{y_n} p(x,y) \right] \\ &= \sum_x x(p(x,y_1) + p(x,y_2) + \cdots + p(x,y_n)) \\ &= \sum_x x(p(x)p(y_1|x) + \cdots + p(x)p(y_n|x)) \\ &= \sum_x p(x)x(p(y_1|x) + p(y_2|x) + \cdots + p(y_n|x)) \\ &= \sum_x p(x)x(1) \\ &= \sum_x p(x)x \end{aligned}$$

Note that we use conditional probabilities to calculate the joint probabilities of pairs of values x, y that X and Y might take, rather than simply multiplying the unconditional probabilities of the possible values. This way we leave room for the possibility that X and Y are dependent.

By the same reasoning we can show that

$$\begin{aligned} \sum_y \left[y \sum_x p(x,y) \right] &= \sum_y p(y)y(p(x_1|y) + p(x_2|y) + \cdots + p(x_m|y)) \\ &= \sum_y p(y)y(1) \\ &= \sum_y p(y)y. \end{aligned}$$

You might wonder why $p(x_1|y) + p(x_2|y) + \cdots + p(x_m|y)$ and $p(y_1|x) + p(y_2|x) + \cdots + p(y_n|x)$ each equal 1. Answer: For the same reason that $p(x_1) + p(x_2) + \cdots + p(x_m)$ and $p(y_1) + p(y_2) + \cdots + p(y_n)$ each equal 1. The numbers $x_1, x_2, \ldots, x_m$ represent all the values X can take, and the numbers $y_1, y_2, \ldots, y_n$ represent all the values Y can take. In other words, X has to equal one of $x_1, x_2, \ldots, x_m$, and Y has to equal one of $y_1, y_2, \ldots, y_n$. It stands to reason then that with probability 100%, be it conditional or unconditional, either of the two r.v.'s takes some value or other in the corresponding range. But that can only be the case if the probabilities of the individual values in the range add up to 100%, which is of course equal to 1.

So we have obtained

$$\sum_x \left[x \sum_y p(x, y) \right] = \sum_x p(x)x$$

as well as

$$\sum_y \left[y \sum_x p(x, y) \right] = \sum_y p(y)y.$$

Synthesizing these two results, we get

$$\sum_x \left[x \sum_y p(x, y) \right] + \sum_y \left[y \sum_x p(x, y) \right] = \sum_x p(x)x + \sum_y p(y)y.$$

Appendix

Let $D(x, y)$ be the joint cumulative distribution function (CDF) of discrete random variables X and Y. Just as the probability density function (PDF) corresponding to a continuous CDF is the derivative of that CDF, the PMF corresponding to discrete CDF $D(x, y)$ is the jump of $D(x, y)$ at (x, y). This jump is denoted by $p(x, y)$. A function must be continuous to be differentiable—hence the need to identify probability masses with something other than derivatives in the discrete case.

What's the jump of a discrete joint CDF at a point? It helps to know what a joint CDF is first. I won't give a definition (i.e., a set of individually necessary and jointly sufficient conditions), but I will explain what you need to know about the concept, which luckily enough I know. A joint CDF is a function $D(x_1, x_2, \ldots, x_n)$ in multiple variables $x_1, \ldots, x_n$ such that, if each x_i is a value

a distinct random variable X_i can take, the function outputs the probability of the conjunctive event that X_1 takes a value $\alpha \leq x_1$, X_2 a value $\beta \leq x_2$, ..., and X_n a value $\zeta \leq x_n$. That is,

$$\begin{aligned} D(x_1, x_2, \ldots, x_n) &= P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n) \\ &\neq P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n). \end{aligned}$$

So the joint CDF is not the joint PMF $p(x_1, x_2, \ldots, x_n)$, but the two are intimately related. Intuitively, the probability that an r.v. X takes a value no greater than x is no less than the probability that it takes value x. The same goes for the probability that each of multiple r.v.'s X_i takes a value no greater than x_i; that probability is no less than the probability that each takes value x_i. But the former probability could easily be greater than the latter. To be exact,

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

is equal to

$$P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n) - P(X_1 < x_1 \vee X_2 < x_2 \vee \cdots \vee X_n < x_n).$$

In English: we can obtain the probability that, collectively, the variables X_i each take value x_i by subtracting the probability that some X_i or other takes a value less than x_i from the probability that each takes a value less than or equal to x_i. This is because, given that no X_i exceeds x_i, there are two mutually exhaustive and exclusive possibilities:

1. Each X_i is equal to x_i.
2. Some X_i or other is less than x_i.

So the probabilities of these possibilities add up to the probability that no X_i exceeds x_i. Moreover, the likelihood of either possibility is the likelihood of the other's negation—again, assuming that no X_i exceeds x_i. But if we dispense with that assumption, then we must additionally consider how likely it is, in the first place, that no X_i exceeds x_i.

To reiterate: we calculate $p(x, y)$ by subtracting $P(X < x \vee Y < y)$ from $D(x, y)$, with this latter expression being equivalent to $P(X \leq x, Y \leq y)$. We're now well on our way to seeing what the jump of a discrete joint CDF at a point is and why it matters. Observe that $P(X < x \vee Y < y)$ equals

$$P(X < x, Y \leq y) + P(X \leq x, Y < y) - P(X < x, Y < y).$$

Thus, the difference between $P(X \leq x, Y \leq y)$ and $P(X < x \vee Y < y)$ is

$$P(X \leq x, Y \leq y) - [P(X < x, Y \leq y) + P(X \leq x, Y < y) - P(X < x, Y < y)].$$

This is where the jump of $D(x, y)$ at (x, y) makes its entrance. We may define this jump as the difference

$$D(x, y) - [D(x^-, y) + D(x, y^-) - D(x^-, y^-)],$$

which, believe it or not, is the same difference (pun intended) presented immediately prior. This makes a bit of sense, as we already know that $D(x, y) = P(X \leq x, Y \leq y)$, by definition. It remains to be elucidated how $D(x^-, y) = P(X < x, Y \leq y)$, how $D(x, y^-) = P(X \leq x, Y < y)$, and how $D(x^-, y^-) = P(X < x, Y < y)$.

First, some definitions:

$$D(x^-, y) := \lim_{\alpha \to x^-} D(\alpha, y)$$

$$D(x, y^-) := \lim_{\beta \to y^-} D(x, \beta)$$

$$D(x^-, y^-) := \lim_{\substack{\alpha \to x^- \\ \beta \to y^-}} D(\alpha, \beta)$$

Now then, I'll try to give you a rough idea of how these sundry puzzle pieces interlock. It's easier to understand how the jump of a discrete CDF in one variable works than in two variables. We'll start there. Take the CDF $D(x)$ of a discrete random variable X, where the range of X is $\mathcal{X} \subseteq \mathbb{R}$. By the definition of "discrete random variable," $\mathcal{X}$ has countably many elements. So for each value $x \in \mathcal{X}$ aside from the maximum in $\mathcal{X}$, there is a "next" value: the least value in $\mathcal{X}$ greater than x. If the set of values X can take is $\{5, 13, 21\}$, then relative to 5, the next value is 13; relative to 13, it's 21.[13]

Recall that $D(x)$ is the probability that $X \leq x$. As you shift your attention from one value in $\mathcal{X}$ to the next, the probability that X is less than or equal to the value under consideration either increases or remains the same. This is a straightforward consequence of the fact that more values in $\mathcal{X}$ don't exceed the current one than didn't exceed the last. So there's now a greater number of ways X can be less than or equal to the current value. The probabilities of those various ways add up, and each probability is nonnegative. So the overall probability never decreases. On the contrary, it tends to increase.[14] And $D(x)$ increases along with it. That increase in $D(x)$ accompanying the transition from x's taking one value in $\mathcal{X}$ to its taking the next—*that* is the jump of $D(x)$ at x, which is also the probability mass at x.[15] If you want to get technical about it, this jump is $D(x) - D(x^-)$, i.e., $D(x) - \lim_{\alpha \to x^-} D(\alpha)$. So we have the formula

$$P(X = x) = D(x) - \lim_{\alpha \to x^-} D(\alpha),$$

where $P(X = x) = p(x)$.

By more or less the same reasoning, the jump of discrete joint CDF $D(x, y)$ at (x, y) is

$$D(x, y) - \lim_{\alpha \to x^-} D(\alpha, y) - \lim_{\beta \to y^-} D(x, \beta) + \lim_{\substack{\alpha \to x^- \\ \beta \to y^-}} D(\alpha, \beta) = P(X = x, Y = y),$$

where $P(X = x, Y = y) = p(x, y)$.

I leave it to you to make explicit how this follows, should you find the motivation to do so.

Endnotes

1. To be clear, I'm not suggesting that, without exception, GPT predicts that a word will follow the most recent one. Were it to behave that way, its response to a prompt would never end. For it outputs every word it predicts, and once it outputs a word, it predicts what would follow that word. (It doesn't just make one prediction as to what would follow a prompt from the user, but also a distinct prediction of what would follow the sequence for every addition it makes to the sequence.) Its responses are not endless because, sooner or later, it predicts that the hypothetical interlocutor it imitates would stop talking and thus that no word would come next—or, if there would be an additional word, the user would be the one to say it, so GPT has no need to predict it.

2. I leave open whether each of a language's idioms counts as a "word" in its vocabulary.

3. At least, the training data represent these probabilities as accurately as the words' actual frequencies do.

4. Note that a sequence, and even a sentence in particular, can contain repetitions of a word. This is why we simply raise $171{,}000$ to the tenth power; there are $171{,}000$ options for each of the ten words in the sequence, regardless of how many of those ten words we've already selected.

5. Granted, many of the sequences we're counting are not well-formed sentences, paragraphs, or subsentential expressions, since we are considering every possible arrangement of any ten words, without regard for the constraints of grammar or individual words' rules of use. But for every ungrammatical or nonsensical ten-word sequence, there must be a perfectly acceptable eleven-, twelve-, thirteen-, or n-word sequence that we could add to our count.

6. There are a grand total of $26^4 = 456{,}976$ four-letter sequences, but only $26^3 = 17{,}576$ three-letter sequences.

7. We may ignore broken subsequences of s, because one would hope an LM would base its "understanding" of a language exclusively on continuous pieces of text in its training sample, so as to keep track of the correct ordering of words (and characters, for that matter). A broken subsequence of s is a subsequence of s with consecutive terms that are not consecutive terms of

s (at least, not consecutive terms lying between the positions in s corresponding to the endpoints of the subsequence).

8. Note that in calling r a "frequency," we're saying it's a number of occurrences rather than a ratio of such numbers. It's a question of how many whosits, not how many whosits out of n whatsits. In other words, the frequency is not the number of whosits divided by the number of whatsits, though, here, the number of whosits we have in mind just so happens to be the number of whosits associated with n whatsits, rather than the total number of whosits.

9. Alternatively, if we let r be the number of times a *particular* sequence of l words occurs, we can find the number n_r of sequences of l words that each occur r times.

10. However, it is not commonly known how many of the words in the English vocabulary were used during a given period in the history of the English language.

11. The linked page calls it a "probability density function," but technically we should use "mass" rather than "density" when talking about discrete random variables, ones whose ranges' elements have gaps between them (unlike, for instance, the elements of the continuum/set of real numbers).

12. You might think it would make more sense to classify the individual outcomes as events than to classify sets of them as events. But in measure theory probabilities are assigned to sets of outcomes according to how likely it is that the disjunction of those outcomes obtains. Take a standard die. Suppose the numbers you can roll are the possible outputs of our random variable X. So X can take any integer value between 1 and 6, inclusive. This gives us the range $\mathcal{X}$ of X, but what about X's domain Ω, i.e., the sample space? To each number $x \in \mathcal{X}$ there corresponds a side of the die with x dots on it. Let ω_x be the outcome that the side with x dots on it is facing up, after the die is rolled. Then we can take the domain Ω of X to be $\{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}$. The probability of $\{\omega_2, \omega_4, \omega_6\}$ is then that of the event that ω_2, ω_4, or ω_6 is the outcome—in other words, the event that an even number in Ω is rolled. When we conceptualize sets of outcomes this way, as themselves being disjunctive outcomes (though not elements of Ω), they do start to look like events in the familiar sense—like things that can happen.

13. That's not due to the order in which the numbers are written in the set notation; regardless of the order they're written in, it helps to imagine they're arranged in increasing order.

14. Some sources even suggest that, by definition, the overall probability always increases. They say the range of a discrete r.v. X is defined as the set of values x such that $P(X = x) > 0$.

15. You may be wondering how the first value a in X's range has a probability mass in that case, or how there could be a jump at a. The answer is that it's the difference between $P(X = a)$ and $P(X < a)$, where the latter of course equals zero.