A gentle introduction to

Generative AI: Theory and Practice

Grandee Lee

August 18, 2025

Disclaimer
This is a work in progress, feedback is welcomed.

Website
https://github.com/grandeelee

Colophon
KOMA-Script and LaTeX using the kaobook class.

Let me think of something.

– Grandee Lee

Contents

List of Figures

List of Tables

The Basics

Introduction 1

1.1 What is a model?

We know that a model is a *representation of reality*, such as a miniaturized globe, or a toy solar system. We need such models because they tell us something about the reality. For example, the globe will tell us the countries location and explain why there are different seasons between the southern and northern hemisphere. A model of the solar system will tell us the relative position of the planets. The model may not be a physical object, it could be a formula, such as a formula about projectile motion that tells us exactly how the ball will behave for its entire duration in the air. It predicts the ball's future. A model of the planetary motion allows us to know everything about the position and speed of the planets. And this is exactly why models are necessary, because it allows us to make predictive decision of a future outcome. A model is also a *simplified reality* that it only inform us about the phenomenon that we are interested in, each model taking care of a specific cause and effect. So it may not be entirely accurate. It's part of our minds' endeavor and struggle to explain and isolate each effect to its own cause.

One of the reasons why we want to create a model in the first place is for *prediction*, we want to gather enough information on the behaviour of the object and based on that, simulate the future. In this sense, when we say using the model for *simulation* and using the model for *prediction*, we are talking about the same thing. In our minds, we are constantly modelling situations around us, we cannot survive without a model to help us understand economics, language, law, etc. There is a model for everything. Many models around us are used for prediction, for example, we have stock market price models that are used for price forecast, and we have weather pattern models that are used for weather forecast. It is in our human nature to find certainty in midst of chaos, why do we have a model of the solar system, well so that we know when is the day and night and winter and summer. In fact if we think about it, the calendar that we have is a prediction, of the days to come.

Figure 1.1: A model is a simplified description of some phenomenon that interest us.

Not all models or formulas need to be mathematical, many models that describe psychological behaviours or social interactions are a collection of rules. Rules are models as well, although in this course we are not focusing our attention on those models. Another relevant example is grammar which is a model for synthesizing language. Grammar, as we now know it, is a rule-based model, it describes the order of subject, verb and object as well as the various parts of speech. However, we will learn a method to formalize these rules in a mathematical model.

Figure 1.2: We need model for prediction in our daily life.

There is no such thing as a perfect model. The moment we have a model, it is a simplified version of the "reality" it represents. Think about the atomic model, from pudding to planetary to energy band to probabilistic spheres. The complexity of the model changes according to how accurately we want to represent the reality.

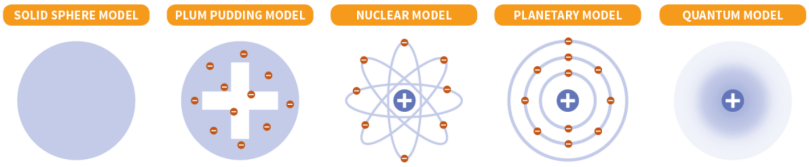

Figure 1.3: The evolution of atomic models

1.2 How accurate should be our model?

All models are simplified versions of reality. ChatGPT which is a large language model is a simplified version of the human language and knowledge. The projectile motion is a simplified model, it ignores air resistance, which we can add in, and it ignores the vibration of the object in response to the air resistance. We can add them in, in other words; increasing the detail of the model. But it is necessary? When is it necessary? When we are talking about very high-speed objects, we increase the complexity of our model to describe a more accurate phenomenon. But even so, there is a limit to how accurately can we model that phenomenon. *There is always a limit to the accuracy of our description of the reality*. So now let's make a very rash conclusion first, i.e. model complexity positively correlates to accuracy, within some upper bound and we only do that when the situation arises. If not we keep to the simple model. Why? What is the benefit of having a simple model? You may refer to the evolution of atomic models and how it aids our understanding.

There is always a limit to the accuracy of our description of the reality. Always. And a cost to it.

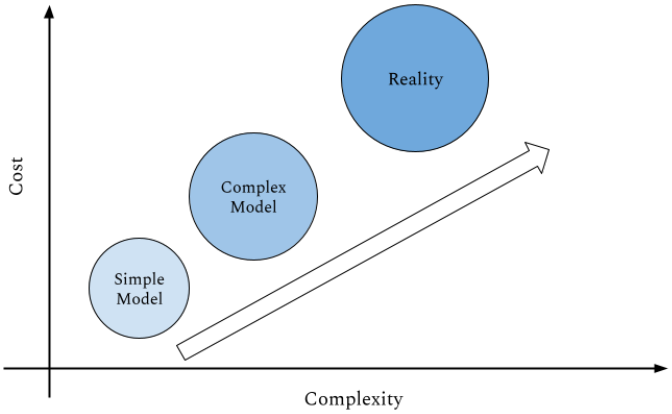

Figure 1.4: There is a reason why we have simple models, they are fast and cheap. On the other hand, they may be less accurate.

1.3 What's wrong with simplicity?

There is always a *trade-off*, whenever we design a model, a simple model will be quick, but understandably less descriptive, less detailed or less accurate and sometimes just wrong. Like "all adjectives come before a noun", not really, adjectives can come before another adjective. But it is easy to learn and it is fast. On the other hand, we can have more complex models that describe reality in greater detail, but we cannot be sure if that detail is real or noise. Like the daily fluctuation in the stock market. So before we get too philosophical, let me highlight the two principles; complexity may help accuracy (i.e. increasing details), and simplicity may reduce noise (i.e. reducing details). Here I am a bit loose with my definitions of accuracy, noise, and error, and relate them to the concept of

more or less details. A complex model leads to more details or increased accuracy could mean tracing a stock market price better or producing a higher quality picture, conversely, a simple model could mean a poorer trace of the graph. But if the model is good enough, simplicity doesn't imply it is wrong, a good and simple model can ignore all the noise and predict an accurate general trend. This is related to another principle called *Occam's razor* which prefers a simpler explanation or solution. A complex model could have added in some wrong fluctuations and totally missed the trend. The two principles are actually one, *bias vs accuracy trade-off*.

1.4 Sequence modelling

Generally, all models are predictive or used for simulation purposes, but prediction may not involve the future. For instance, in image recognition, usually, we don't say we are predicting a future event. It is just predicting an outcome.

What prediction involves the concept of time? It is the prediction about a chain of events, based on the previous events. So market price, and weather status, all of which requires us to gather the previous data points about price and weather, as well as many other related event *in the past to predict the next data point*. We call this type of modelling, sequence modelling. And *language is also a sequence*.

N-gram Language Model 2

2.1 What is language?

Language is a fascinating and intricate system of communication that has evolved over thousands of years. At its core, language is a series of symbols – be they sounds, letters, or gestures – that convey meaning. These symbols, often referred to as 'tokens', are arranged in sequences to construct words, sentences, and entire narratives. But language is far more than a string of tokens; it is a dynamic and complex structure where the significance of each token is often defined by its relationship to others, an aspect we refer to as 'context'. For example, the word 'bright' has a different meaning when used in 'bright light' versus 'bright student'. This variation in meaning underscores the importance of *tokens* and their *context*.

What is a token? In Natural Language Processing (NLP), a token is a fundamental unit of analysis. The concept of a "token" is akin to the concept of a "word" in everyday language. So for now, let's use "token" and "word" interchangeably. For example, in the sentence "The cat sat on the mat," each word would typically be considered a separate token ("The", "cat", "sat", "on", "the", "mat"). Tokens can also include punctuation marks and special characters. For instance, in "Hello, world!" the comma and the exclamation mark are tokens in their own right. In more complex scenarios, a token might not correspond neatly to a word as understood in everyday language. For instance, contractions like "don't" can be split into two tokens ("do" and"n't"), and compound words or hyphenated phrases might be treated as a single token or split into multiple tokens depending on the specific requirements of the NLP task. But this is for another day.

Exercise 2.1.1 Now we know what are tokens, given the sentences in the toy corpus, Example 2.4.1, build our first set of vocabulary. Vocabulary is the set of tokens which the system will recognize.

Exercise 2.1.2 Explore the different tokenization packages; 1) NLTK and 2) tiktoken. What are the differences in the tokenized results?

2.2 How do we model language?

We model language not so differently as we model the stock price. It is a prediction model. This perspective views language as a series of sequential tokens, typically words or characters, where the goal is to predict the next token in the sequence based on the previous ones. Modelling language and the stock market as sequences reflects a shared underlying principle in both domains: the belief that future events or tokens (whether words in a sentence or stock prices) can be predicted

based on preceding patterns. This predictive approach is grounded in the assumption that sequences – be they linguistic or financial – contain patterns or dependencies that can inform about future occurrences.

While both language and stock market modelling rely on sequence prediction, there are notable differences:

Nature of Data Language: The data is textual and often has a clearer structure (grammar, syntax) and more direct relationships between sequential elements (tokens). Stock Market: The data is numerical and heavily influenced by a complex web of factors, including economic indicators, company performance, and broader market trends, many of which can be unpredictable or non-linear.

Predictability Language: There is a higher level of predictability due to grammatical rules and common usage patterns. Stock Market: The market is famously difficult to predict with high accuracy due to its susceptibility to a wide array of unpredictable factors, including human behaviour and external events.

Objective Language Modeling: The primary goal is to generate or understand language in a way that is coherent and contextually appropriate. Stock Market Modeling: The aim is often to forecast future prices or trends for financial gain, requiring a focus on minimizing risk and dealing with uncertainty.

The reason that language models can work as a sequence prediction is very much because of contextual dependencies. Language is inherently sequential and context-dependent. Words gain their meaning and function within the context of surrounding words. A predictive approach allows models to understand and utilize these contextual dependencies.

2.3 Context

The context in language is akin to understanding not just the notes in a musical composition but also how they harmonize to create a symphony. It's what allows us to grasp that 'He's cold' could mean something entirely different in a medical drama versus a detective story. Context encompasses not just the immediate surrounding tokens but also the cultural, social, and situational nuances that shape meaning. However, when we talk about "context" words in NLP, they refer to a very specific set of words. It is good to clarify beforehand the concepts of "context words", and they are usually in contrast to the "focus word".

Context Words vs Focus Word

A focus word, often referred to as the "target word", is the word of interest in a given sentence or phrase whose meaning or usage is being specifically analyzed or predicted. In language models, the focus word is typically the word for which the model is trying to predict something, such as its probability of occurrence, its classification, or its relationship with other words. For example, in the sentence "The cat sat on the mat," if we are trying to predict the likelihood of the word "sat" given the other words in the sentence, "sat" becomes the focus word.

Context words, on the other hand, are the words surrounding the focus word. These words provide crucial background information that helps in understanding or predicting the focus word's meaning, usage, or other linguistic attributes. In language models, context words encapsulate the

surrounding linguistic environment of the focus word. Continuing with the previous example, in the sentence "The cat sat on the mat," if "sat" is the focus word, then "The", "cat", "on", "the" and "mat" are the context words. These words help in predicting or understanding the meaning of "sat" in this specific instance.

By now, we should try to see language as a sequence of tokens whose meaning is determined by the context. If the context words are randomly selected, even if they follow the grammatical structure, the sentence is still meaningless. Consider the following sentence, "Colorless green ideas sleep furiously". This sentence is grammatically correct based on rules but meaningless. This example brings us to our next point, how do we model language that is both meaningful and grammatically sound? if we want to model language, we have to consider context. Let's start with the simplest way to model language, the N-gram. And guess what, it is actually a generative model.

2.4 What is an N-gram Language Model?

N-grams are continuous sequences of 'n' tokens from a given text or speech. The most common types of n-grams in natural language processing (NLP) are unigrams (single words), bigrams (two-word sequences), and trigrams (three-word sequences). They play a pivotal role in language modelling and are instrumental in predicting the next item in a sequence. To understand the concept of n-grams, let's consider a small corpus of seven sentences, each reflecting a different opinion:

Example 2.4.1 (A toy corpus) This corpus contains 7 sentences expressing various views.

1. Electric cars are the future of transportation.
2. Traditional gasoline vehicles have a long way to go.
3. Renewable energy will completely replace fossil fuels.
4. Solar panels are not as efficient as people think.
5. Vegan diets are healthier than omnivorous diets.
6. Meat consumption is essential for human health.
7. Technology will solve all of humanity's problems.

What is a corpus?

A corpus (plural: corpora) in the context of Natural Language Processing (NLP) refers to a large and structured set of texts. These texts are used to statistically analyze and help computers learn about the structure and usage of a language. A corpus can range from a collection of documents, sentences, paragraphs, or even single words. Corpora are foundational to machine learning models in NLP. They serve as training data for models to understand and generate language. For instance, chatbots are trained on vast corpora to provide relevant responses to user queries. There are different types of corpora, *e.g.*:

Monolingual Corpora Contains texts in a single language. For example, the complete works of Shakespeare can be considered a monolingual corpus.

Parallel Corpora Contains texts in two languages. It's often used in machine translation. For instance, the European Parliament Proceedings Parallel Corpus contains texts of the European Parliament's proceedings in 21 European languages.

Annotated Corpora These are corpora where the text is annotated with additional information, such as part-of-speech tags, named entity labels, or syntactic trees. The Penn Treebank is a famous annotated corpus used in syntactic studies of English.

Specialized Corpora These corpora are designed for specific purposes or fields. For example, medical corpora contain texts related to medicine, which can be used to train medical chatbots or assist in medical research.

2.5 Tokenization

Tokenization is the process of dividing text into a series of meaningful pieces, called tokens. These tokens can be words, characters, or subwords and are used as the input for various tasks in natural language processing (NLP), including language modeling. By defining how text is split, it directly influences the granularity of the learned linguistic features. Because a continuous stream of text are broken down into manageable units, *e.g. don't* $\rightarrow$ *do* + *n't*, which allows models to more easily learn patterns and linguistic structures, thereby enabling better understanding and generation of languages. Effective tokenization strategies directly impacts the effectiveness of subsequent processes in NLP workflows, such as parsing, syntax analysis, and semantic analysis. High-quality tokenization ensures that the model understands the input text thoroughly, enabling it to predict or generate subsequent text with greater precision.

Character-Level Tokenization involves breaking down text into individual characters. This method allows models to understand the text at the most granular level, *e.g.*

```
Input: "Hello world"
Output: ["H", "e", "l", "l", "o", " ", "w", "o", "r", "l", "
    d"]
```

Word-Level Tokenization splits text into tokens at the word boundaries. This common approach is straightforward and works well for languages with clear word delimiters, such as spaces in English, *e.g.*

```
Input: "Hello world"
Output: ["Hello", "world"]
```

Sentence Tokenization divides a text into sentences. This is useful for tasks that require understanding or generating text at the sentence level, like summarization and translation, *e.g.*

```
Input: "Hello world. It is a sunny day."
Output: ["Hello world.", "It is a sunny day."]
```

Byte-Pair Encoding (BPE) is a hybrid between word-level and character-level tokenization. Originally used for data compression, BPE

iteratively merges the most frequent pair of bytes or characters in a sequence. In NLP, this method helps in managing the vocabulary size efficiently, allowing for the encoding of rare words through subword units, *e.g.*

```
Input: "lower newer mower"
Start by tokenizing to letters: ["l", "o", ..., "e", "r"]
The most frequent pair "er" gets combined
Until a satisfactory level of segmentation is achieved
```

2.5.1 Preprocessing of Text

When handling punctuation, the process becomes slightly more complex as punctuation can affect the meaning of the text significantly and should not be discarded. Here are several strategies outlining the treatment of punctuation in word-level tokenization:

Treat Punctuation as Separate Tokens This is a common strategy in which punctuation marks are treated as standalone tokens. This approach allows models to recognize punctuation explicitly, which can help in understanding sentence boundaries and syntactical nuances within the text, *e.g.*

```
Input: "Hello, world! It's a beautiful day."
Output: ["Hello", ",", "world", "!", "It's", "a", "beautiful
    ", "day", "."]
```

Attach Punctuation to Neighboring Words Depending on the analytical goals, punctuation can be attached to the preceding or succeeding word, creating tokens that include punctuation. This method can be useful for applications like sentiment analysis, where the sentiment might be strongly tied to the punctuation used, *e.g.*

```
Input: "Wow! Really?"
Output: ["Wow!", "Really?"]
```

Ignore Punctuation In cases where punctuation does not add significant value, such as when performing statistical text analysis or certain types of machine learning operations, it may be removed. This simplifies the tokenization process and may reduce noise.

Utilizing Advanced Tokenization Tools Automated tools and libraries (e.g., NLTK, spaCy, or the tokenizer component of larger NLP frameworks) broadly use one or a combination of the above strategies but with more nuanced rules and exceptions handling. These tools ensure that punctuation is handled appropriately according to the context and the nature of the task.

Before tokenization, text often undergoes several preprocessing steps to improve the quality of the tokens generated. Common preprocessing methods include:

Lowercasing Converting all the characters in the text to lowercase.

Removing special characters and numbers Stripping out unnecessary punctuation and numerals which may not be needed.

Expanding contractions Changing "can't" to "cannot" to make each token more meaningful.

Eliminating stopwords Removing commonly used words (such as "and", "the", etc.) that might not contribute much meaning in analysis.

2.5.2 Limitations of Tokenization

While tokenization is essential, it has its limitations. Language nuance, such as sarcasm or idioms, can often be lost during tokenization, especially when using subword tokenization which will break the words. Additionally, tokenization can struggle with handling multi-word expressions that might carry meanings different from the individual words combined. Handling morphologically rich languages (like Turkish or Finnish) where words often contain a lot of embedded information is also challenging.

2.6 Bigram Language Model

Given the above corpus in Example 2.4.1, how do we learn a pattern? In the world of Natural Language Processing (NLP), understanding the structure and pattern of language is crucial. One of the foundational concepts used to capture this pattern is the idea of a “bigram”. A bigram, also known as a 2-gram, is a sequence of two adjacent words from a string of tokens or words. It’s a specific case of the more general concept called “n-gram”, where “n” represents the number of words grouped together.

Consider the sentence: "Electric cars are the future of transportation" The bigrams in this sentence are:

Table 2.1: Bigrams are all the pairs of tokens in the sentence.

Electric	cars					
	cars	are				
		are	the			
			the	future		
				future	of	
					of	transportation

2.7 Building a Bigram Language Model

A bigram language model uses the conditional probability of a word given the preceding word to predict text sequences. Let’s walk through the steps to create a basic bigram language model and provide relevant code snippets in Python.

Listing 2.1: Read input text

```python
# read a text file "corpus.txt" and store as a list of list
def read_file(path):
    with open(path, "r") as f:
        lines = f.read().split("\n")
    lines = [x.split(" ") for x in lines]
    return lines
```

Listing 2.2: Creating the bigram model

```
# create a dictionary of bigrams with frequency counts
def create_bigram(lines):
    bigram = {}
    for line in lines:
        for i in range(len(line)-1):
            key = line[i]
            if key in bigram:
                if line[i+1] in bigram[key]:
                    bigram[key][line[i+1]] += 1
                else:
                    bigram[key][line[i+1]] = 1
            else:
                bigram[key] = {line[i+1]: 1}
    return bigram
```

2.8 Generative Bigram

Once we have generated the bigram language model, we can use it to predict the next word based on the preceding word. For example, given the word "Vegan", the model might predict "diets" as the next word, given its presence in our corpus.

Listing 2.3: Generate from bigram

```
# generate a sentence using bigram model
def generate_sentence(bigram, start):
    sentence = [start]
    key = start
    while len(sentence) < 20:
        if key in bigram:
        # use argmax first, in temperature section change to
        # random.choices
            next_word = random.choices(list(bigram[key].keys()),
    list(bigram[key].values()))[0]
            sentence.append(next_word)
            key = next_word
        else:
            break
    return " ".join(sentence)
```

Listing 2.4: Main function

```
# main function
def main():
    lines = read_file("corpus.txt")
    bigram = create_bigram(lines)
    print(generate_sentence(bigram, "Vegan"))
```

Exercise 2.8.1 Given the bigram frequency table, 1) what is the most likely word to follow ”Vegan”, 2) what is the probability of the next word?

Exercise 2.8.2 Now that you have learnt how to implement a bigram language model and generate content from that model. Please try to implement a trigram language model.

Contextual “Understanding”: Bigrams help in understanding the context in which words appear. For example, given that the previous word is “New” if our bigram model is trained on a sufficiently large corpus, the bigram “New York” may stand out. This means that bigram model which remembers “New York” as a bigram gives a clearer context than just the words "New" and "York" individually. There is no real understanding here in our commonly understood sense, however it does mean some capacity in differentiating various phrases. This can reduce ambiguity in language processing. For instance, the word "bank" can refer to a financial institution or the side of a river. In bigrams like "river bank" or "commercial bank," the context becomes clearer.

2.9 Data-Driven Opinion in N-gram Models

Now that you have the bigram language model and the way to generate content. Let’s analyse the output content from this model. If we run the code snippet Main function in Listing 2.4 a few times, we may observe the following output,

1. Vegan diets are not as people think
2. Vegan diets are healthier than omnivorous diets are not as efficient as people think
3. Vegan diets are the future of humanity’s problems
4. Vegan diets are not as efficient as efficient as people think
5. Vegan diets are the future of transportation

Exercise 2.9.1 What types of error do you observe in the generated outputs? Can you think of the factors contributing to each type of error that you have identified? bias? hallucination? misinformation?

The opinions reflected in the trigram model are purely data-driven. The model doesn’t “understand” or “believe” any of the opinions; it simply reproduces patterns it has observed in the provided data. For instance, if someone were to input “Vegan”, our bigram model might suggest “Vegan diets are the future of humanity’s problems”, reflecting the word patterns in our corpus. However, this doesn’t mean the model believes vegan diets are humanity’s problems; it’s simply mirroring the data it was trained on.

This highlights an essential concept in NLP: models are inherently neutral and rely heavily on the quality and diversity of the data they are trained on. An n-gram model trained on a narrow or biased dataset will reflect those biases in its predictions. It’s up to the data scientists and developers to ensure that the data used to train these models is as unbiased and representative as possible.

Exercise 2.9.2 (Conflicting Opinion Statements) Example Corpus:

1. Dogs are better than cats.
2. I believe cats are superior to dogs.
3. Chocolate ice cream is the best flavor.

4. Nothing beats the taste of vanilla ice cream.
5. The city is a better place to live than the countryside.
6. I think the countryside offers a more peaceful life than the city.
7. Digital books can never replace physical ones.

Now, using this corpus and your own trigram model, predict the next word based on the previous two words. For instance, if we provide "better than", what might the model suggest as a probable next word? And how is the generated sentence related to our training data.

How to clean data? How to look out for bias?

The opinions represented in our trigram model are purely based on our initial corpus. If our corpus consists of a particular bias or skewed opinion, our model will reflect that. In the above example, we had conflicting opinions on subjects like dogs vs. cats, chocolate vs. vanilla ice cream, city vs. countryside living, and digital vs. physical books. Thus, generating sentences from this model will provide a mix of these opinions, but it's important to remember that these are purely data-driven and not the model's "beliefs". If our corpus only had positive statements about dogs, our model would be biased towards dogs. When working with NLP models, especially ones that involve opinions or sentiments, it's crucial to ensure that the training data is diverse and representative to avoid unintentional biases.

Exercise 2.9.3 (Guess the corpus) Based on the following output example, which category do you think the training corpus belongs to?

"Natural language processing is probably the hardest problem for AI, and the reason is that language is almost in some sense equivalent to thinking." Using language brings together all of our knowledge about how the world works, including our understanding of other humans and our intuitive sense of fundamental concepts. As an example, Mitchell presents the statement "a steel ball fell on a glass table and it shattered." Humans, she notes, immediately understand that "it" refers to the glass table. Machines, by contrast, may or may not have enough contextual knowledge programmed in about materials and physics to come to that same determination. For people, she explains, it's common sense.

2.10 The Mathematics of N-grams

You should by now understand the basic components of N-grams and have the skills to implement a simple n-gram model. In this section, we will formalize the definition of n-gram model, this knowledge will be the foundation for statistical language modeling which in turn is the basis for us to understand neural language models like the Large Language Models. By quantifying the likelihood of word sequences, we can predict text, recognize speech, and more. The mathematical foundation of n-gram language model is based on probabilities and conditional probabilities.

Unigram Probability: The likelihood of a single word occurring in the corpus.

$$p(w_1) = \frac{\text{Count of } w_1}{\text{Total words in corpus}} \tag{2.1}$$

Joint Probability of two adjacent words.

$$p(w_1, w_2) = \frac{\text{Count of } w_1 w_2}{\text{Total words in corpus}} \tag{2.2}$$

However, we often want the conditional probability, or the likelihood of w_2 following w_1:

$$p(w_2|w_1) = \frac{p(w_1, w_2)}{p(w_1)} \tag{2.3}$$

$$= \frac{\text{Count of } w_1 w_2}{\text{Count of } w_1} \tag{2.4}$$

Example 2.10.1 Consider the toy corpus in Example **??**. For the bigram "are the", the probability $p(\text{"the"} \mid \text{"are"}) = \frac{1}{3}$ since "are the" appears once out of the three occurrences of "are".

Theorem 2.10.1 (N-gram language model) *In general we can estimate n-gram probabilities from corpora using the following formula,*

$$p(w_n|w_{n-1}, w_{n-2}, \ldots, w_1) = \frac{\textit{Count of n-gram sequence}}{\textit{Count of (n-1)-gram sequence}} \tag{2.5}$$

Example 2.10.2 Example: In a corpus with the sentence "She loves to play. She loves music.", the trigram "She loves to" appears once. So,

$$p(\text{"to"} \mid \text{"She loves"}) = \frac{\text{Count of "She loves to"}}{\text{Count of "She loves"}} = \frac{1}{2}$$

Exercise 2.10.1 What is the context window of a 7-gram language model?

What is perplexity?

Getting the probability of the a ngram is not the end, to compute the probability of the whole sentence or the whole corpus, we use perplexity. So far we have introduced the conditional probability of the next word based on the previous words $p(w_n|w_{<n})$. To compute the probability of the whole sentence, we can use the following equation,

$$p(W) = p(w_1, w_2, w_3 \ldots w_n) = \prod_{i=1}^{n} p(w_i|w_{<i}). \tag{2.6}$$

We can further approximate the equation by assuming that the current word is only conditioned on the previous word, which is a very bad assumption, of course. But this assumption greatly simplifies the equation to:

$$p(W) \approx \prod_{i=1}^{n} p(w_i|w_{i-1}). \tag{2.7}$$

This $p(W)$ is the probability of this sentence W occurring based on the training corpus. Computationally, we don't use multiplication due to the risk of overflow, so it is more common to compute the log probability.

Averaging the log probability over the total number of words in the sentence gives us,

$$LP = -\frac{1}{n} \log p(w_1, w_2 ... w_n). \tag{2.8}$$

Raising the above value to the power of e gives a very special value called the *perplexity*. Perplexity measures the amount of uncertainty in predicting the next word. Perplexity is

$$PP = e^{LP}, \tag{2.9}$$

where LP is the log probability of the next word. A perplexity of 1 implies that the probability of predicting the next word is 100%, thus it could be thought of as an imaginary equivalent list whose words are equally likely to be predicted by the model. A perplexity of 200 will imply the system cannot distinguish a list of 200 words for the next prediction.

Perplexity is a measure of the language model's ability to predict the next word, lower value *usually* indicate better performance.

2.11 Laplace Smoothing

Exercise 2.11.1 A challenge arises when an n-gram hasn't been seen before in the training data. Its probability would be zero, which can be problematic.

Solution: Add-one smoothing. Each n-gram count is increased by one to ensure non-zero probabilities.

Formula without smoothing:

$$p(w_n|w_{n-1}) = \frac{\text{Count of } w_{n-1}w_n}{\text{Count of } w_{n-1}} \tag{2.10}$$

With smoothing:

$$p(w_n|w_{n-1}) = \frac{\text{Count of } w_{n-1}w_n + 1}{\text{Count of } w_{n-1} + V} \tag{2.11}$$

Where V is the vocabulary size (number of unique words). Drawback: While it ensures non-zero probabilities, it might assign too much probability to unseen n-grams, especially in large vocabularies.

Exercise 2.11.2 How might modern language models tackle the issue of zero probabilities for unseen n-grams? What are potential drawbacks of other smoothing techniques?

The mathematics behind n-grams is pivotal in predicting text sequences. While the basic probability calculation gives a raw estimate, techniques like Laplace smoothing adjust for unseen n-grams. However, as with all models, it's essential to consider the context and the specific challenges of the dataset in use. With a foundational understanding of the mathematics of n-grams, we're better equipped to delve into the intricacies of language modeling and prediction.

2.12 What is temperature?

It is a way to introduce variability. Why do we want to do that? To generate varied content and possibly more interesting content. If you take a look at Co-pilot, it offers three conversation styles, 1) more Creative 2) more balanced and 3) more precise. One of the ways to achieve this and usually the most common way is to vary the temperature. You can think of it as the degree of divergence or randomness during generation.

The most basically implementation of temperature, using random.choices from Numpy.

Listing 2.5: A simple implementation of temperature in generative bigram

```
# generate a sentence using bigram model
def generate_sentence(bigram, start, temp=100):
    sentence = [start]
    key = start
    while len(sentence) < 20:
        if key in bigram:
            p = list(bigram[key].values())
            # introduce temperature parameter
            p = [np.exp(x/temp) for x in p]
            # Do we need to normalize?
            next_word = random.choices(list(bigram[key].keys()), p
    )[0]
            sentence.append(next_word)
            key = next_word
        else:
            break
    return " ".join(sentence)
```

Using the above function, each possible next word is weighted by its probability from the corpus. So that all words will be selected at some point in time. This makes sense because we believe that whatever word combinations that occurred in the corpus are valid, so we should not exclude any potential candidate no matter how unlikely it is. This is the basic implementation, however at times we would like to vary the degree of our generative freedom. Can we generate more 'unlikely' sentences instead of the most commonly observed ones? We can by modifying the shape of the probability curve for the potential candidates. Study the implementation below, the additional steps introduce the temperature parameter that controls the probability curve. When the temperature is increased, the curve becomes flatter, when the temperature is decreased the curve becomes sharper. Still using 'numpy.choices' as the base selection mechanism, the flatter probability curve grants more chance to the unlikely candidates compared to the sharper curve.

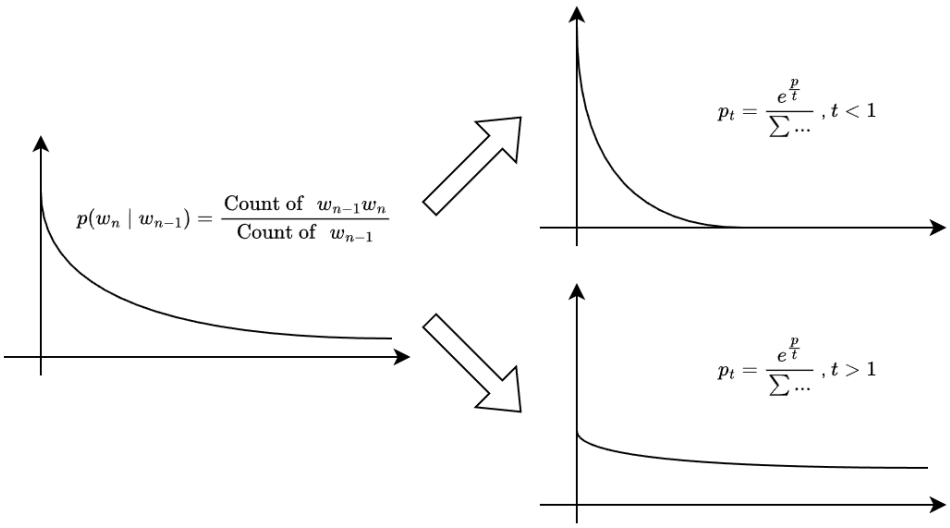

Figure 2.1: The temperature modify the probability distribution of the next word, if temperature is smaller than 1, the model will prefer the most likely word. If the temperature is larger than 1, others words are given more chance to be selected.

2.13 N-gram Scorer

An n-gram language model isn't just for predictions. It can act as a 'scorer' to evaluate the naturalness or likelihood of a given sentence based on trained data. A higher score means the sentence is more in line with the patterns seen in the training data.

For a given sentence, the model calculates the probability of each n-gram sequence and multiplies them to get the sentence's overall likelihood. Sentences with higher likelihoods are deemed more "natural" based on the training data. For example, using a bigram model trained on English novels, the sentence "The cat sat on the mat" will likely have a higher score than "Mat the on sat cat the."

Exercise 2.13.1

Rank Sentences Given a list of sentences, use your bigram model to rank them from most to least natural according to the training data.

Perturb & Score Start with a "natural" sentence. Modify or shuffle words to create variations. Score each variation. How do scores change as you alter the sentence?

Compare Corpora Train two bigram models on different corpora (e.g., news articles vs. poetry). Score the same set of sentences on both models. How do the scores differ? What does this reveal about the nature of the two corpora?

2.14 The Limitations of Bigrams

Imagine the following scenarios: A friend tells you, "I am so blue today." You're selecting paint colors and you say, "I want blue today." In both sentences, the bigram "blue today" appears. But does it convey the same

meaning in both? Certainly not! In the first, "blue" refers to feeling sad, while in the second, "blue" is literally about the color.

Exercise 2.14.1 Can you think of other examples where the same bigram can have different meanings in different contexts?

The OOV Problem

By now you should now that given the context word, the language model, GPT or bigram, will give you a probability distribution of your next words. But for bigram model, we have a particular limitation, i.e. you don't get probability for all the words, just a selected few that appeared in the corpus.

Exercise 2.14.2 What if the context is not found in the trigram model? Can the model still predict the next word?

The corpus determines the available words in our bigram model, if the word doesn't appear in the corpus, the model cannot provide any meaningful guess for the next word.

Forgetting

The more words we consider together (trigrams, 4-grams, and so on), the better we can capture context. A trigram involves three words. For example, in "I love ice cream," the trigrams are "I love ice" and "love ice cream." Compare: "I am feeling a bit under the weather." "Let's go under the weather balloon." The bigram "under the" is common to both, but the trigram "bit under the" versus "go under the" provides more context.

While it might seem like the solution is to always use larger "n-grams", there are challenges:

Data Sparsity As 'n' increases, the likelihood of encountering the exact n-gram in a dataset decreases. For a 10-gram, you might not find many matches, making it harder to generalize or predict.

Computational Complexity Storing and processing larger n-grams require more computational power and memory.

Overfitting If we rely on very large n-grams, our model may become too specific to our training data, reducing its ability to generalize to new data.

Exercise 2.14.3 Imagine you're using 20-grams to process a book. What potential challenges might arise?

Exercise 2.14.4 Which tasks should use a longer n-gram? Which should use a short n-gram?

Exercise 2.14.5 Given a sentence, what n-gram is not able to do? 1)extract important info like time and person. 2) infer the previous sentence.

Surface representation

One of the main problems with n-gram language models is the issue of surface representation. N-gram models mainly focus on the local context of words and are unable to capture the broader semantic or syntactic relationships between words in a sentence. This limitation arises because n-grams do not take into account the hierarchical structure of language.

For example, consider the sentence: “The cat sat on the mat.” In a trigram model, the probability of the word sequence “cat sat on” would be calculated based on the co-occurrence frequency of the trigram “cat sat on” in the training data. However, the trigram model fails to capture the fact that “cat” is the subject of the sentence and “sat” is the verb.

Let’s delve deeper into the semantic aspect of the language. What is a word, let’s say ‘cat’. Is it a string made up of 3 letters, ‘c’, ‘a’ and ‘t’? This is the surface representation. To someone who doesn’t understand English, a surface representation in English fails to communicate anything meaningful.

2.15 Other Applications

One of the first usage of n-gram model is in spell check and correction. Bigrams can help in identifying commonly misspelled words in the context of their adjacent words. For example, if someone types "I have a blu skye", the bigram "blu skye" can be flagged as uncommon, leading to a suggestion of "blue sky". However, if your spell checker is a bigram model, it will not flag out the rare instance of me owning the sky. Because the bigrams are common in the training corpus. Using n-gram models as scorers can help in numerous applications: Grammar Checkers: To suggest more natural-sounding corrections. Machine Translation Evaluation: To score the fluency of translated sentences.

2.16 Implications & Considerations

So in our generative or predictive model, bigrams can be used to predict the likelihood of a word following another. The generated or predicted sequence is 100% determined by the underlying corpus which provides the bigram probabilities.

The scores are relative to the training data. A high score only indicates that the sentence is similar to patterns in the training data, not necessarily that it’s "correct" or "fluent" in a broader sense. Over-reliance can make models blind to novel yet valid constructions not present in the training data.

Alphabetical Index