Coding poems, Networking tensors, and a Caterpillar 🐛

Or, how to build a simple text Auto-Complete tool

😄

Amitava Banerjee, Abu Patoary, Gautam Nambiar, Subhayan Sahu

Girls Talk Math, University of Maryland

Summer, 2021

Contents

1 Welcome

"When day comes, we step out of the shade,
Aflame and unafraid
The new dawn blooms as we free it"
- The Hill We Climb

Welcome to the Girls Talk Math 2021! Thank you for joining us, and choosing this topic. We are looking forward to a great journey ahead of us.

As the title suggests, this package will help us get a glimpse of what goes behind the scenes of the text-suggestion or auto-complete algorithms that we often encounter when we type something. Two examples are given in Fig. 1.

That's right, we will get to learn some of the secrets behind those typing suggestions that pop up as we write text message, emails, and stuff (spoiler: they don't always come from just remembering what you previously typed). If you know how to code, then you can even use some of the lessons learned here to write your first text suggestion code. If you don't know coding or don't like it, no problem. Because the aim is to learn the fascinating math concepts behind, and some of their far-reaching applications in different fields of our lives. But that is the end goal. Throughout the journey, we shall get exposed to several important and exciting concepts of modern mathematics, computer science, and data science, including matrices and tensors, language modeling, and, some glimpses intp artificial intelligence and machine learning, and get to see how they all affect our lives. Let us get started with a brief summary of what is ahead.

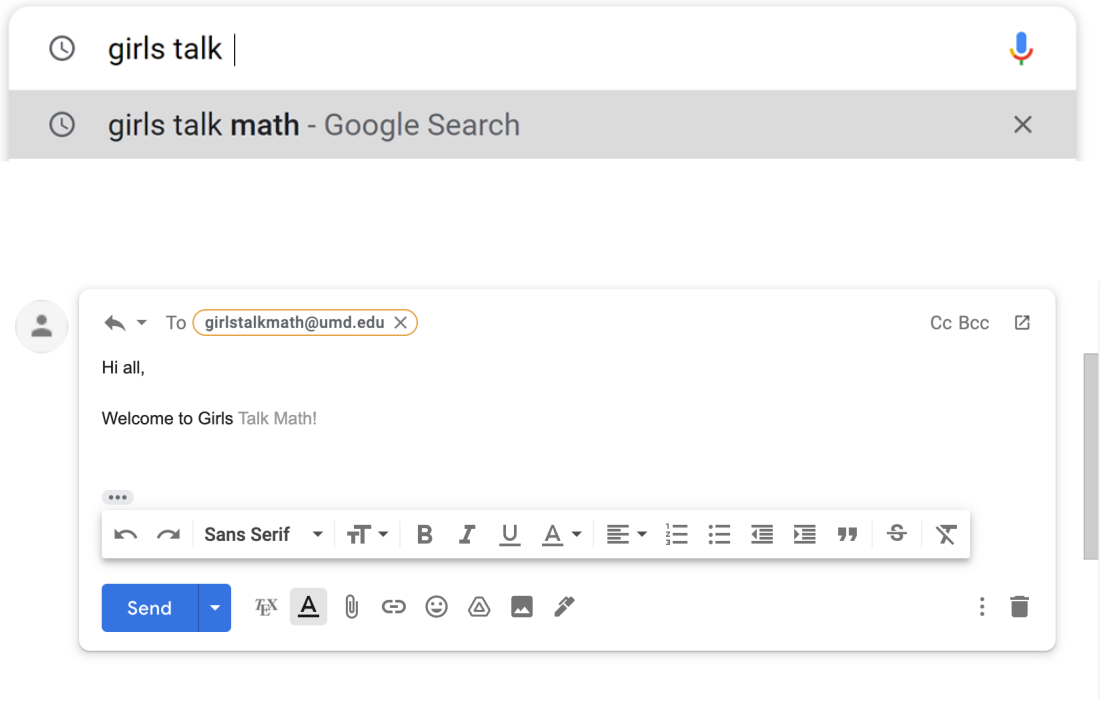


Figure 1: Google search automatically suggests some possible endings to what we are typing. In the draft email, the light gray part is automatically suggested by Gmail.

Since we wish to understand the math behind typing suggestions, we need to get a good grasp of the English language – *but mathematically*. What we mean by this is that, we wish to crunch texts from our favorite English novels, stories, and poems – to make a mathematical or computational model of the rules of English grammar. In particular, using works from the English literature as our data, we would like our model to be able to answer questions like,

How likely it is for a sentence to start with a Verb? Or end with an Adjective?

If a sentence starts with an Adjective, how likely is it to have a Noun following it immediately?

If our model is able to address questions like this, then we can predict what is the most likely word that can end an unfinished sentence that we have typed. So we shall start with some probability theory – the useful math behind statements such as "there is a 80% chance of rain today". But in order to formulate all these for the English language, we will need the mathematical framework of handling the numbers that will encode the occurrence of different English words. Why so?

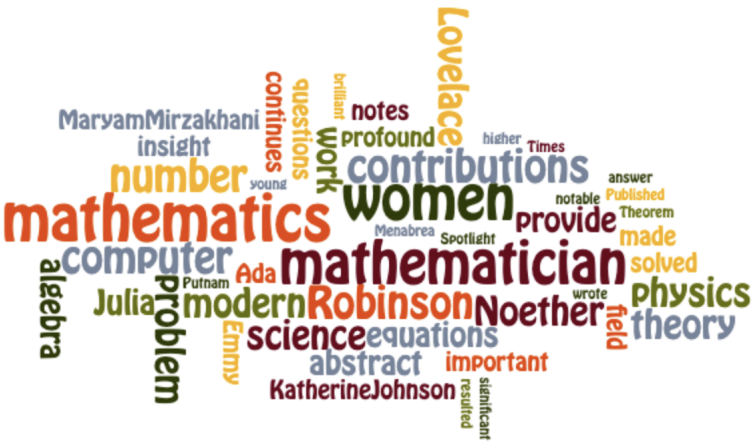


Figure 2: We took an essay on female mathematicians and constructed a word cloud, where the sizes of the words depend on the frequency of occurrences of the words. This is an example of how a computer might analyze text, by just counting how many times a particular word appears.

Well, as we are going to see in the very next section, when we start language modeling, we shall encounter mathematical structures having the capacity to store huge amounts of data – think of lots and lots of numbers stored in computer memories – and mathematical entities keeping track of them. At a point, the size of the computer memory required to store all of them will be big enough so that we shall be forced to find a way to compress the data, so that we can easily store, visualize, manipulate, and an-

alyze them – without losing their quality. Mathematical objects known as matrices, vectors, and tensors (we shall introduce them in due time) will be our friends while we attempt to do so. In particular, we shall express our language model in terms of something known as tensor networks – structures which are widely used in many important areas of physics and data science, fields that you might have heard of, like image processing, machine learning, and quantum physics. Tensor networks will finally take us to our goal of predicting the most likely piece that would finish a sentence. In figure. we give a glimpse of what a tensor network looks like. Hopefully by the end of this package you will understand what some of the structures in the Figure 3 mean.

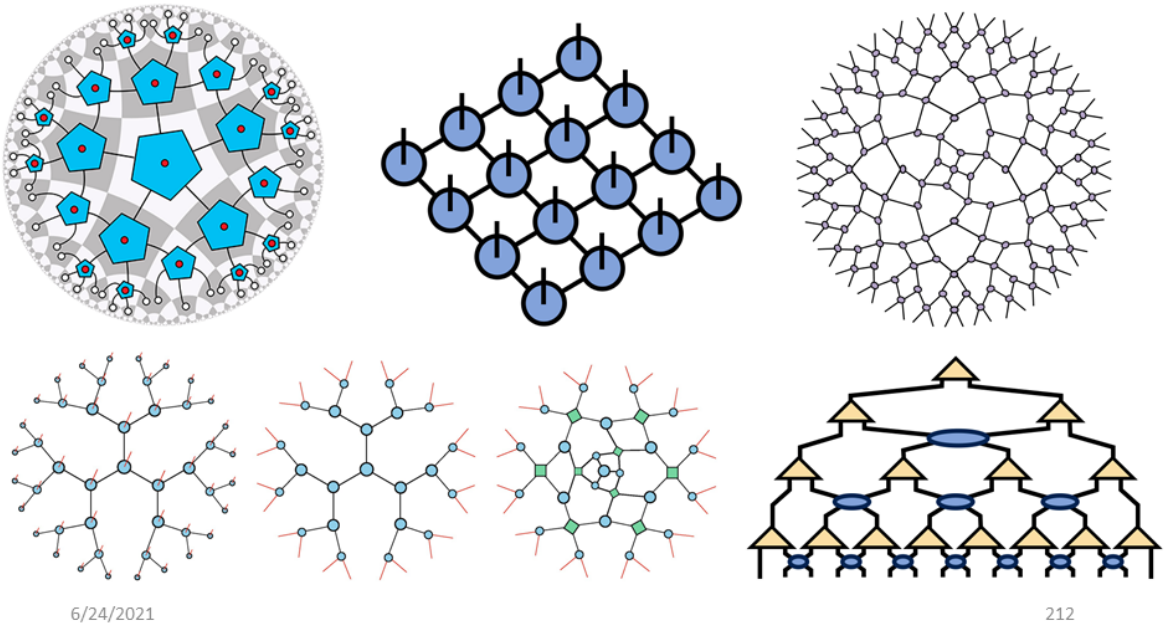


Figure 3: Some examples from the zoo of tensor networks. We will learn about making sense of these shapes with legs in this course.

With our goals fulfilled, we will briefly discuss some of the exciting things we can do with the all that we learned. These are the things that has come into place only very recently, say in the last five years or so, but they have changed our lives and the

society dramatically, and not always for the good. We shall talk about the over-arching effects of language modeling and similar other tech marvels in all aspects of our lives, and how we can dive into the game (and why we should).

Hope the journey gets exciting. We shall provide you the codes we have used in the background, and resources on coding, on request.

2 How to read a novel without reading it?

"Scripture tells us to envision"
- The Hill We Climb

In this chapter we are going to make a recipe which will predict how to finish a sentence. But in order to do that, we need to make a program which should encode some key features of English language grammar usage - things which we learn when we first learned English and use since then, almost subconsciously. How to do that? What do we start with? Well, let us start from actual data – from real English texts, stories, poems, and novels 📜. We first need to find out a way to crunch literature, *computationally*. That is, we need to have a computer code which will take a story (or a novel) as its input, and will be able to analyze it word-by-word, just like we do, when we read it. The difference will be, of course, it will be an automated reading 🖥️ – much faster than human beings!

Have we seen something like this before? Well yes, if you have ever typed in a Word document. The Word document counts the number of words, occasionally highlights spelling mistakes, and sometimes (particularly if you are using additional softwares), grammar mistakes. In order to that, the software must have been reading what you were typing all the way. Then, of course, there are fancier features like auto-correct and auto-complete in messaging apps which do this all the time – that we would like to better understand here. Similarly, websites like Amazon need to crunch through loads of customer reviews that they get everyday, and pick out what is good or bad about a product according to its customers

– and this data is generated faster than anyone human being can read or analyze them. These days, there are softwares to read entire scientific articles to generate one-line summaries. There are computer algorithms which read and score essays for standardized tests, or college applications (that is when it could get scary and problematic). Thanks to the huge demand in our everyday lives, automatic text analyzers are here to stay. Shortly, we will take their help as we start our task of coding English grammar.

What kind of information can we readily get with these text analyzers? Let's take an actual English text. We'll take the Inaugural Poet Amanda Gorman's poem "The Hill We Climb" to begin with, but everything we will be doing in this packet can be done for any other English text as well. Here's a link to the poem: The Hill We Climb.

2.1 A First Attempt at using Math for Texts

We first put the poem into our text analyzer 🤖. This friendly software 🤖 has its own details, which we need not delve into too much here. Very briefly, it first looks for spaces within the text and separates the text into "tokens" - words, punctuation marks, emojis, and so on. After that, it splits the whole text 📜 into meaningful sentence-chunks or lines (which it can do easily by searching for commas and full stops, for example). Then it stores each sentence-chunk word-by-word, and, using its own dictionary, adds a parts-of-speech (we'll use the abbreviation PoS when we get tired of writing it out) tag to each word. So, in the output, we get the list of sentences-chunks or lines in our text, and the parts of speech of different words in each of them. So now, our input poem is essentially a computer-readable 🖥️ list of lines, each stored as

a string/array (just another fancy name for a list) of parts-of-speech. The Figure 4 shows this process spelt out, and a couple of examples. For our analysis we will use **10 parts of speech (PoS) categories** - Noun, Pronoun , Verb, Adjective (including numbers), Adverb, Adposition, Determiner, Conjunction, Particle, and everything else clubbed into "Other". In the following table, we give some examples of these PoSs from the poem:

Parts of Speech (PoS)	**Examples from 'The Hill We Climb'**
Noun	The *Hill* we climb
Pronoun	The Hill *we* climb
Verb	The Hill we *climb*
Adjective	we will raise this *wounded* world
Adverb	it can never be *permanently* defeated
Adposition	we have our eyes *on* the future
Determiner	*The* Hill we climb
Conjunction	we know our inaction *and* inertia
Particle	move *to* what shall be
Other	0

With these data, we are now all set to have a first attempt at some preliminary analysis of English language. Let's start at the level of a single sentence.

Let's have a look at some of the numbers we have got. The text analyzer 🤖 reports are just in and they say that the poem "The Hill We Climb" has 110 lines in total. As expected, their lengths vary in size. Take a look at Figure 5 - our first figure of the session, showing the variation of line lengths.

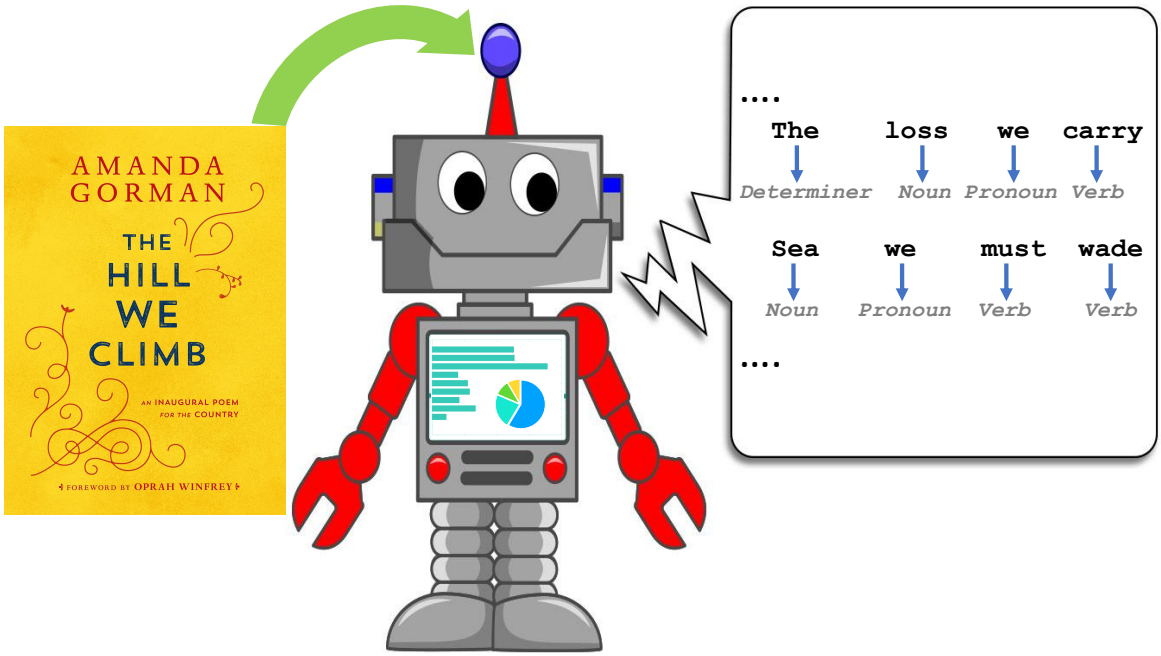


Figure 4: Our friendly text analyzer 🤖 reading the poem, splitting it up into lines, identifying parts of speech of different words, and showing some stats in the display screen attached on itself.

Plots like this are known as "histogram". They simply show you the counts of different things happening. For example, here the histogram shows how many sentences are there in the poem (that's the reading in the vertical axis) for each given length of line or poetic sentences (that is the reading on the horizontal axis). The Figure 5 tells you that there are 15 4-word lines (like "The loss we carry"), 11 9-word lines (like "where can we find light in this never-ending shade"), and so on. One thing we can readily see from this plot. 7-word sentences/lines (like "We close the divide because we know") are apparently the poet's favorite, as they occur the most (22 times). Now let's do an exercise.

Exercise 2.1

Looking at the histogram in Fig. 5, estimate the total number of words in the poem. What fraction of the poem

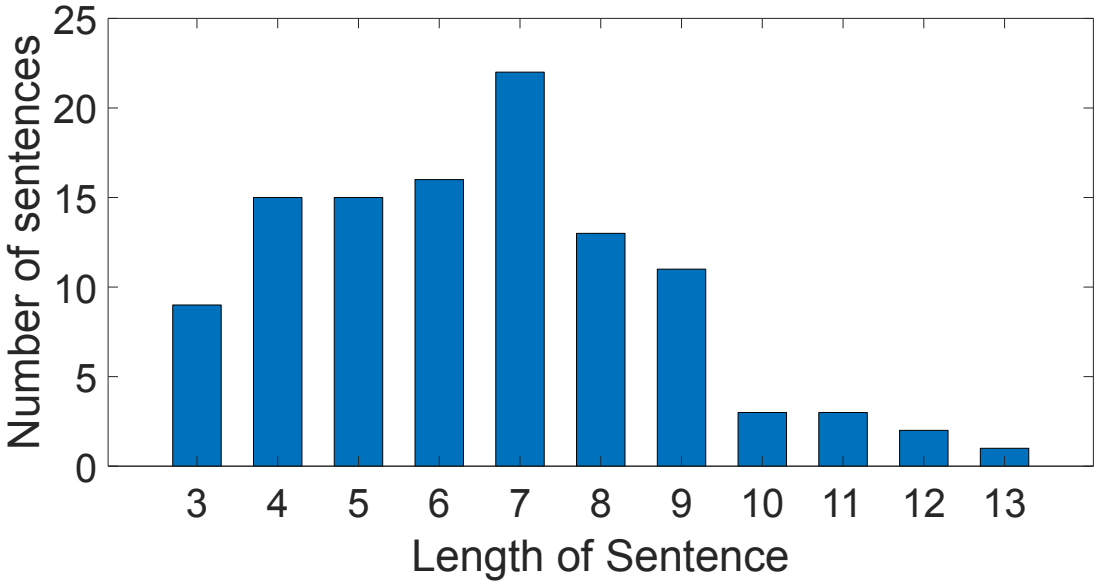


Figure 5: This plot shows the variation of sentence length.

(in terms of words) is made up of sentences having their lengths between 6 *and* 8 *words?*

The above analysis tells us how useful histograms are for getting a big picture view of the variations of different things in a given collection. Here, it gives us the information about the variation of sentence lengths within the poem. Also, with this piece of information, we are now starting to have an idea why such automated analysis of texts are useful – clearly if we had to count all the sentence lengths ourselves, it would be a much tedious job. But the computer does it in just a few seconds. Moreover, as we shall shortly see, such histograms will take us closer to our goal – of building the sentence auto-complete feature.

Let us now zoom into individual sentences. This is when we get to see the parts of speech details, and things start to get complex, and interesting. Thanks to our text analyzer 🤖, as shown in Figure 4, each sentence is a string of parts of speech categories.

Two such examples are shown in the Figure 4.

Time for another histogram! This time, we plot the histogram with horizontal bars. Let's collect all words from the poem and make a histogram of their parts of speech classes. The resulting histogram is shown in Figure 6.

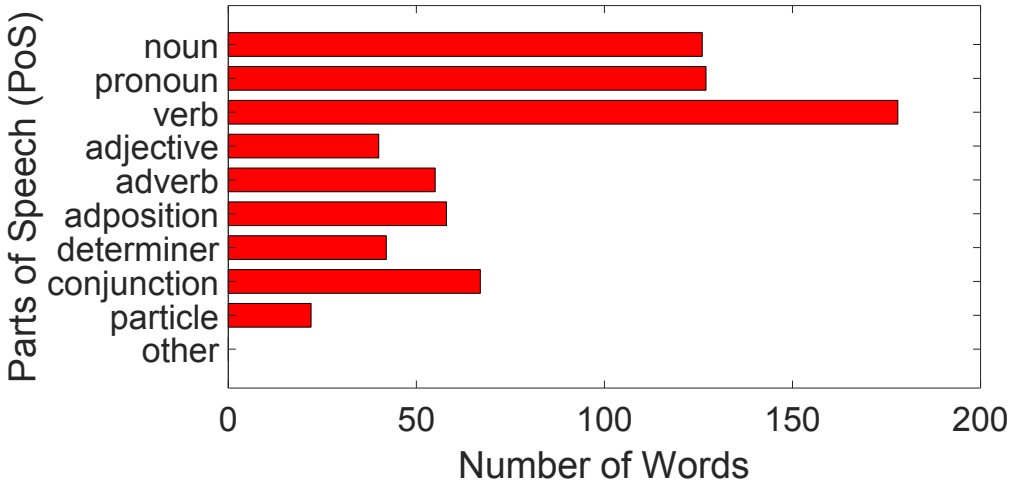


Figure 6: Histogram of PoS of words in Gorman's poem.

We see that the entire poem has approximately 125 nouns, 40 Adjectives, 180 Verbs, and so on. The histogram tells us that the most popular part of speech to use in an English sentence (à la poet Gorman) is a Verb. So, here is our first attempt to build an auto-completion algorithm. If no other information apart from this histogram is available to us, our best prediction for the next word in an incomplete sentence is a Verb. This is because our analysis revealed that a verb is the most frequent part of speech in a sentence. At least, this is what our text analyzer 🤖 has learned about the English language after analyzing 'The Hill we Climb' by Gorman.

Exercise 2.2
According to Fig. 6 what fraction of the words in the poem are Pronouns?

Ok, we get it, this is not sophisticated enough. But hey, at least we got something to start with! We can also get a hint about how we can make better guesses. First, let's look at our naive auto-complete lacks. When we finish a sentence with a Verb every time invariably, we are disregarding two things. In fact, incorporating them will be one of our central goals in the following sections.

One, the likelihood of a parts of speech in a sentence depends on its position. Common sense tells us that lines are more likely to end with nouns and Verbs ("the hill we climb", "the loss we carry") than, say, Conjunctions, which join two sentences or lines. So, maybe, there is some merit to incorporating this positional information (that is, where a part of speech occurs in a sentence) in the histograms we plotted earlier. Indeed, the previous histogram counted the total number of times a part of speech turns up – anywhere in a sentence. So that does not give us much information about questions like, what is the most likely place for a given part of speech in a sentence. Or, what are the most common parts of speech for starting or ending a sentence. To answer questions like these, we plot three histograms with more details, again from the data generously given out by our text analyzer 🤖. They are shown in Figure: 7.

Now the histograms show the number of times each part of speech occurs at three particular positions in a given sentence – the first, second, and the last words in a sentence. Of course, we could have done the same exercise for other positions near the middle of a sentence as well, but let's analyze these ones for the time being.

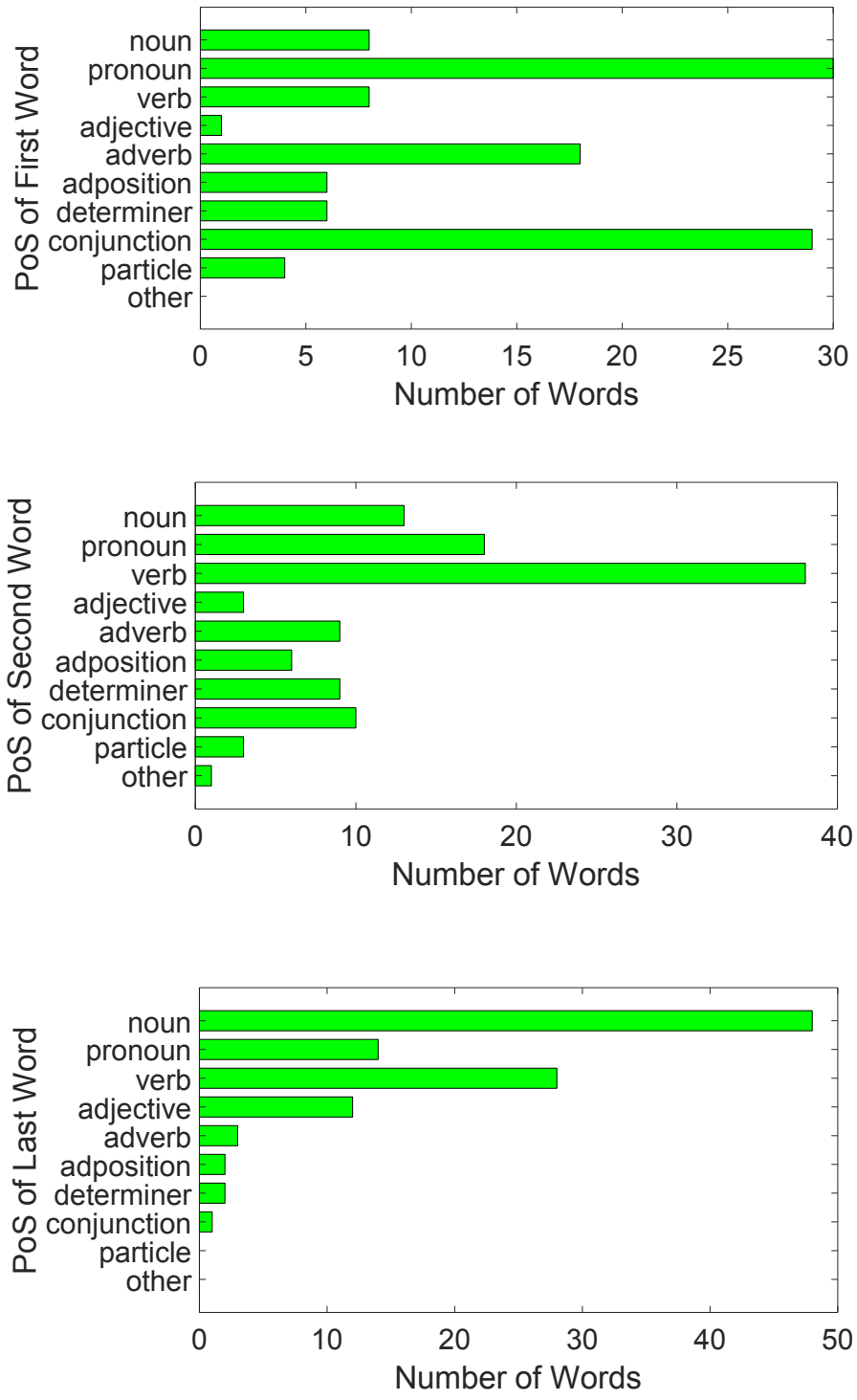


Figure 7: Histograms of The PoS of the the first, second and the last words of the lines of Gorman's poem.

We have already started to get a lot of data. Thankfully, we can still visualize the relevant information with the help of histograms.

These histograms show that lines (again, à la Poet Gorman) are most likely to start with a Pronoun, and end with a Noun ("We've braved the belly of the beast"). So, considering this new information at hand, our best bet to complete a sentence would be to use a Noun to do the job.

Exercise 2.3
Use the histogram in Fig. 7 to answer the following questions. What parts of speeches are far more likely to appear as the last word of a sentence than as the first word? What PoS-s are far more likely to appear at the beginning than the end of a sentence? Are there PoS-s which are equally likely (or unlikely) to appear at the beginning and the end of a sentence?

We still have one important piece missing so far, which makes our attempts at understanding language look primitive and not that promising. Can you guess what it is?

2.2 A Puzzle

Well, let us do a calculation to figure this out. How many sentences start with a Pronoun followed by a Verb (like "We are striving to forge a union with purpose")? Our know-it-all text analyzer 🤖 counts this number by crunching the poem-data, and finds it to be 23. Now let's try to do some fun math to come to this number in a different way. How many sentences begin with a Pronoun (followed by anything, no restriction)? Text analyzer 🤖: 30 sentences. Ok,

then how many of these actually have a Verb as the second word? This time, we will not bother our text analyzer 🤖 but we will try to find it ourselves based on the data at hand. How to find it? Well, from our second histogram (the one about second words in sentences) in Fig. 7, we know that the fraction of times a Verb turns up at the second position of a line is $\frac{19}{55}$ (38 out of 110 lines have Verbs at their second places). So we could guess that out of the 30 sentences which begin with a Pronoun, about a fraction $\frac{19}{55}$ should have a Verb coming up at the second position. This gives the number of sentences starting with a Pronoun followed by a Verb is about 10 ($\approx 30 \times \frac{19}{55}$). Note that the estimate $30 \times \frac{19}{55}$ is actually 10.36, but number of sentences must be whole numbers, so we guess the number to 10, the closest whole number. But this is very different than the actual number 23 that our meticulous text analyzer 🤖 counted!

What is going on over here? Why are these two estimates differing more than a factor a two? What are we missing? Can you guess? This puzzle will guide us into our next chapter, which is all about probabilities of many things at the same time.

3 Probabilities and lots of cartoons: vectors, matrices, and tensors

"We lay down our arms
So that we can reach our arms out to one another."
- The Hill We Climb

There is an obvious piece of common-sense knowledge that we have been ignoring so far in our analysis of language: words depend on other words before or after them. We usually cannot, for example, just put a Pronoun before a Conjunction. Nouns are most often followed by Verbs, and so on. So far, we have been missing this structure of sentences, and have been pretending that words sit at different positions in a sentence independently of what other words are around them. Thus, in the last example from Chapter 2, we were wrong in not acknowledging that the probability of a word being a verb, which our text-analyzer 🤖 had earlier estimated to be $\frac{19}{55}$, actually varies widely depending on what are the PoS-s of the words around it. Check out Figure 8 for a fun example of this.

So now, the lesson is - we cannot estimate the likelihood of the PoS of a particular word without considering the PoS of the words surrounding it. We need a mathematical framework to take that into account - and that framework is the theory of probabilities.

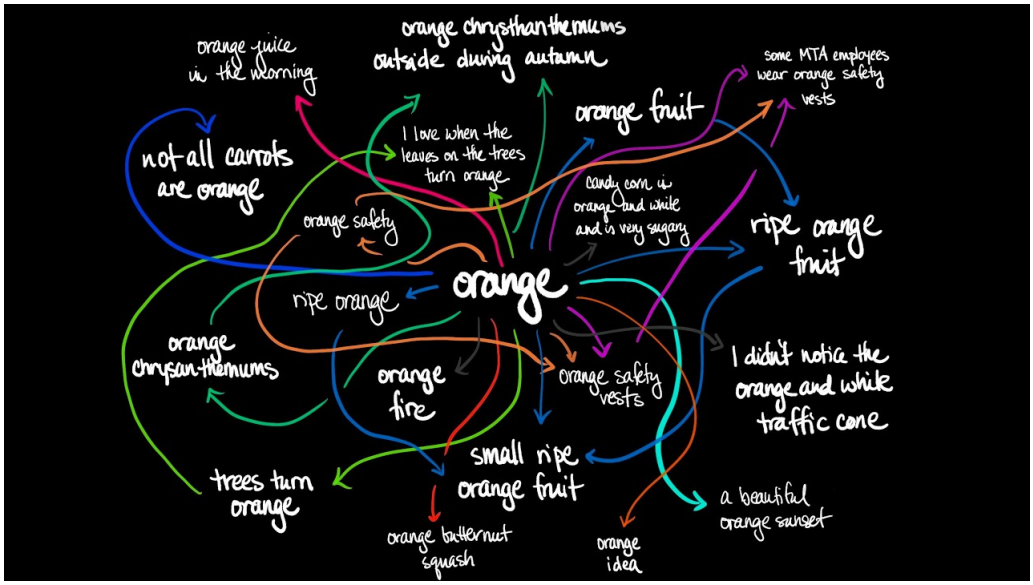


Figure 8: Words need other words to make sense! The meaning of the word orange (see the word network in the picture) depends on the context of the sentence of the phrase where it is used. Picture credits: Tae Danae Bradley (see her explanation in this video).

3.1 Probabilities and their Cartoon Representations

In this chapter we will learn about probabilities, and how to express them as cartoons! What is probability, really? For our purpose, the definition of probability matches our intuition closely. **The probability for something to happen is the ratio of the ways that this particular thing can happen, to the total number of ways all possible things can happen.**

It is best to understand this using an example. Suppose your friend brings you a bag with 10 balls. The balls are either red, blue or white. Suppose, there are 5 red balls, 3 blue balls, and 2 white balls. Now, your friend asks you to pick a ball from the bag without looking, so you don't know beforehand which color

you will get. What is the probability that you will get a red ball? Well, you could pick any of the 10 balls, so the total number of possibilities is 10. Out of these, only 5 balls are red, so you need to pick among these 5 to get a red ball. So probability of you picking a red ball is $\frac{5}{10} = \frac{1}{2}$! Let us play more with this example in the next exercise.

Exercise 3.1

We again have a bag with 10 balls - 5 red, 3 blue and 2 white.

a) If you were to pick 1 ball without looking, what is the probability that the ball you pick is blue?

b) What is the probability the ball you pick is yellow?

c) What is the probability the ball you pick is either blue, red, or white?

d) [Optional - a bit tricky] Now we change the rules of the game slightly. You first pick a ball without looking, and note down its color. Then you put the ball back in the bag and shake it up. Now, you pick a second ball without looking, and again note its color. What is the probability that both the balls you picked up were red? What is the probability that the first was red and the second was blue?

Take your time to work out this exercise carefully! From this exercise, we have actually learned many important concepts of probability. You must have seen that any probability is a fraction between 0 and 1. Probability that nothing happens or that something impossible happens, is 0. Probability that everything happens, on the other hand, is 1. Anything else has a probability

between these two numbers. We will write these properties more formally later in the chapter.

Now we will start applying these concepts in our analysis of text. Let us start by considering the Parts of speech (PoS) probability of a given text in more detail.

First, we look at the probability of the occurrence of a particular PoS (at any position in a sentence). Looking at the histogram of Figures 6 and 7, the probability of a PoS is the ratio of the number of words from that PoS (length of the corresponding bar in the histogram) to the total number of words (equal to the sum of lengths of all the bars in the histogram, which turns out to be 715). Let's unpack this a bit more. Due to the way we have defined the probability from the histogram, the probability of each PoS is a fraction less than 1, and they sum up to 1 (yet, another chance to go do the previous exercise!). The exact probabilities look something like this table -

Type	**Probability**
Noun	0.176
Pronoun	0.178
Verb	0.249
Adjective	0.056
Adverb	0.077
Adposition	0.081
Determiner	0.059
Conjunction	0.094
Particle	0.03
Other	0
Total	1

Exercise 3.2
Using the data from the PoS histogram in Figure 6 of the last section, work out one or two numbers in the preceding table and convince yourself that they make sense.

We can represent the above table by a sequence of numbers, denoted by the symbol P_i. P_i denotes the probability of parts-of-speech type 'i' where i (called an 'index') can be any PoS. For example, $P_{\text{Noun}} = 0.1762$, $P_{\text{Verb}} = 0.2490$, and so on. We can think of i as an open slot in P, where you insert one PoS at a time, and get back the probability of that PoS. **A sequence like P is called a vector.** These individual probabilities P_i are called 'elements'/'components' of the vector P. In the example considered in the section on text analysis, there were 10 types of PoS, so P is a sequence or vector of length 10 (that is, it has 10 components or elements). Written in full, it reads $P = [0.176, 0.178, 0.249, 0.056, 0.077, 0.081, 0.059, 0.094, 0.03, 0]$.

Just like the histogram in Figure 6, the other histograms that we created (in Figures 7) can all be represented by such probability vectors like P. So, we can all store a lot of data in a single letter P, if we can just remember how to read them out.

Exercise 3.3
If you feel you need to practice more on how to convert histograms to probability vectors, try doing the same for the other histograms in the last section.

How can the general properties of probabilities be expressed by the algebraic properties of P_i? Since the P_i-s are all probabilities, as we discussed above, they must be fractions adding up to exactly

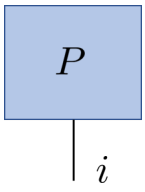
1. So they must satisfy,

$$0 \leq P_i \leq 1 \text{ for all i, and} \tag{1}$$

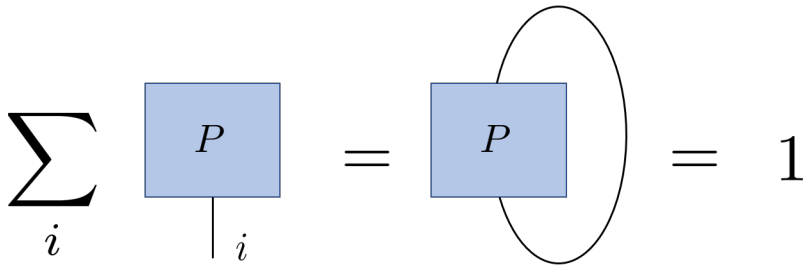
$$\sum_i P_i = P_{\text{Noun}} + P_{\text{Pronoun}} + P_{\text{Verb}} + ... + P_{\text{Other}} = 1. \tag{2}$$

where $\sum_i$ is a notation used to represent "summation over the index i". As you might have been feeling already, we are being pretty lazy lately with representing numbers (that is why we introduced vectors like P, to address 10 numbers at once), and this symbol $\sum_i$ is just another instance of that. We don't want to write long sums term by term anymore, since that takes lots of space and time to write, instead we use that symbol. Whenever you see this symbol $\sum_i$ before something, know that there will be an expression containing i (like P_i here) following the symbol, and the summation symbol $\sum_i$ wants you to evaluate that expression for all possible values of the index i that you can think of (like, here you are evaluating P_i for all the 10 different PoS), and add them up. The final result is just one number (the result of addition, no mystery here), like 1 here. Note that the index i is called "dummy", since we could have used any other symbol to express the same thing, just keeping in mind to repeat the same in the expression following it. Thus, $\sum_i P_i = \sum_j P_j = \sum_\alpha P_\alpha = 1$ etc.

Now let's represent vectors like P as a cartoon. Think of the vector P as a "probability vending machine" – there is one free slot to drop (not a coin but) a PoS, indicated by one index i. As such, we represent P_i as a single-legged (/ single-horned/single-armed, whichever you wish) object 🦄 as,



The free 'leg', or the free slot, which is marked by the 'free' symbol i, is the place where we interact with P from the outside, by putting in a PoS, and getting back the corresponding probability. The sum condition of the probabilities, $\sum_i P_i = 1$ can also be represented by a picture,



Here, since we sum over all the possible values of i, we don't have a free 'leg' anymore (the slots are full, no hand/leg pointing out for us to interact) - which we represent by wrapping the leg back into the square. Now there are no free legs and it is just a number. For our case, since this is the total probability, this number is 1. Oh yes, individual numbers, unlike vectors, do not have any free legs. Because, you know - they are just a fixed number each with no variation in output whatever we ask them. So, they never stick out any open leg.

The other lesson from this part is that, whenever we see there is a leg which is not sticking out and instead bent around, then we understand that there is a summation corresponding to that leg. More on this soon.

3.2 Back to Language Modeling again

3.2.1 A Matrix Model of Language

Before being carried away too much with fancy cartoons and indices, let us get back to our original problem involving probabilities of different parts of speech in lines. So now we know that PoS probabilities (anywhere in a sentence), pictured in the histogram of Figure 6, can be summarized into a vector which can be expressed as a cartoon with one leg sticking out.

Now the real fun begins! We can keep adding more legs to our cartoons and draw more pictures for studying probabilities of PoS of multiple words together, instead of a single word. This is the thing we are looking for, after trying the over-simplified estimation using the probability of a single word, which we did for the puzzle at the end of the last chapter. Suppose we want to study PoS probabilities of the first two words of a sentence. These could be phrases like "Black girl" or "We climb". In our representation, the first example is a two-word cluster of type "Adjective Noun", and the second is of type "Pronoun Verb". What is the probability of such two-word phrases? Extending our previous notation, we can represent them by this symbol $\mathbf{P}_{ij}$, which is the probability that the first word is of PoS type i and the second word is of PoS type j. Since this is a probability again, these numbers must satisfy some properties. Firstly, like before, they are all non-negative fractions, and add up to 1:

$$\sum_{ij} \mathbf{P}_{ij} = 1. \tag{3}$$

Here, there are two free indices, i and j, each of which belong to one of the ten types {Noun, Pronoun , Verb, Adjective,..., Other},

just like the index i last time. The difference is that, now we have two such indices in $\mathbf{P}$, so two slots for inserting PoS, or two legs. To get the "joint" probability of a pair of PoS anywhere in a sentence, put the first PoS into the first slot i (or the first leg, or the first free index), and the second one into the second slot j, and voila! $\mathbf{P}_{ij}$ spits out the probability you were looking for. So, the summation above runs over all possible PoS combinations ($10 \times 10 = 100$ of them) of the two indices i and j. Now you get to see why such summation notations were invented for lazy (or smart) people! Clearly, we could not write out 100 terms in the sum one by one.

We can also find probability of the first word of a sentence from this information. What is the probability vector of the first word of a sentence? We get this by summing over the second index,

$$P_i = \sum_j \mathbf{P}_{ij}. \tag{4}$$

This equation tells you that the probability that the first word is a Noun is the sum of probabilities of two-word phrases of types {Noun, Noun}, {Noun, Verb}, {Noun, Adjective}, and so on, where the first word is always a Noun.

Another way of seeing the symbol $\mathbf{P}$ is as something called a Matrix (with $\mathbf{P}_{ij}$s being the matrix elements, like the P_is were elements of the vector P last time). We are using the same letter for different objects, careful! We will talk more about matrices later in this section! Note that for our word PoS problem, we expect to store 100 numbers using the elements $\mathbf{P}_{ij}$, since there are $10 \times 10 = 100$ pairs of words like {Noun, Noun}, {Noun, Verb}, {Noun, Adjective}, and so on. This can also be represented by a structure like the one shown in Fig. 9. Like P was a 'linear' array of 10 numbers, the matrix $\mathbf{P}$ is a square array of 10×10 num-

bers, made up of the elements $\mathbf{P}_{ij}$ (but remember that matrices, in general, don't have to be squares, they can be rectangular as well).

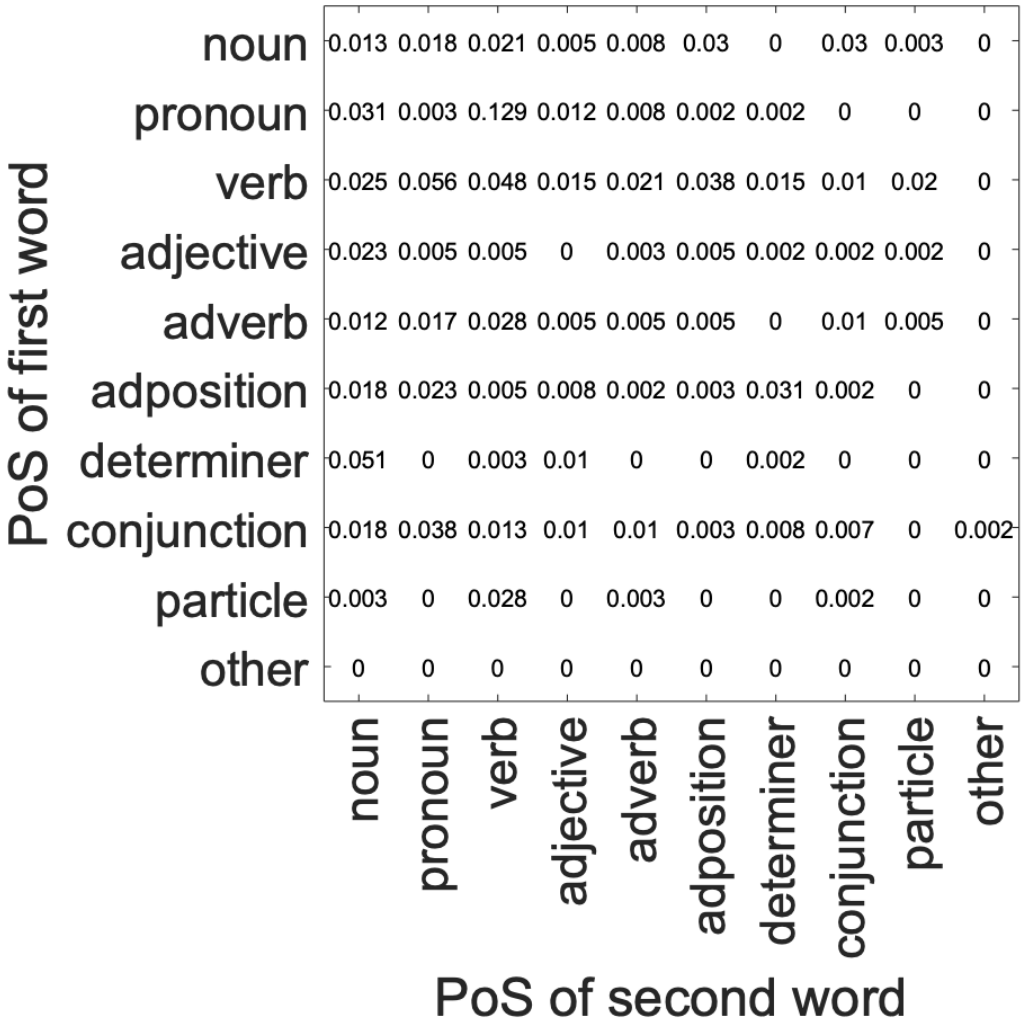


Figure 9: Probability matrix of two word phrases in Gorman's poem.

The Figure 9 was plotted with some new data freshly out of our text analyzer 🤖. It shows the probabilities of different PoS pairs of two consecutive words appearing at any position in the lines. The way to read the figure is as follows: choose a PoS along the vertical axis (say, Pronoun), choose a PoS along the horizontal axis (say, Verb), read the corresponding number from the square

array (0.129). You got the probability of a pair of consecutive words to be {Pronoun , Verb}, appearing at any position in the sentence, to be 0.129. So, if we put these two PoS in the two open slots of $\mathbf{P}$, we'd get $\mathbf{P}_{\text{Pronoun , Verb}} = 0.129$. Add up all the numbers in the matrix, you'll get 1. That is how it all works.

Before we dive deeper into matrices, let's us have a closer look at Figure 9 and see what does it tell about PoS of consecutive words in English language?

Exercise 3.4
To make sure we understand the information summarized in Figure 9 answer the following questions: which PoSs are the most likely to come (1) after a verb, (2) before a verb, (3) after a conjunction, and (4) before a noun, according to the data we have got from the poem? Convince yourself that your answers make sense grammatically (maybe try to think of examples of word pairs in each case).

What are some of the top entries of the matrix? These are the most favored PoS pairs from the poet. Figure 9 tells us that the top elements are {Pronoun , Verb} ("we have"), {Verb, Pronoun} ("knew it"), {Determiner, Noun} ("a country"), {Verb , Verb} ("have learned"), {Conjunction , Pronoun} ("so we"), { Verb, Adposition} ("reciting for"), and so on. All the top elements are grammatically sensible, and are common English PoS pairs which are used together. So, we have indeed encoded some more structure into our formalism now.

Let us also have a look at some elements in the matrix which are zero – meaning that they correspond to the particular PoS

pairs totally absent in the poem. Some of them, like {Determiner, Pronoun}, {Determiner, Adposition}, {Particle, Pronoun}, {Particle, Adposition} etc. are PoS pairs that are genuinely absent in English grammar (try to think of examples and convince yourself that they sound funny), and our matrix captures them correctly. There are also some pairs, like {Pronoun, Particle}, {Adjective, Adjective}, {Adverb, Determiner} etc. which are allowed grammatically (we know that from experience), but the poet decided against using them. Why is the study of tracking missing word pairs important? Well, firstly, we want to make sure that our matrix model of language not only codes which PoS pairs occur frequently, but it also identifies the forbidden combinations, since, those are parts of the grammar two. On top of that, like our last set of examples, items which are grammatically allowed but missing in the text are a result of the poet's preferences and unique style of writing. Finally, researchers like Margaret Mitchell working on artificial intelligence-based models of language have commented that just like what we say, what we don't say also contains our own implicit biases. As an example, they saw that while people are asked to label different images, people labeled green bananas as "green bananas", but yellow ones as simply "bananas", because they assumed the default color of bananas to be yellow. Similarly, many people who say "woman doctor", do not often say "man doctor", because of their bias! So, absences of grammatically possible word pairs encode lots of interesting information (often about the biases of the writer and/or the subject being written about).

Anyways, let's get back to some math for analyzing matrices, now that we are convinced how they can encode the quirks of language using matrices.

3.2.2 Matrices have two legs to walk

This structure, shown in the figure 9 above, has 10 rows and 10 columns, which is the generic structure of a matrix with dimension 10×10. We can again represent this represent probability (matrix) as a picture in Fig. 10. This is very similar to the last

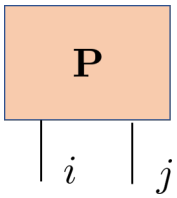


Figure 10: Probability of a 2-word phrase as a matrix - a box with two legs .

time, but now, it has got two open slots, or legs. The algebraic property written in Equation 3 can also be represented in the form of pictures, in Fig. 11.

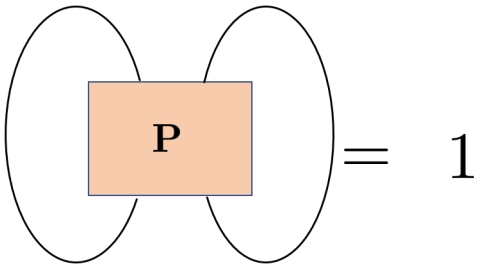


Figure 11: Properties of probabilities in pictures.

.

What is this picture saying? There are no 'free' legs in the box, all the legs are summed over (hence the curved lines going back into the box). In Equation 3 we have two indices i and j (since this is the probability of two-word phrases), so we have two curved legs to represent them in the picture.

3.2.3 A better guess towards our puzzle

With this new information at hand, can we do a better estimation of the number of sentences that start with a Pronoun followed by a Verb? The exact number is counted to be 23 by the text analyzer 🤖, and we estimated it as 11 last time! Well, let's try again. We have 30 sentences that start with a Pronoun. Now, Figure 9 tells us that the probability of a Pronoun to be followed by a Verb is 0.129, while the probability of a Pronoun to not be followed by a Verb is $0.031+0.003+0.012+0.008+0.002+0.002 = 0.058$ (adding all the other numbers in the Pronoun row of the matrix shown in Figure 9). Using these two numbers, we estimate that, about $\frac{0.129}{0.129+0.058}$ fraction of the sentences that starts with a Pronoun have a Verb in the second place. Thus, the number of sentences starting with Pronoun followed by a Verb is $30 \times \frac{0.129}{0.129+0.058} = 21$, which gives us a much better estimation than before! This is progress!

How can we do even better? Well, why stop at two-word phrases? We can consider longer collection of words, or sentences/-lines, and see the probabilities of different PoS strings. A sentence with L words can be represented by a probability with L 'legs' 🕷 - this is called a 'tensor'. Such a tensor can be represented by $\mathbf{P}_{i_1 i_2 \ldots i_L}$. Here, as before, each index runs over the number of types - in our case 10. Except for the number of legs (and the size), they can be handled in the same way as matrices.

A general L word sentence can thus be drawn as a L-legged object 🕷, satisfying all the properties of probabilities that we have discovered. This is shown in Fig. 12.

A general L-legged 🕷 box like the one shown above is called a tensor. We have already seen $L = 2$ case before - yes those

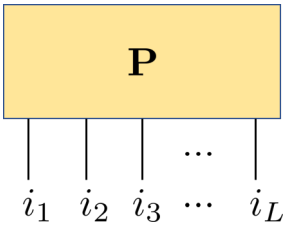


Figure 12: Probability of $L-$ word sentence as a tensor - a box with L legs 🕷.

are just matrices! In the next few sections we will have more to learn about tensors. Tensors are very useful mathematical structures which can be found in the study of many important topics in mathematics and physics, including the theory of gravitation. At the end, however, tensors are nothing but a collection of numbers that can be represented by a box with many legs 🕷. Now that you know what tensors are, you are well on the way to unraveling many more mysteries of the universe!

We shall encounter with tensors again, and get to see their full potential, near the end of this packet. But let's take a step back again, to have a closer look at the matrices – since using them already gave us a much better answer to our puzzle.

3.3 What is a Matrix, really?

Now we will take a detour and introduce the concept of matrix formally. You may not be aware of it but you utilize matrix everyday! Matrices are behind almost everything you see on a computer or mobile screen. The graphics and animations of video games extensively use matrices. Our laptop, mobile, and tablet screens are nothing but rectangular pixel arrays. So pixel brightness values, colors etc. can all be expressed with matrices. You may wonder

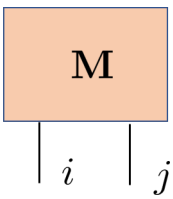
how can a table of numbers can create something as cool as animation. Well you see, animation is basically a series of images changing rapidly. And the way to store an image in a computer is to divide the image into many tiny squares, referred to as pixels. You assign a number(s) to each pixel which tells the computer what should be the color of that pixel. Combining the color corresponding to each pixels, the computer create the whole image. So for the computer, the image is a table of numbers or a matrix.

Previously, we discussed matrices for probabilities - which is a special form of matrix. In general, a matrix is a rectangular table of numbers arranged in rows and columns. For example,

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -1 & 5 \end{bmatrix} \tag{5}$$

$\mathbf{M}$ is a matrix with 2 rows and 3 columns. If a matrix has m rows and n columns, we say that its dimension or size is $m \times n$. So the dimension of matrix $\mathbf{M}$ is 2×3. We can identify each elements uniquely by specifying in which row and column it is located. For example, the element in the first row and first column is 1, in the first row and third column is 3, so on and so forth. Based on this we introduce the notation $\mathbf{M}_{i,j}$ to represent the element of matrix M with row index i and column index j. In the above example $\mathbf{M}_{2,1} = 2$ and $\mathbf{M}_{2,3} = 5$. Now we introduce the following diagram to represent matrix M:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -1 & 5 \end{bmatrix} \equiv \tag{6}$$



where the left hand of the box represents the row index and the right hand represents the column index of matrix M. This dia-

gram, a box with two hands, is the pictorial representation of a matrix.

Now we will learn about some operations relevant to matrices. For natural numbers like 1, 2, 3 we can do addition, subtraction, multiplication and division. We can do similar operations with matrices. For the purpose of our lesson, we need to learn addition, multiplication of a matrix by a number and multiplication of a matrix by another matrix.

3.3.1 Matrix Addition

We can add two matrices when they have the same dimensions. The summation of two matrices is computed by adding the corresponding elements as shown in Fig. 13.

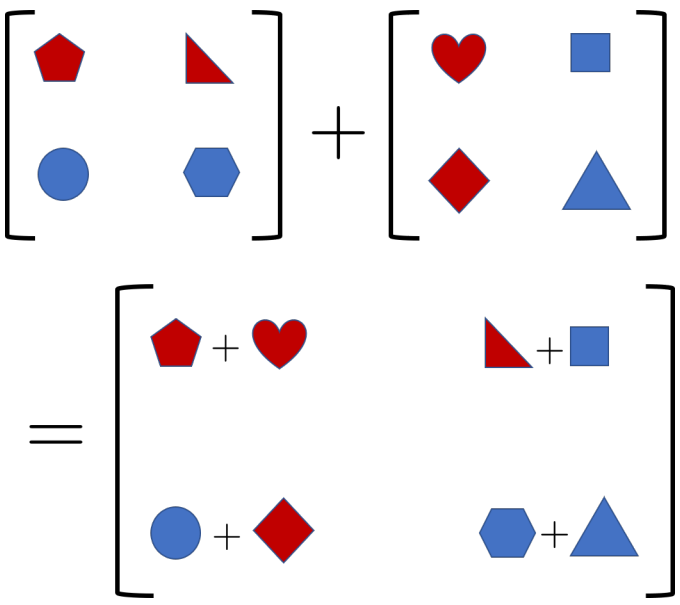


Figure 13: Sum of two 2×2 matrices. See how the symbols are being combined in the product.

In general, if **A** and **B** are two matrices of the same size then their addition gives another matrix (**A**+**B**) whose elements are

given by:

$$(\mathbf{A}+\mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j} \tag{7}$$

Exercise 3.5
What do you get after adding $\begin{bmatrix} 1 & 4 & 1 \\ 2 & -4 & 6 \end{bmatrix}$ *and* $\begin{bmatrix} 1 & -2 & -1 \\ -2 & 3 & -5 \end{bmatrix}$*?*

3.3.2 Multiplication by a number

A matrix can also be multiplied by a number N. The answer is simply a matrix of same size, where each number or entry of the matrix is multiplied by the number N, as shown pictorially in Fig. 14. If you think of the brightness of your digital screen as a matrix made up of the brightness values of the individual pixels, then increasing the screen brightness multiplies the same number to the brightness of all the pixels, making them all uniformly brighter!

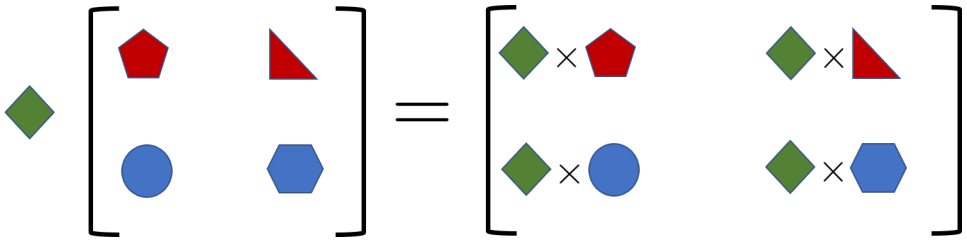


Figure 14: Sum of two 2×2 matrices. See how the symbols are being combined in the product.

Exercise 3.6
What do you get after multiplying $\begin{bmatrix} 1 & 4 & 1 \\ 2 & -4 & 6 \end{bmatrix}$ *with the number 5?*

3.3.3 Matrix multiplication

We can also multiply two matrices! You may guess that multiplication of two matrices would involve multiplying their elements in some way. But note that there can be many ways to multiply the elements. Which one is the 'correct' way? We will give you the answer which, at first, might seem counter-intuitive. For a graphical representation, see Fig. 15.

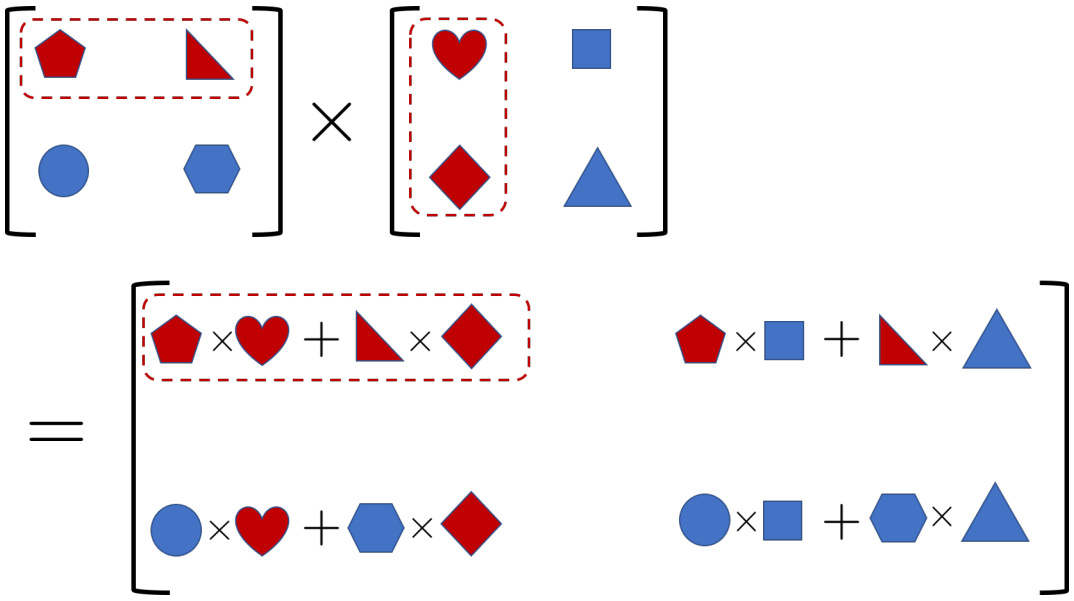
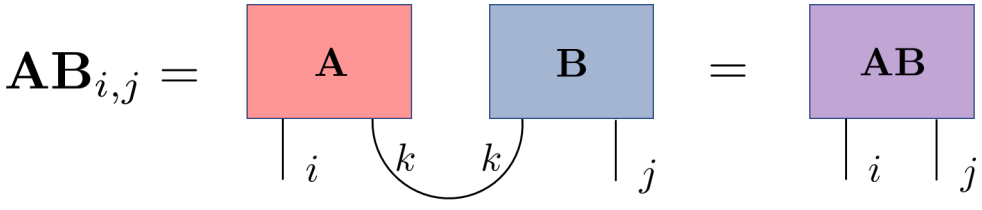


Figure 15: Multiplication of two 2×2 matrices. See how the symbols are being combined in the product.

As you can see in the figure, for matrix multiplication you multiply row times coloumns.We can also define matrix multiplication using the diagram of a matrix introduced earlier. Let's say $\mathbf{A}$ and $\mathbf{B}$ are two matrices with dimensions $m \times n$ and $p \times q$ respectively. The matrix multiplication is defined, diagrammatically, in the following way:



(8)

Now let's decode the diagram. In the intermediate figure we have summed over the column index of $\mathbf{A}$ and row index of $\mathbf{B}$, both of which are denoted by k. So the final result is again a diagram with two hands or a matrix. Hands/legs which do not stick out to us denote a summation, remember? Here that summation is over the index k. This will become clearer if we spell out the multiplication procedure in terms of the elements of the matrices. Multiplication of $\mathbf{A}$ and $\mathbf{B}$ gives a new matrix whose elements are given by:

$$\mathbf{AB}_{i,j} = \mathbf{A}_{i,1}\mathbf{B}_{1,j} + \mathbf{A}_{i,2}\mathbf{B}_{2,j} + \mathbf{A}_{i,3}\mathbf{B}_{3,j} + ... + \mathbf{A}_{i,n}\mathbf{B}_{n,j} \quad (9)$$

where n is the number of columns of matrix $\mathbf{A}$. We can concisely write the above summation using the summation notation as:

$$\mathbf{AB}_{i,j} = \sum_{k=1}^{n} \mathbf{A}_{i,k}\mathbf{B}_{k,j}. \quad (10)$$

Notice that, multiplication of any two arbitrary matrices are not allowed. Multiplication of $\mathbf{A}$ and $\mathbf{B}$ is allowed when the number of columns of matrix $\mathbf{A}$ is equal to the number of rows of matrix $\mathbf{B}$.

These formula might look imposing, but the underlying pattern is not too hard to grasp. For the pattern, take a look at Fig. 15, and practice! Matrix multiplication rules can look weird when we first come across it. For a more intuitive picture of why this is a good definition of matrix multiplication, check out the cool videos by 3Blue1Brown [1, 2] and this blog-post by Tae-Danae Bradley [3].

Exercise 3.7
Consider two matrices $\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -1 & 5 \end{bmatrix}$ *and* $\boldsymbol{B} = \begin{bmatrix} 2 & 3 & 3 \\ 1 & -1 & 0 \\ 2 & -1 & 1 \end{bmatrix}$.

*What is **AB**? What is **BA** (trick question!)?*

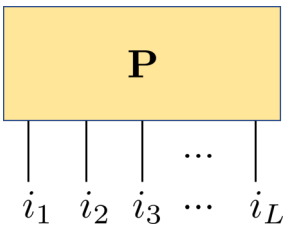
Exercise 3.8
*From the previous question we have seen that changing the order of matrix multiplication may not be allowed. Let's understand that with some more details. Suppose **A** and **B** are two matrices with dimensions $m \times n$ and $p \times q$ respectively. What should be the relation between the dimensions of **A** and **B**, if both **AB** and **BA** are allowed?*

3.4 A bit more on probabilities as Tensors

Let's get back to tensors again.

3.4.1 How big are tensors?

Let us remind ourselves of how we represented the probability distribution of a sentence. We found that a sentence or a line with L words can be drawn as a L-legged object  ,



Think of this big tensor P as an Ask-me-anything box. You ask it "What is the probability that word 1 is a Noun, word 2 is a Verb, word 3 is an Adverb, ... word L is a Noun,...?" and it will output an answer, say 0.089 (we just made this number up,

but the big tensor P has this info because our text analyzer 🤖 has diligently processed all the texts we gave it). But how many such questions are there that you can pose to P? Clearly we do not want our exercise to be anti-climactic by creating elaborate tensors which are only able to answer us one or two questions about the poem under study.

Why are we interested in this number? Because this number will give us an estimate of how much memory we need to store the big tensor P. Since P knows the answer to all these questions, it should have stored them in some way (supposing P simply stores the answer to each question separately). Let's estimate this number. As you might have guessed, it depends on the number of words L. For starters, let us take $L = 3$, that is, 3-word sentences. i_1, i_2 and i_3 each runs through 1 to 10. For example, take the sentence "Far from pristine" which goes as "Adjective Adposition Adjective". Here, i_1 =Adjective, i_2 =Adposition, i_3=Adjective. How many such questions are there that you can pose to P? Do this exercise to find the answer.

Exercise 3.9
We want to study the probability distribution P of 3 word sentences, where each word can be a parts of speech (PoS) of 10 types. How many entries are there in this tensor, or alternatively, how many combinations of 3 PoS can you make? (Each combination makes one question that can be posed to P)

What about 4-word sentences? Answer: $10^4 = 10000$ (if you are not familiar, note the notation too, here 10 has "power" 4, which gives a number having 4 zeros after 10. We shall use this notation later too. This is used to write large numbers in a com-

pact form). This growth is said to be "exponential" in the number of words (a word you would have heard in the context of the pandemic).

3.4.2 Visualizing tensors

How do we visualize these tensors? Now we will see that we can actually see them as matrices! Let's start by considering a 4-word phrase - the tensor has 4 legs 🐘. However, we can transform this into a 2-leg object 🦩 by combining legs. An example of how we can do this is shown below, in Fig. 16.

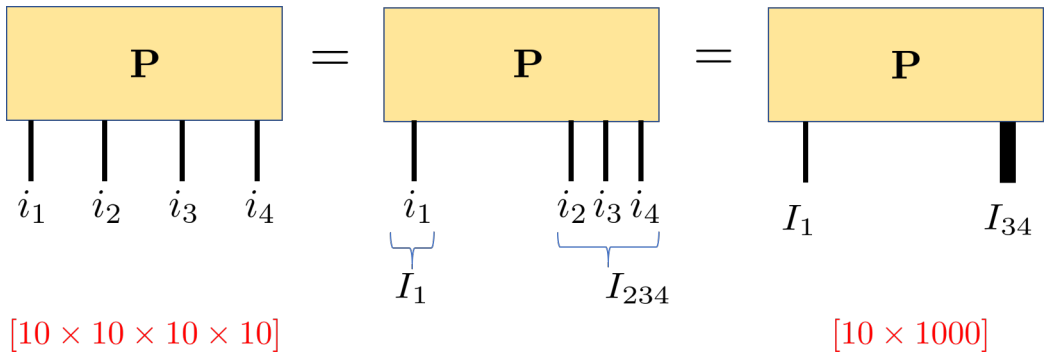


Figure 16: Reshaping the 4-legged Tensor 🐘 into a 2-legged Matrix 🦩

What are we doing here? Recall that P, the 4-word probability distribution is the probability of sequences like {Word1, Word2, Word3, Word4}. But now we can pair some of these elements like this, {Word1,{Word2,Word3,Word4}}. To this 'reshaped' P we can ask a question like, what is the probability that Word1=Noun, and {Word2,Word3,Word4}= {Verb,Conjunction,Verb}, that is, the last three words are Verb,Conjunction,Verb triplet. Check that this is the same question as asking for the probability that {Word1,Word2,Word3,Word4} = {Noun,Verb,Conjunction,Verb}!

How many such collections, {Word2,Word3,Word4} are possible? $10 \times 10 \times 10 = 10^3$. So, this reshaped Matrix (with only two legs 🦩) has 1000 columns! This matrix is of size 10×1000. Check, that the number of entries of this matrix is again $10 \times 1000 = 1000 = 10^4$, which is the same as the number of entries of the 4-legged tensor 🐘. However, this is not the unique way in which the 4-legged tensor can be represented as a 2-legged matrix 🦩. In the next exercise we will discover another way of representing this as a matrix.

Exercise 3.10

We can view the 4-legged tensor 🐘 as a different kind of matrix as well! Follow the strategy shown in Fig. 17. What is the size of this reshaped matrix?

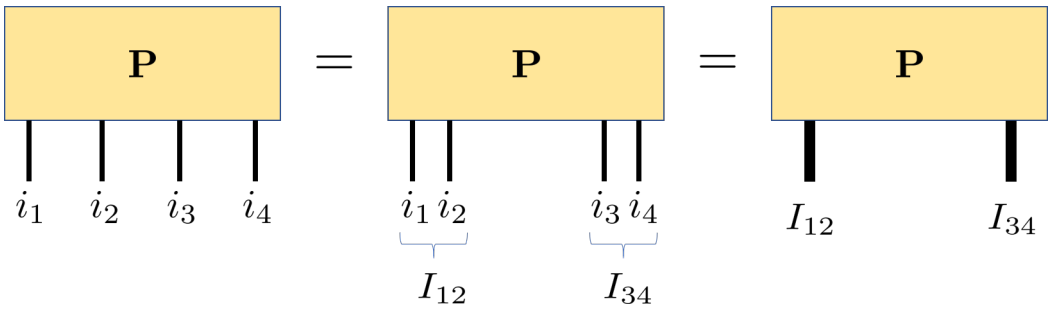


Figure 17: Reshaping the 4-legged Tensor 🐘 into a 2-legged Matrix 🦩 (Exercise)

In this chapter we have learned how to visualize probabilities as tensors, and we have also drawn pictures to represent tensors. In Fig. 18, we summarize the visualization of tensors with different numbers of legs. For more cool drawings and to read more about tensor network representations as diagrams, check out this blog post by Tae-Danae Bradley [4].

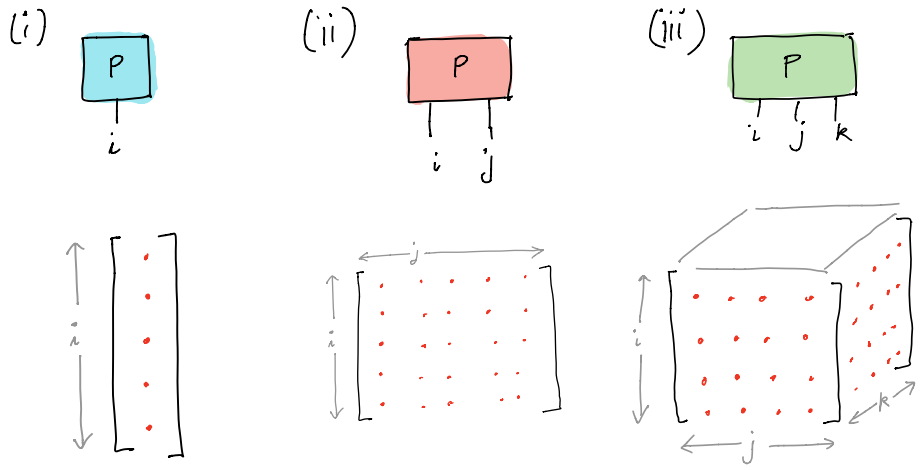


Figure 18: Summary of tensor visualization

3.5 A Storage Problem

In the last section, we saw the number of entries of a tensor representing the probabilities of a 4-word phrase, where each word can be one of the 10 types of parts of speech. Now let us take this further and ask, how much memory do we need to store a big tensor P for $L = 15$ (15-word sentences)? Suppose one needs 64 bits to store each probability. There are 10^{15} such probabilities. So we need 64×10^{15} bits in all. This is 8000 terrabytes! To put things in perspective, a modern computer usually has around 1 terrabyte of memory. So, to store the tensor we need 8000 of computers. We wanted these tensors to store many numbers, but now their size also increases catastrophically! How can we handle them?

4 Slicing it for storing

"So while once we asked: How could we
possibly prevail over catastrophe?
Now we assert: How could catastrophe
possibly prevail over us?"

- The Hill We Climb

At the end of the last chapter we saw that storing the probability of PoS of a 15-word sentence will need 8000 normal computers!! Surely, there should be a more practical way to store $\mathbf{P}$ than to use 8000 TB! But how? One solution is that we don't separately store the answer to every single question you can ask to $\mathbf{P}$. Instead, we get a recipe so that given a question, and some data stored beforehand (of a more manageable size, of course, not 8000 TB), it *computes* the answer, instead of pulling it out from memory. To see that it is in possible as a matter of principle, we will first look at an oversimplified example. Suppose, life was easier, and the PoS of all words are independent of each other (which, as we saw earlier, is not the actual case!). Continuing what we learned in Section 2, this means (for example) that the probability to get {Noun, Verb, Adverb} is $P_{\text{Noun}} \times P_{\text{Verb}} \times P_{\text{Adverb}}$.

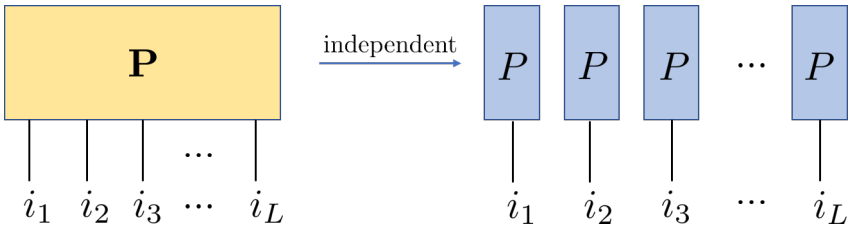


Figure 19: When all words are independent of each other, the tensor $\mathbf{P}$ factorizes into *vectors* P

Exercise 4.1
Convince yourself that the probabilities should actually multiply if the words sit in sentences independent of one another. In this simplified situation, how many numbers should the computer store?

So 10^{15} entries have got compressed to just 10 entries! Before we celebrate, we should remind ourselves that this is possible only when the words sit in their positions independently. What if they are not (and they are not, as we saw)? This is where Tensor Network methods come in. But we reserve this discussion for later.

For now, let's start with matrices like before, and see how we can compress them, without losing any essential information.

4.1 Compressing Data - let's start with numbers

We will now introduce an idea of how to compress a matrix. But first, what do we mean by compression? Well, people who have some familiarity with handling files and folder in a computer will often find a "compress" option. What this does is that it stores the data in a smaller memory (in terms of fewer numbers) than what would be needed for the uncompressed file. Do we lose any data during this? To be honest, yes. Otherwise you won't be able to fit in files to a smaller memory. But, if the compression is done cleverly, that loss can be made small in terms of the usage of the file. You shouldn't be able to see any practical difference between the compressed and uncompressed files, unless you are looking very hard. So, compression allows you to store and analyze your

data in a much efficient way, without losing much of the important features of the data.

How do we compress matrices? Well, let's take one more step back and see how we can compress individual numbers first. Consider a number 15799069 – a many-digit, pretty big number. But, it can be factorized: $15799069 = 11 \times 13 \times 17 \times 67 \times 97$. So we see that if you can break a big number up into small pieces, you can just store the small pieces (here, the factors 11, 13, 17, 67, 97) which would consume much less memory. When you want to use the full number, just remember to bring back and multiply all the small pieces. This is also the same experience we had, for the example at the very beginning of this section, when our long probability tensor factorized into small bits – each with only a leg. It is just like cutting a cake 🎂 into smaller pieces 🍰 for storing in the fridge.

But the above compressions are exact (both for the number and the tensor for independent PoSs), we did not lose any information since the factorization was exact in both cases, and multiplication of the factors accurately gave us the original big number (or the long tensor) back. But we may not be that lucky all the time. Consider a nearby number, 15799073, which, as it turns out, is a prime number, so cannot be factored. This is bad news, since we won't be able to factor it like 15799069, and thus we won't have any exact compression.

But wait, all is not lost. in fact, we can still write $15799073 = 15799069 + 4 = 11 \times 13 \times 17 \times 67 \times 97 + 2 \times 2$, and, if the places where you will be using these big numbers allow for some small error, then you can just get rid of the extra 4, and approximate 15799073 by 15799069, which would allow you to use the goodness

of factorization again. So, if we are willing to have a little bit of error (think of some crumbs and bits stuck to the knife while you slice the cake), we can save a whole lot of memory space, and write big things 🎂 in terms of smaller ones 🍰 in a way which allows easy analysis and visualization. Moreover, the compression is not unique. For example, we can also write $15799073 = 15799073+4 = 2 \times 5 \times 72 \times 19 \times 1697 + 3$ and get rid of the 3 at the end, again to get a factorized expression.

4.2 Compressing Matrices

Can we do the same for big matrices? Yes, we can. Matrices are nothing but numbers when you look inside them. Multiplying two or more matrices gives you another matrix – just like numbers multiply to numbers. So, matrices can be factorized too, but just like the numbers above, often approximately. This technique is known as "singular value decomposition" (quite a mouthful) or shortly SVD. This gives you a recipe to fragment a matrix into three smaller matrices, as shown in the Figure 20. We shall not go into the details of how this factorization is calculated, but only discuss and see how useful this neat trick is.

The middle one of the three factor matrices, **D** is a "diagonal" one, meaning that it has non-zero elements only along its diagonal, i.e., $D_{ij} = 0$ whenever $i \neq j$. (For example, in Figure 20, only elements on the diagonal, colored green, are different from 0). The elements of D are sorted in decreasing order (going from top left to bottom right). Note, we have not learnt you how a matrix can be factorized using the singular value decomposition, but just learnt that it can be done. However, once we have the factor matrices (like in the Right Hand Side of Figure. 20), we can multiply them

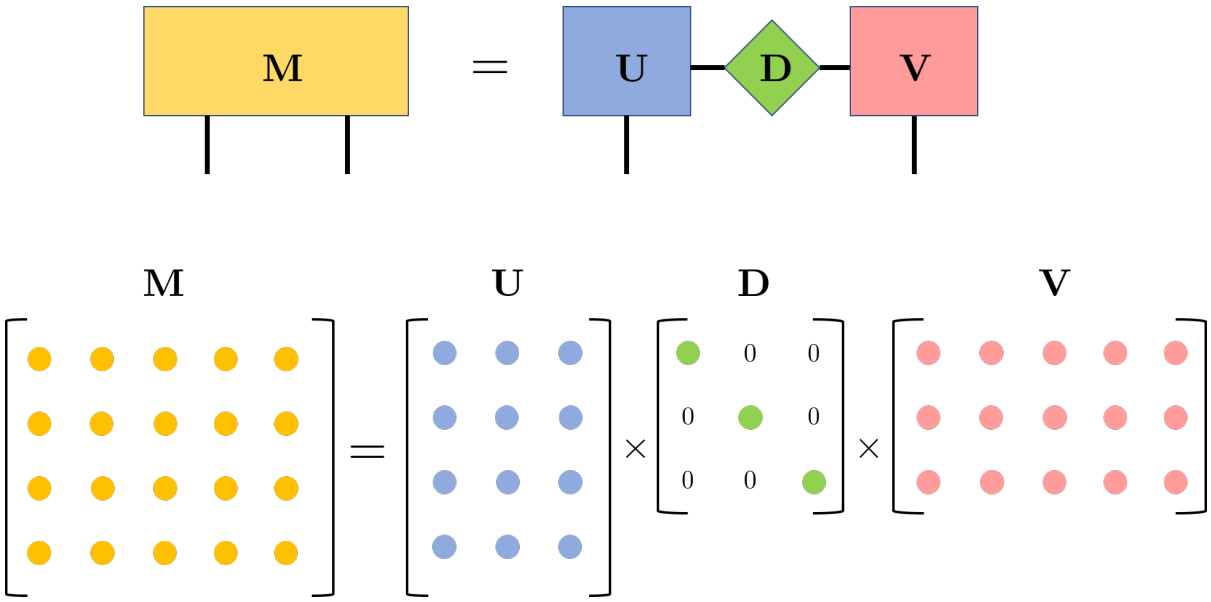


Figure 20: This is the diagrammatic representation of singular value decomposition (SVD). The matrix **M** is factorized into three smaller matrices **V**, **D** and **U**. In the form of an equation $\mathbf{M} = \mathbf{VDU}$.

back together to get the original matrix. Think of the time when you first learnt about arithmetic operations. Typically, we were taught multiplication of numbers before division or factorization. At that stage, you wouldn't have known how to factorize a number (i.e. write 30 as $2 \times 3 \times 5$) but when given the factors $2 \times 3 \times 5$, you can easily multiply them together to get 30. We are at that stage of our understanding of matrix arithmetic. In this course, we will not learn exactly how to factorize a matrix, but learn that there is method of factorization called singular value decomposition. Once the factors are known, we have already learnt how to multiply matrices and we can reconstruct the original matrix.

At first sight, it doesn't seem like we have made any progress in our effort to compress a matrix – in fact we have got three matrices instead of one! To convince ourselves why this is a step in the right direction, we will go through the following exercises,

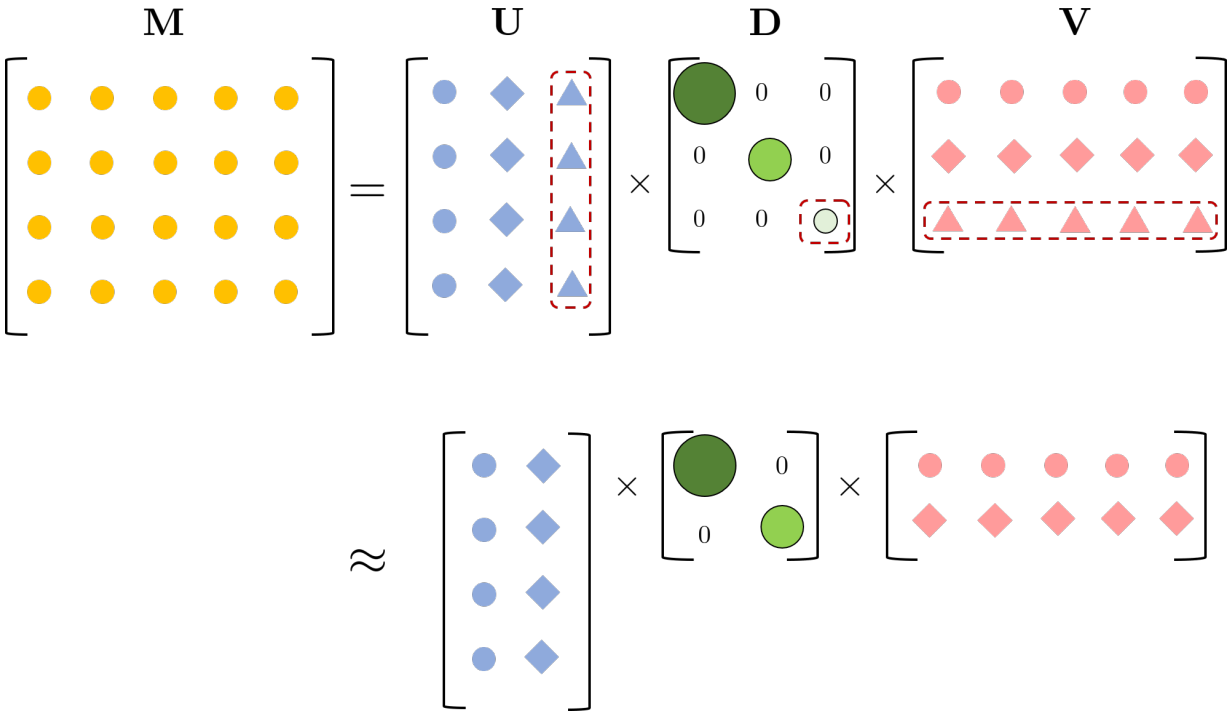


Figure 21: Compressing a matrix by SVD

which are built around Fig. 22, where we consider the same process as in Fig. 20, but now with a level of compression.

Exercise 4.2

1. *In Fig. 21 how many numbers does* $\boldsymbol{M}$ *on the left hand side store? How many numbers in total are stored on the right-hand-side?*

2. *Convince yourself that the first line of Fig. 21 obeys the rules of matrix multiplication that you learned in Section 3.3.3. In particular, do the sizes of the rows and columns make sense?*

3. *Suppose the bottom-right-most number (with dashed box) in* $\boldsymbol{D}$ *is negligible compared to other numbers on the diagonal. (Here is our chance to use our wisdom from slicing cakes and discarding crumbs!) We discard this number. But here is the secret – by*

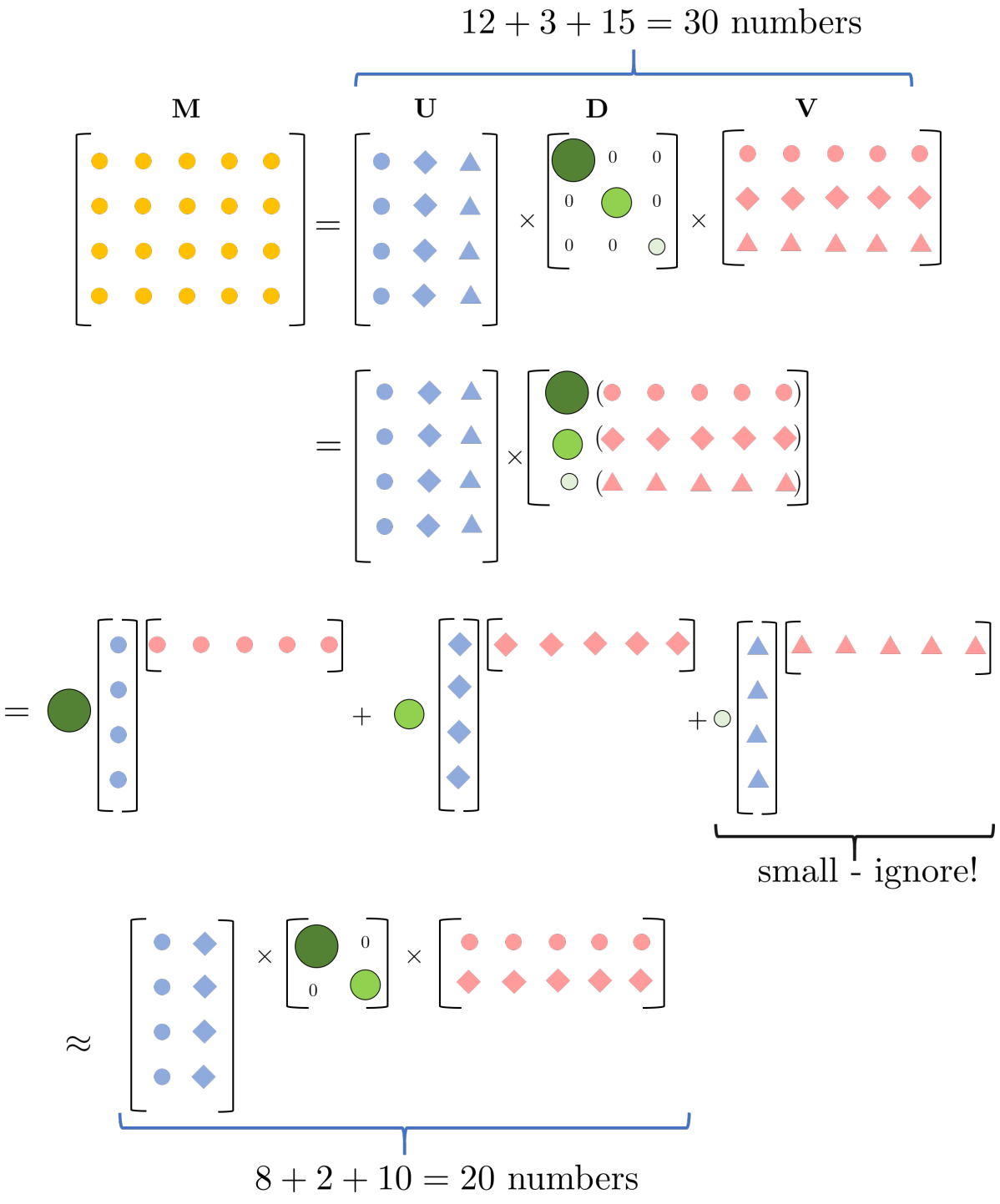


Figure 22: Compressing a matrix by SVD - in detail

doing so, you have effectively discarded not one, but many more numbers. Calculate how many. (This is a hard question. Don't worry if you can't solve it in the first try. The dashed boxes in $\boldsymbol{U}$ *and* $\boldsymbol{V}$ *will give you a hint. For more help, we have worked this out in detail in Figure 22. Don't hesitate to ask your mentor for help)*

4. *Now suppose we could ignore the smallest two numbers in* $\boldsymbol{D}$ *instead of just one. How many numbers in total does this effectively get rid of? Calculate the compression achieved, i.e the difference between the number of entries on the right-hand-side and left-hand-side. [Hard]*

5. *This may not sound like much. But now, suppose the matrix* $\boldsymbol{M}$ *was of size* 100×100*. Now the* $\boldsymbol{U}$*,* $\boldsymbol{D}$ *and* $\boldsymbol{V}$ *are each* 100×100 *matrices, with* $\boldsymbol{D}$ *having only* 100 *non-zero numbers along the diagonal. The real power of the SVD compression technique is revealed when all except a handful of the elements of* $\boldsymbol{D}$ *can be ignored. For example, suppose we keep the largest 2, and discard the remaining 98, what is the compression we can achieve? [Hard]*

To see that this works in practice, we have two examples worked out for you. The first example is a fun one, see Fig. 23. The leftmost picture on Fig. 23 is a self-potrait of the Mexican painter Frida Kahlo. As we discussed while introducing the concept of matrices, any image is just a matrix. In particular a grayscale image as shown here is just a matrix with a grayscale value for each pixel (basically, this value determines how dark the gray pixel at

any position is - it ranges from white to black). In the version of the digital picture that we downloaded from the internet, the image size is 360×277 pixels. We can think of it as a matrix of the same size. How many elements does this matrix have? It is $360 \times 277 = 99720$. You need to specify the grayscale values of that many pixels to get the full image. Too many numbers to specify, huh? But, we can now compress the original picture-matrix, by the trick of singular value decomposition that we just learnt about. In each of the subsequent pictures in Fig. 23, we keep only a few of the numbers in the middle diagonal matrix of the singular value decomposition. The leftmost one is the original one. All the other ones have the same size as the original one ($360 \times 277 = 99720$), but the grayscale values of the 99720 pixels in each case can be stored using fewer than 99720 numbers (if you let go of some of the details, the "crumbs"). For example, in the second one, we need to store only 3190 numbers, in the third one 6380, and in the fourth one, 12760 numbers only, to approximately get an idea of how the full 360×277-pixel images would look like. That is an example of an image compression in action! Depending on how much details you want to retain, you can choose the number of terms to keep, in your singular value decomposition. We get to see how the details emerge when we take more and more terms in the summation, but also appreciate that we can get some broad features right away when we only take a few terms in the singular value decomposition. For example, if you pay for only 3190 numbers, you can get an idea of the painter's face, her hair, and maybe a rough glimpse of the bird near the bottom. For 6380 numbers, you get more details of her eyes, eyebrows, and the branches around her. You need to use 12760 number to get the animals – the cat and the monkey, and the leaves in the background. But the control is now in your hand, you can choose how many numbers you wish to keep.

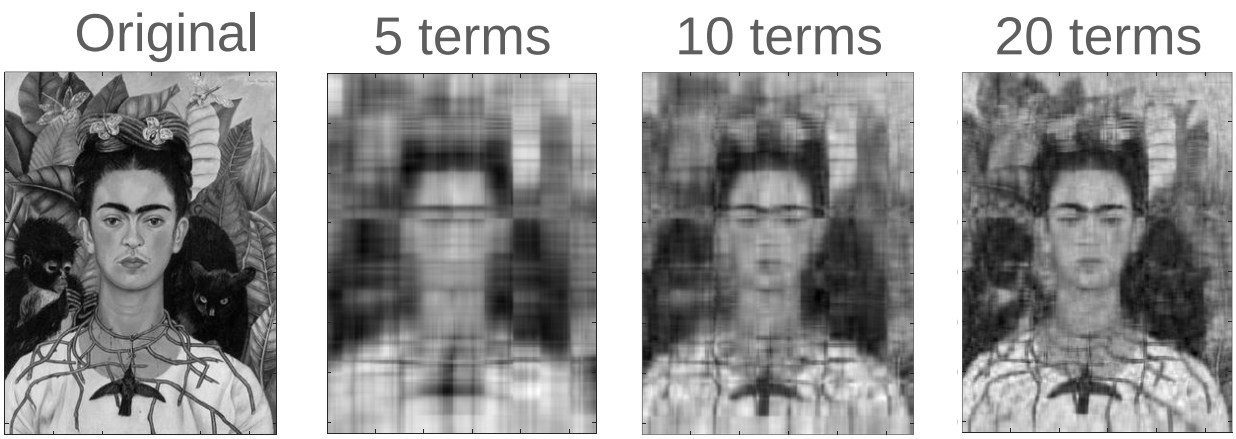


Figure 23: The painting by Frida Kahlo can be thought of a matrix of pixels, each assigned with their grayscale values. We compress that matrix, write is as summation of decreasing matrices via singular value decomposition, and show the results when we truncate the summation at different number of terms.

So how does it all relate to our language modeling? In the next example, we consider our familiar two-word probability matrix in Fig. 9, and approximate that using singular value decomposition. That matrix is only 10×10. So the wisdom from the last exercise says that the compression won't be that dramatic, but just let's give it a try anyways. In Figure 24 we show how the matrix can be seen as a sum of different matrices, like in Fig. 22. Here the matrix is plotted in color - red means bigger probability and blue means smaller probability. As in the Figure 22, matrices in the later parts of the sum are made up of smaller numbers (you need to zoom in to see the numbers, if you wish to check it!). In Fig. 24 we also show the retrieved matrix in the cases where you keep one, two, and three terms of the summation. As for the quality of retrieval, we see that just the colors of the sum of the first two terms (we need 42 numbers to get that, which means we still have some compression from the original 100 numbers) look a lot

like the original probability matrix. In particular, the top entries of the original matrix, like {Pronoun, Verb}, {Verb, Pronoun}, {Verb, Verb}, {Verb, Adposition} etc are already prominent in the reconstructed matrix.

4.3 Factorizing matrices using diagrams

How can we describe the process of factoring matrices this way compactly using our diagrams? The diagram, as we see from the Figure 25 looks really simple. The process of singular value decomposition just adds a "hidden joint" 🦴 between the legs of the original matrix.

One important concept here is the notion of an internal, or "bond" index, represented by the joint 🦴 in between the legs. Why is it internal? Well, firstly, the joint 🦴, unlike the legs, have not got a loose end, where we can plug inputs. This makes it different from the two other legs, each of which has got a free end where we can plug a parts of speech, in order to get the number of instances where a particular parts of speech pair occurs. Rather, the joint 🦴 acts like an internal gear, controlling the output you get when you plug the parts of speech in the open ends of the legs. What do we know about such internal joints which do not stick out? Well, we know that they refer to a summation. The joint 🦴, or "bond" accounts for a hidden index, and the summation corresponding to the bond runs over that index. This is the same sum that was shown in the steps in Figure 22. The number of terms in such a summation is called the "bond dimension" of a bond 🦴, and, it is nothing but the number of terms we take in the sum of Figure 22.

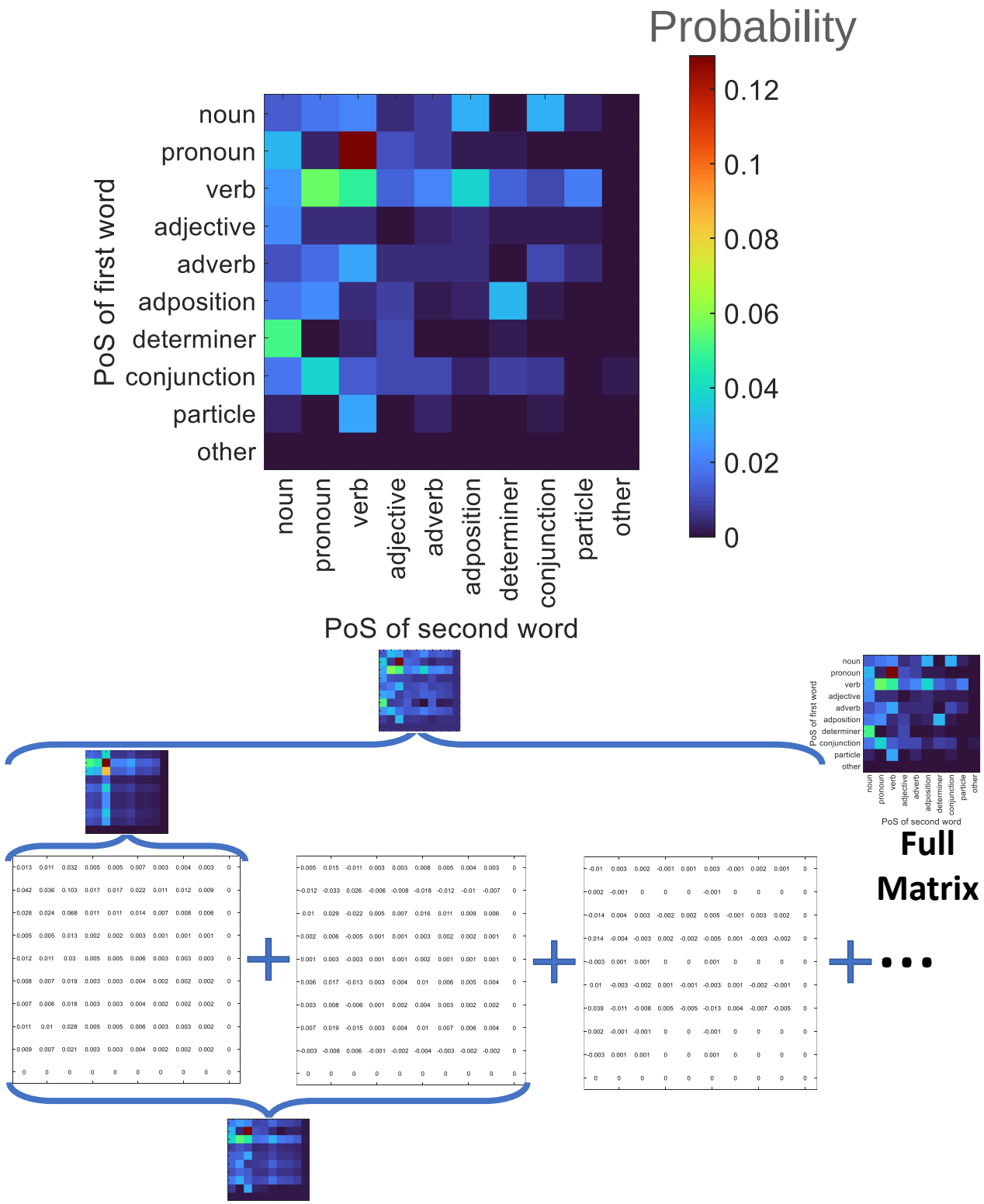


Figure 24: Our original PoS matrix in top and bottom left (same as in Figure 9, but now plotted in color) and its different reconstructions from one, two, and three singular value decomposition terms (in three bottom panels). The individual matrix elements of the summation terms are also shown. If you zoom in to see them in more details, you'll see that they are fast decreasing, so we can safely neglect the last few terms.

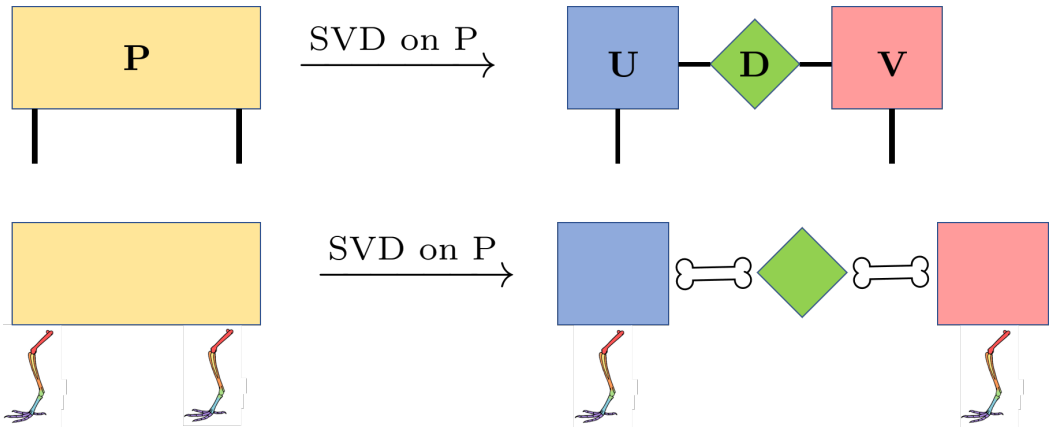


Figure 25: SVD as a diagram

Exercise 4.3

In this exercise, we will come up with a neat-looking formula for the amount of compression of a general matrix which is compressed using SVD with a given bond dimension. Suppose, a matrix A has size or dimension $m \times n$.

a) What is the number of entries in this matrix A?

b) Now, we perform SVD followed by compression for A, where we keep the bond dimension of d. What is the number of entries for all the factor matrices that we have for A?

[Hint: Look at Fig. 22 - where the matrix M is 4×5 dimensional. With SVD and compression, in the last line of the figure, we keep the bond dimension (or the number of 🦴 indices) to 2. What are the number of entries in the last line of the figure? How will this depend for the general case with m,n and d?]

4.4 The Final Piece: Tensor Networks and the Caterpillar

"That we'll forever be tied together."
- The Hill We Climb

A quick recap of what we have got so far: taking help from our text analyzer robot 🤖, we counted joint appearances of different parts of speech pairs in the poem we were analyzing, expressed that information in terms of a matrix (a two-legged object 🦩), and compressed that matrix a bit using SVD. We also understood that PoSs of words will have connections which extend all the way along a sentence. Thus, modeling that will need a tensor for the joint probabilities of multiple words in a sentence, so that, we can predict what are the most likely parts of speech to appear at the end of an unfinished sentence. However we were stuck when we anticipated the immense size of that tensor, which seemed to grow exponentially as we increased the length of the tensor.

How do we compress this long tensor? In the last section, we saw how we can compress matrices, the two-legged objects 🦩, by simply "adding a hidden joint" 🦴 in between them. Let's apply this technique again, and, add a hidden joint 🦴 between all consecutive leg pairs of our "caterpillar 🐛 tensor". We will not go into too much details, but the process of singular value decomposition for tensors is very similar to what we did for matrices, and the process can be captured in the flowchart cartoon in Fig. 26. Like the cartoon shows, it has to be done in steps - adding 🦴s one-by-one between each pair of consecutive legs. This introduces a lot of bond indices (or 🦴s). And just like the case of matrices, we have a compression of the data stored in the tensor by discarding smaller terms in each SVD – and the compression is more

dramatic this time. The resulting caterpillar-with-joints tensor that we got has different names – Tensor Train, Matrix Product State (MPS), and so on. It is useful for fields ranging from data science to quantum physics, the subject that describes how small things like atoms and electrons behave. Yes, you read that right! The same object that describes the structure of English sentences is also an essential tool for predicting how electrons behave. That is the beauty of math, and of many-legged objects .

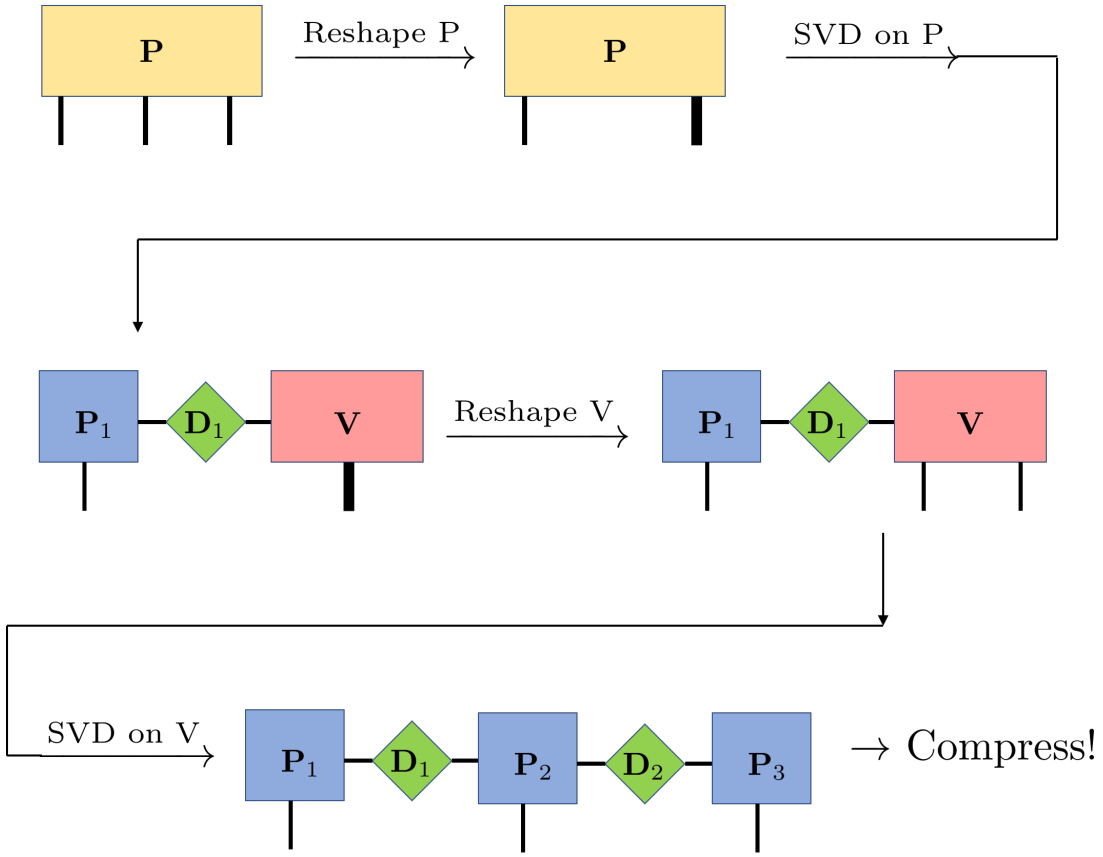


Figure 26: Tensor to Tensor Network, for PoS probability distribution of 3-word lines

So, just like we created the PoS matrix from the poem "The Hill We Climb", we can create a PoS tensor from any text. But in order to do that, let's take a novel this time, because the poem does not contain enough words or lines to give us a good structure of English grammar of long sentences/word clusters. So, for this

purpose, we crunch the words novel "Little Women" by our text analyzer 🤖, which counts the number of times a given PoS string occurs, and gives out the probabilities. We take that information to create the multi-legged tensor $\mathbf{P}$ 🕷. Then, we use the tricks we just introduced to make a Matrix Product State 🐛 out of this tensor, and thereby compressing it.

Let us estimate the amount of compression now. Can we get a neat formula like the one we got in the last Exercise? We are considering 10 PoS categories as usual, and suppose, we are considering lines of length 3. So, without any compression, we need to specify 10^3 numbers. What if we use the Matrix Product State 🐛 with 3 legs, like the one showed in Figure 26? Well, starting from the left, suppose we keep d_1 terms in the first joint 🦴, and d_2 in the second. So, the bond dimensions for the 2 hidden bonds or joints 🦴 are d_1, and d_2, from the left to the right. Now, for each of these hidden bonds, we need to store the diagonal elements, just like the case for matrices (refer to Fig. 22 if you need). That is $d_1 + d_2 = \sum_{i=1}^{2} d_i$ numbers. Note how we have mentioned the lower and upper limits (1 and 2) of the summation index i in the summation sign here. Then, just like in the last exercise, we need to store the elements of $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$. How big are these objects? Well, the ones at the two ends, $\mathbf{P_1}$ and $\mathbf{P_2}$, are matrices – they have got two legs (or a leg and a joint 🦴, if you like that way) each 🦩 (check it for yourself from the Figure 26!). Their sizes are $10 \times d_1$ and $d_2 \times 10$ respectively, very much similar to what we had in the last exercise and Fig. 22 – that makes it $10d_1 + 10d_2$ numbers. But, how about the middle one, $\mathbf{P_3}$? Fig. 26!) reveals that that one has got three legs 🌵 (one outward-pointing leg and two hidden joints 🦴). It is a 3D tensor, or a "cube of numbers" (the cartoon in Figure 18) will be helpful to look at), having size $d_1 \times 10 \times d_2$. Thus, it stores $10d_1d_2$ numbers. So, adding up ev-

erything, we need $\sum_{i=1}^{2} d_i + 10(d_1 + d_1 d_2 + d_2)$ numbers, instead of 1000. Pluggin in different values of the bond dimensions d_1 and d_2, you can estimate the compression, using this formula.

What about word strings longer than 3? For a general string of length L, without compression, we need 10^L numbers. For the Matrix Product State 🐛, there will be $L-1$ hidden joints 🦴 and we'll have the same number of bond dimensions, let's call them d_1, d_2,...,d_{L-2}, and d_{L-1} respectively, from left to right. In this general case, we need to store these many diagonal numbers like before – that makes it $d_1 + d_2 + ... + d_{L-1} = \sum_{i=1}^{L-1} d_i$ numbers. Again, we would still have two matrices 🦩 the at two ends of the caterpillar 🐛, having sizes $10 \times d_1$ and $d_{L-1} \times 10$ respectively. But, in between, we'll have $L-2$ 3-legged tensors 🌵, having sizes (from left to right) $d_1 \times 10 \times d_2$, $d_2 \times 10 \times d_3$, $d_3 \times 10 \times d_4$,..., $d_{L-3} \times 10 \times d_{L-2}$, $d_{L-2} \times 10 \times d_{L-1}$. Adding up everything, it is $\sum_{i=1}^{L-1} d_i + 10(d_1 + d_1 d_2 + d_2 d_3 + ... + d_{L-3} d_{L-2} + d_{L-2} d_{L-1} + d_{L-1})$ numbers. Using the summation sign, we can write it in a more compact form as $\sum_{i=1}^{L-1} d_i + 10\left(d_1 + \sum_{j=1}^{L-2} d_j d_{j+1} + d_{L-1}\right)$. Plug in numbers again to see compression in action, using this formula. But for a real example, keep reading on!

Let's get back to out novel "Little Women". We were looking at PoS distributions among 4-word lines. Just like the case with matrices, let's look at how well we can retrieve our original PoS probabilities from the Matrix Product State (MPS) 🐛. The bad news is that unlike last time we cannot visualize them now, since they are high-dimensional, and most of our visualization capacity ends at three dimensions. But we can compare the actual probabilities of some PoS strings, with those found from the Matrix Product State after compressing the information.

Figure 27 shows this data for 4-word clusters, with each word being one of the 10 PoS types. One the left are the strings which appear the most in the novel, and on the right are 10 others which have very small probability and are chosen randomly. For each of these 4-word clusters, we estimate the actual probability from the novel (note - for 4 word clusters, we can still store the information of the whole tensor in our computer), and compare that with the result from a compressed tensor network. Note that the two probabilities are quite close to each other. Thus the compressed tensor network shows a good reconstruction for the PoS strings shown, both for the most frequent in the novel and the random ones, which occur only a few times (or are outright absent) throughout the novel.

PoS sequence of 4-word sentences	Probability		PoS sequence of 4-word sentences	Probability	
	Actual	From MPS		Actual	From MPS
pronoun verb adverb adjective	0.026570	0.026579	verb verb determiner noun	0.0096618	0.0098291
pronoun verb pronoun verb	0.024155	0.024185	adverb pronoun verb adjective	0.0048309	0.0039936
verb particle verb pronoun	0.024155	0.024115	adverb noun adposition pronoun	0.0024155	0.0024042
pronoun verb pronoun noun	0.021739	0.021793	verb pronoun determiner adverb	0.0024155	0.0021565
adverb verb pronoun verb	0.021739	0.021761	adverb adverb particle adverb	0.0024155	0.0015078
adverb adverb pronoun verb	0.019324	0.019313	verb pronoun verb determiner	0	0.00071694
verb noun adposition noun	0.016908	0.016916	adjective noun particle verb	0	0.00056281
verb adjective adposition pronoun	0.016908	0.016554	verb adposition verb adjective	0	0.00045884
verb pronoun verb noun	0.014493	0.015015	pronoun pronoun verb adverb	0	0.00034626
verb particle verb adverb	0.014493	0.014829	verb determiner determiner adverb	0	0.00029436

Figure 27: Comparison of original and reconstructed-from-MPS probabilities for top 10 and random 10 4-PoS strings of the novel "Little Women".

What about the compression? Well, originally, we had $10^4 = 10000$ numbers. For the particular Matrix Product State that gives us the numbers of Fig. 27, we took $d_1 = 6$, $d_2 = 14$, and $d_3 = 5$ (we'll give you the code where we did all these, if you want). Plugging these and using $L = 4$, our formula above tells us that

we need only $6 + 14 + 5 + 10(6 + 6 \times 14 + 14 \times 5 + 5) = 1675$ numbers to store, taking a volume less than a fifth of the original 10000 numbers. For longer lines, this will be even more dramatic!

We are almost there. Given that our caterpillar tensor  , or the Matrix Product State (MPS) whatever we call it, reproduces the empirical probabilities really well, and is compact enough to be stored in computers, we can make full use of it. In particular, a simple way to predict how to finish a sentence would be to read the PoS string of the words already typed, and then looking for the PoS that is most likely to come after that (predicted from the Matrix Product State). Then, maybe pick up a word from that PoS. Your sentence is likely to make grammatical sense, even though it might still sound weird, because we are only considering PoSs of words, not words themselves.

Of course, real-world text AutoFinish algorithms are more sophisticated than that. They focus on words themselves, and not particularly to the Parts of Speech of the words, and deals with joint probability distributions of actual words themselves. Needless to say, that involves more complicated structures, understanding contexts of different words in a sentence and nuances of a language, and crunching bigger volumes of data – and a whole lot of fancy artificial intelligence toolbox is being developed for that purpose. While that discussion is well beyond the scope of this packet, we hope that some of you reading this now will end up joining the crew of researchers and developers dedicated to building language models.

The tool that we have used, Tensor Networks, is only one example of the tools for making a compressed model of high dimensional data like language. In real life, auto-complete algorithms

are more complicated than this - they not only identify the PoS of words, but the words themselves. The auto-complete algorithms will not only learn from a single novel, but would have created a compressed model for a language from many texts collected from all over the internet, and also from the text you have written. But even studying Tensor Networks for just the PoS data has led us to the rich mathematical structures underlying complex data problems which our computers solve every day. Moreover, we found out that it is actually not that difficult once we understand the math behind these tools!

5 Summary of it all

"But in all the bridges we've made,
that is the promise to glade,
the hill we climb.
If only we dare."
- The Hill We Climb

5.1 What did we learn this summer?

This brings us almost to the end of the journey which started with us trying to understand how a computer algorithm 💻 or a text analyzer 🤖 might crunch the text we write. On the way we read a beautiful poem and learned about probabilities. Moreover, we saw how we can 'draw' probabilities as cartoons - boxes with legs. This way of representing probabilities as pictures go a long way towards simplifying the complicated mathematics to playing with pictures.

We learned that these pictures have a scientific name, Tensor Networks. We learned how to efficiently store tensor networks by compressing the probability tensor. This step is essential for many different aspects of data processing - our lives run on data, but our storage devices don't have unlimited memory. So we must compress data without compromising too much on the information contained in the data. We learned how the process of singular value decomposition of matrices help towards that goal.

Tensor Networks, in fact, have applications beyond probabilities and textual analysis. It turns out that tensor networks help

physicists understand quantum mechanics better! Quantum mechanics is a branch of physics that determines the underlying laws of all things in nature. However, it turns out that even though results from quantum mechanics match what we see in nature, it is very difficult to simulate quantum mechanics of many objects in our computers. If we thought of nature as a big computer, the memory of that computer would seemingly be many many many times larger than the number of atoms in the universe! Clearly, nature is cleverer than that! This is very similar to the storage puzzle we saw in our example of the probability tensor of words in a sentence. Even though the actual size of the exact probability tensor is very very big (for 15 words, the size was 8000 terabytes, or the memory of about 8000 standard desktop computers!), we could do good job of textual analysis using a tensor network with far fewer entries by clever compression of the data. This property helps us in physics too. It turns out that tensor networks allow us humans to glimpse into the incredible complexity of nature without needing to build a computer larger than our universe!

Remember the picture of tensor networks that we saw in the beginning (see Fig. 28)? It turns out that this is a tensor network that helps physicists model Black Holes in some funny (but simpler) alternate universe! By learning about tensor networks, you are well on your way towards uncovering the joys of the universe! It will take us some more summers to get there, but we took the first steps in this course.

Let us take a step back from Black Holes and get back to the application of tensor networks that we tried to look at in this course - modeling and analyzing language.

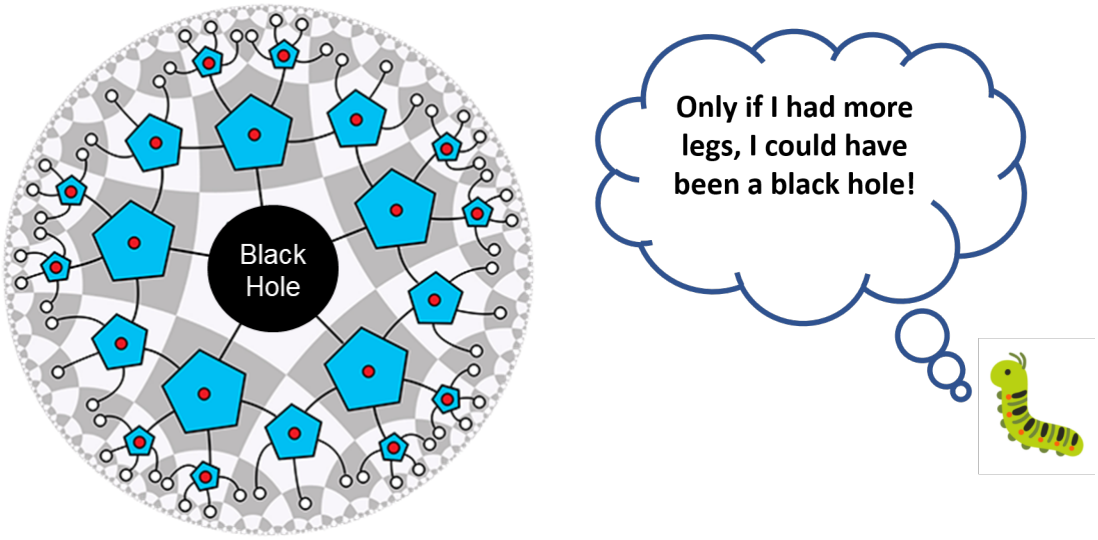


Figure 28: One of the examples of tensor networks we saw at the beginning is actually a model of how physicists think Black Holes work! It is a network of tensors with five indices (or five legs ⭐) connected together via many hidden joints.

5.2 Impact of language models - why it matters?

We have discussed the importance of compressing the probability tensor for efficiency in this course, but this begs the question - are we erasing the less-represented aspects of data in the process, and does this reflect bias and discrimination of our society instead of more objective rules, like the language grammar?

This is relevant for understanding the scope of Artificial Intelligence (AI) which, for example, helps in the auto-complete feature for texting. AI has to be first trained on some training data (some text like "The Hill We Climb" or "Little Women", in our case). For an AI learning a model of the language, the connections which are ignored due to compression or efficiency might be less represented in the training text simply because of the rules of grammar

(for example, a sentence is less likely to end with a conjunction). But this could also happen because the text is not representative of the diversity of the language users, or is a product of historical entitlements of the writer or subject of the text. For example, suppose you ask the auto-complete feature of a phone to complete the sentence, ' ___ like to do math', and it might so happen that the auto-complete is more likely to suggest Boys than Girls. But this is because the auto-complete feature learned this bias from the text that was used to train it (maybe, from newspaper reports that focus more on men doing math), and not because there is any reason why girls are less likely to do math than boys.

Furthermore, Languages carry their own structural bias. For example, in English, the pronoun 'you' is used as both singular and plural. But singular 'they', even though it was first used (not erroneously) in the fourteenth century, was dropped off until recently. Indeed, languages carry the identity and diversity of the people using them and meaning of words depend on who is speaking. Nuances of languages change over time as we get more aware of our history and our present. While it might be tremendously hard to capture all of these features in even the most state-of-the-art language models, we cannot afford to lose any of them either. Therein lies the danger!

If we are not cognizant of this, the machine can learn the pre-existing biases in its training data, and worse, propagate it in all further predictions. This is further exacerbated by the fact that most creators of artificial intelligence technologies have traditionally formed a very homogeneous demography. But the technologies that they have created (like language prediction, facial recognition software, college essay scoring algorithms, criminal justice algorithms etc.) are being unleashed over the entire population, with

direct implications in fields like face identification, mass surveillance, college hiring and admission, and prison sentencing. In absence of proper regulation, accountability, and empathy, this can and does indeed cause immense pain and further discrimination along historical lines.

In the data driven world of today and tomorrow, we need to be conscious of all this - and understanding how and what a machine learns is the first step. While this packet introduces you to how math can be so powerful for all these techs, let us not forget that it is not *all* math. It is about the people, it is about our society, culture, politics, economics, and history, and who we are. You need not be a mathematician, scientist, or a software developer to contribute to the field of tech. You can be a literary scholar, a social scientist, a historian, a linguist, or maybe a philosopher, and your unique contribution might be very much needed in the field of tech. But you will need to have empathy and accountability.

Anyways, now that you have got a sense of how textual analysis can work, maybe you can look around and try to find instances where data prediction tools, like the auto-complete feature in your phone, could be biased. Better yet, maybe one day you will create a better and more just textual analysis engine!

"We will not be turned around,
Or interrupted by intimidation,
Because we know our inaction and inertia
Will be the inheritance of the next
generation.
Our blunders become their burdens.
But one thing is certain:
If we merge mercy with might, and might
with right,
Then love becomes our legacy,
And change, our children's birthright."

– The Hill We Climb

References

[1] 3Blue1Brown. Linear transformations and matrices | chapter 3, essence of linear algebra. https://youtu.be/kYB8IZa5AuE.

[2] 3Blue1Brown. Matrix multiplication as composition | chapter 4, essence of linear algebra. https://youtu.be/XkY2DOUCWMU.

[3] Tae Danae Bradley. Viewing matrices and probability as graphs. https://www.math3ma.com/blog/matrices-probability-graphs.

[4] Tae Danae Bradley. Matrices as tensor network diagrams. https://www.math3ma.com/blog/matrices-as-tensor-network-diagrams.