CS50's Introduction to Artificial Intelligence with Python
CS50 的人工智能入门与 Python

OpenCourseWare 开放课程

Donate (https://cs50.harvard.edu/donate)

Brian Yu (https://brianyu.me)
brian@cs.harvard.edu

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/) (https://www.reddit.com/user/davidjmalan) (https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Lecture 6 讲座 6

These notes reflect the new version of Lecture 6, released on 14 August 2023. If you watched the prior version of the lecture and wish to see its notes, **click here**.

这些笔记反映了 2023 年 8 月 14 日发布的第六讲的新版本。如果你观看了之前的讲座版本并希望查看其笔记，请点击此处。

Language 语言

So far in the course, we needed to shape tasks and data such that an AI will be able to process them. Today, we will look at how an AI can be constructed to process human language.

到目前为止，在课程中，我们需要将任务和数据塑造成形，以便 AI 能够处理它们。今天，我们将探讨如何构建 AI 来处理人类语言。

Natural Language Processing spans all tasks where the AI gets human language as input. The following are a few examples of such tasks:

自然语言处理涵盖了所有 AI 以人类语言为输入的任务。以下是一些此类任务的示例：

- automatic summarization, where the AI is given text as input and it produces a summary of the text as output.

  自动摘要，AI 接收文本作为输入，然后生成文本的摘要作为输出。
- information extraction, where the AI is given a corpus of text and the AI extracts data as output.

  信息提取，其中 AI 被给予一个文本语料库，AI 将数据作为输出提取。
- language identification, where the AI is given text and returns the language of the text as output.

  语言识别，其中 AI 被提供文本，然后将文本的语言作为输出返回。
- machine translation, where the AI is given a text in the origin language and it outputs the translation in the target language.

  机器翻译，其中 AI 被给予原始语言的文本，并输出目标语言的翻译。
- named entity recognition, where the AI is given text and it extracts the names of the entities in the text (for example, names of companies).

  命名实体识别，其中 AI 被给予文本，然后从中提取文本中的实体名称（例如，公司的名称）。
- speech recognition, where the AI is given speech and it produces the same words in text.

  语音识别，其中 AI 接收语音并以文本形式输出相同的单词。
- text classification, where the AI is given text and it needs to classify it as some type of text.

  文本分类，其中 AI 被给予文本，需要将其分类为某种类型的文本。
- word sense disambiguation, where the AI needs to choose the right meaning of a word that has multiple meanings (e.g. bank means both a financial institution and the ground on the sides of a river).

词义消歧，其中 AI 需要选择一个词的正确含义，该词有多个含义（例如，银行既可以指金融机构，也可以指河岸的地面）。

Syntax and Semantics 语法与语义

Syntax is sentence structure. As native speakers of some human language, we don't struggle with producing grammatical sentences and flagging non-grammatical sentences as wrong. For example, the sentence "Just before nine o'clock Sherlock Holmes stepped briskly into the room" is grammatical, whereas the sentence "Just before Sherlock Holmes nine o'clock stepped briskly the room" is non-grammatical. Syntax can be grammatical and ambiguous at the same time, as in "I saw the man with the telescope." Did I see (the man with the telescope) or did I see (the man), doing so by looking through the telescope? To be able to parse human speech and produce it, the AI needs to command syntax.

语法是句子结构。作为某些人类语言的母语使用者，我们不需要努力生成语法正确的句子，也不认为非语法正确的句子是错误的。例如，"九点前，夏洛克·福尔摩斯大步走进了房间"是一个语法正确的句子，而"九点前，夏洛克·福尔摩斯走进了房间的步履匆匆"则是一个非语法正确的句子。语法可以同时是正确的和模棱两可的，例如，"我看到了用望远镜看的人。"我是看到了（用望远镜看的人）还是看到了（那个人），通过望远镜观察？为了能够解析人类的言语并产生这种言语，AI 需要掌握语法。

Semantics is the meaning of words or sentences. While the sentence "Just before nine o'clock Sherlock Holmes stepped briskly into the room" is syntactically different from "Sherlock Holmes stepped briskly into the room just before nine o'clock," their content is effectively identical. Similarly, although the sentence "A few minutes before nine, Sherlock Holmes walked quickly into the room" uses different words from the previous sentences, it still carries a very similar meaning. Moreover, a sentence can be perfectly grammatical while being completely nonsensical, as in Chomsky's example, "Colorless green ideas sleep furiously." To be able to parse human speech and produce it, the AI needs to command semantics.

语义是词语或句子的意义。虽然句子"九点前几分钟，夏洛克·福尔摩斯大步走进了房间"在语法上与"夏洛克·福尔摩斯大步走进房间，就在九点前几分钟"不同，但它们的内容实质上是相同的。同样，尽管"九点前几分钟，夏洛克·福尔摩斯快步走进了房间"使用了与前几个句子不同的词汇，但它仍然承载着非常相似的意义。此外，一个句子可以完全符合语法规则，但毫无意义，就像乔姆斯基的例子"无色的绿色想法狂睡"那样。为了能够解析人类的言语并产生这种言语，AI 需要掌握语义。

Context-Free Grammar 上下文无关文法

Formal Grammar is a system of rules for generating sentences in a language. In **Context-Free Grammar**, the text is abstracted from its meaning to represent the structure of the sentence using formal grammar. Let's consider the following example sentence:

正式语法是一种规则系统，用于生成语言中的句子。在无约束语法中，文本从其意义中抽象出来，使用正式语法表示句子的结构。让我们考虑以下示例句子：

- She saw the city.

  她看到了城市。

This is a simple grammatical sentence, and we would like to generate a syntax tree representing its structure.

这是一个简单的语法句子，我们希望生成一个语法树来表示其结构。

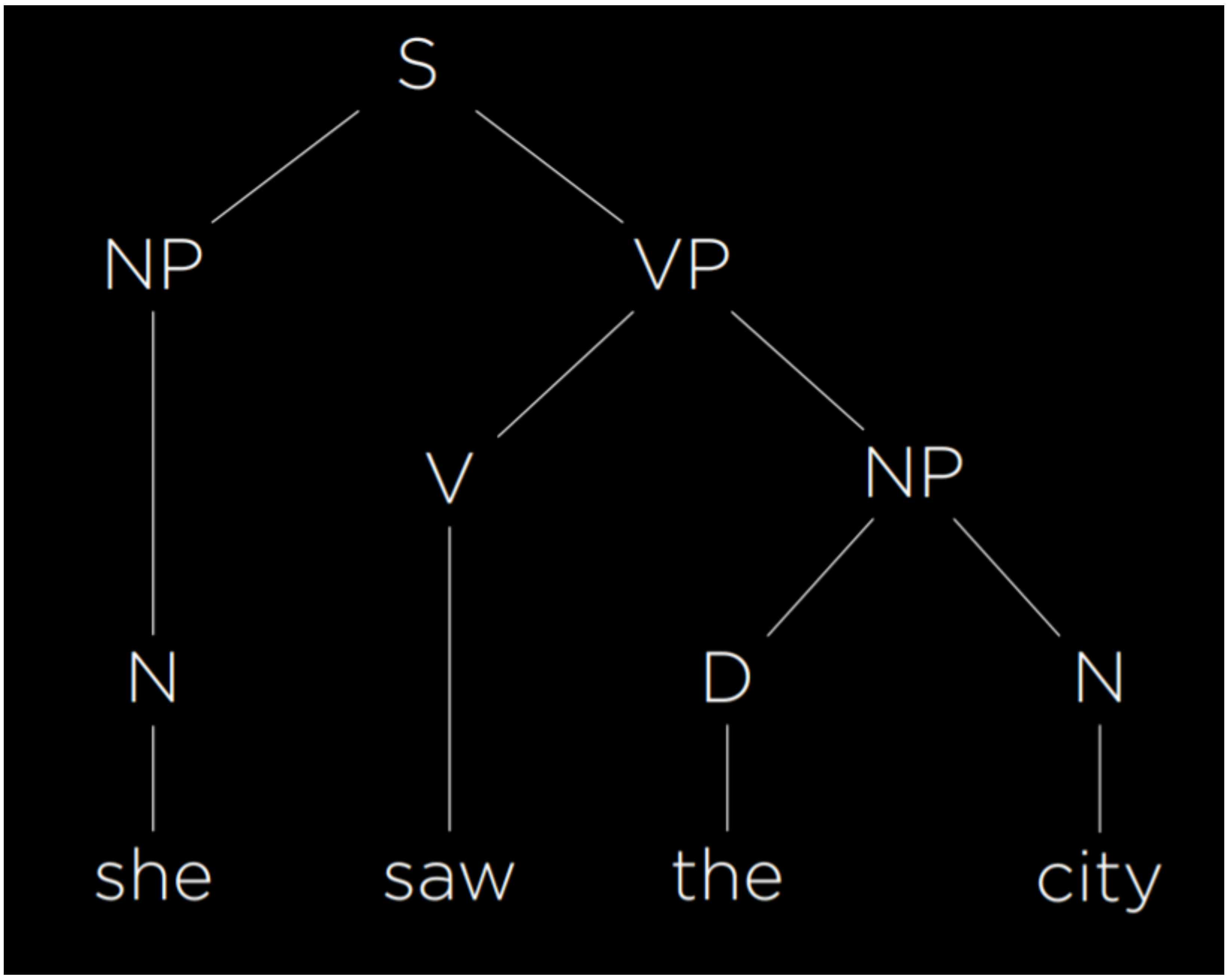
We start by assigning each word its part of speech. *She* and *city* are nouns, which we will mark as N. *Saw* is a verb, which we will mark as V. *The* is a determiner, marking the following noun as definite or indefinite, and we will mark it as D. Now, the above sentence can be rewritten as

我们首先为每个单词分配其词性。她和城市是名词，我们将标记为 N。看到是动词，我们将标记为 V。the 是一个限定词，标记后续的名词为限定或不定，我们将标记为 D。现在，上述句子可以重写为

- N V D N

So far, we have abstracted each word from its semantic meaning to its part of speech. However, words in a sentence are connected to each other, and to understand the sentence we must understand how they connect. A noun phrase (NP) is a group of words that connect to a noun. For example, the word *she* is a noun phrase in this sentence. In addition, the words *the city* also form a noun phrase, consisting of a determiner and a noun. A verb phrase (VP) is a group of words that connect to a verb. The word *saw* is a verb phrase in itself. However, the words *saw the city* also make a verb phrase. In this case, it is a verb phrase consisting of a verb and a noun phrase, which in turn consists of a determiner and a noun. Finally, the whole sentence (S) can be represented as follows:

到目前为止，我们已经从语义含义抽象到词性。然而，句子中的词之间是相互关联的，要理解句子，我们必须理解它们之间的联系。名词短语（NP）是一组与名词相关的词。例如，这个句子中的词"她"就是一个名词短语。此外，"城市"这两个词也构成一个名词短语，由限定词和名词组成。动词短语（VP）是一组与动词相关的词。单词"看到"本身就是一个动词短语。然而，"看到城市"这两个词也构成一个动词短语。在这种情况下，它是一个由动词和名词短语组成的动词短语，而名词短语又由限定词和名词组成。最后，整个句子（S）可以表示如下：



Using formal grammar, the AI is able to represent the structure of sentences. In the grammar we have described, there are enough rules to represent the simple sentence above. To represent more complex sentences, we will have to add more rules to our formal grammar.

使用正式语法，AI 能够表示句子的结构。在我们所描述的语法中，有足够的规则来表示上述简单句子。为了表示更复杂的句子，我们将需要在正式语法中添加更多的规则。

nltk

As is often the case in Python, multiple libraries have been written to implement the idea above. nltk (Natural Language Toolkit) is one such library. To analyze the sentence from above, we will provide the algorithm with rules for the grammar:

正如 Python 中经常发生的那样，已经编写了多个库来实现上述想法。nltk（自然语言工具包）就是这样一个库。为了分析上面的句子，我们将为算法提供语法规则：

```
import nltk

grammar = nltk.CFG.fromstring("""
    S -> NP VP

    NP -> D N | N
    VP -> V | V NP

    D -> "the" | "a"
    N -> "she" | "city" | "car"
    V -> "saw" | "walked"
""")

parser = nltk.ChartParser(grammar)
```

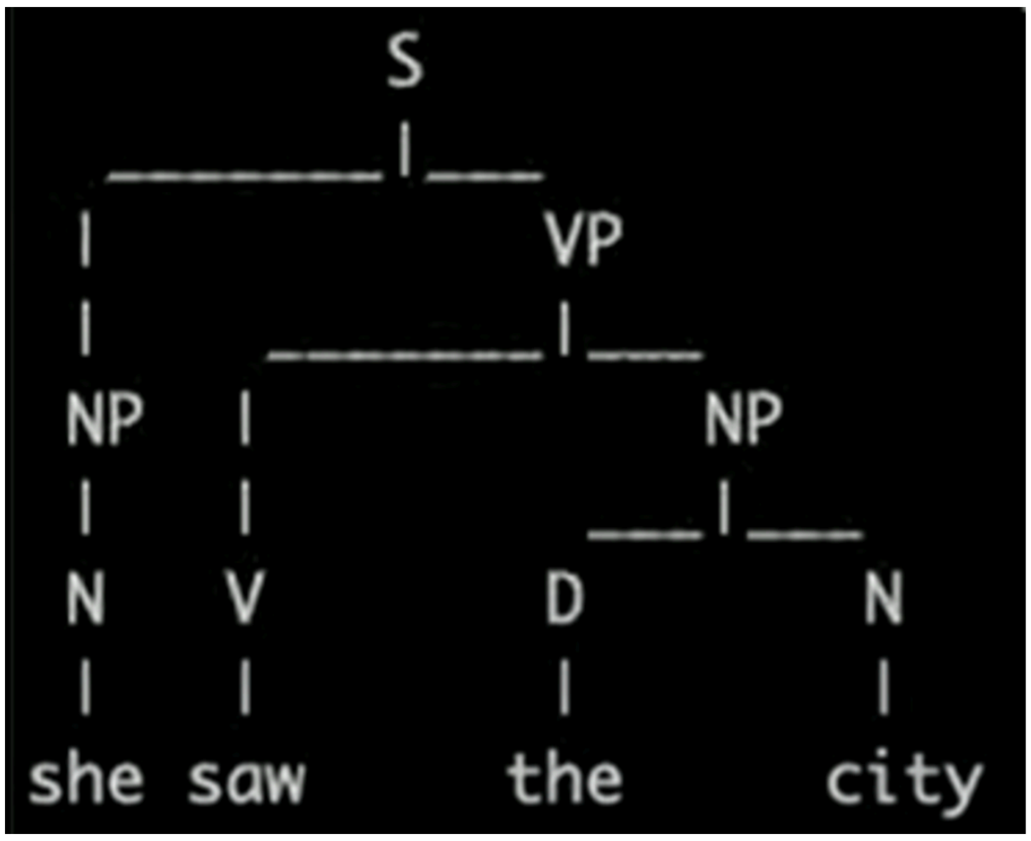
Similar to what we did above, we define what possible components could be included in others. A sentence can include a noun phrase and a verb phrase, while the phrases themselves can consist of other phrases, nouns, verbs, etc., and, finally, each part of speech spans some words in the language.

类似于我们上面所做的，我们定义了可能包含在其他组件中的可能组件。一个句子可以包含一个名词短语和一个动词短语，而这些短语本身可以由其他短语、名词、动词等组成，最后，每个词性在语言中覆盖一些单词。

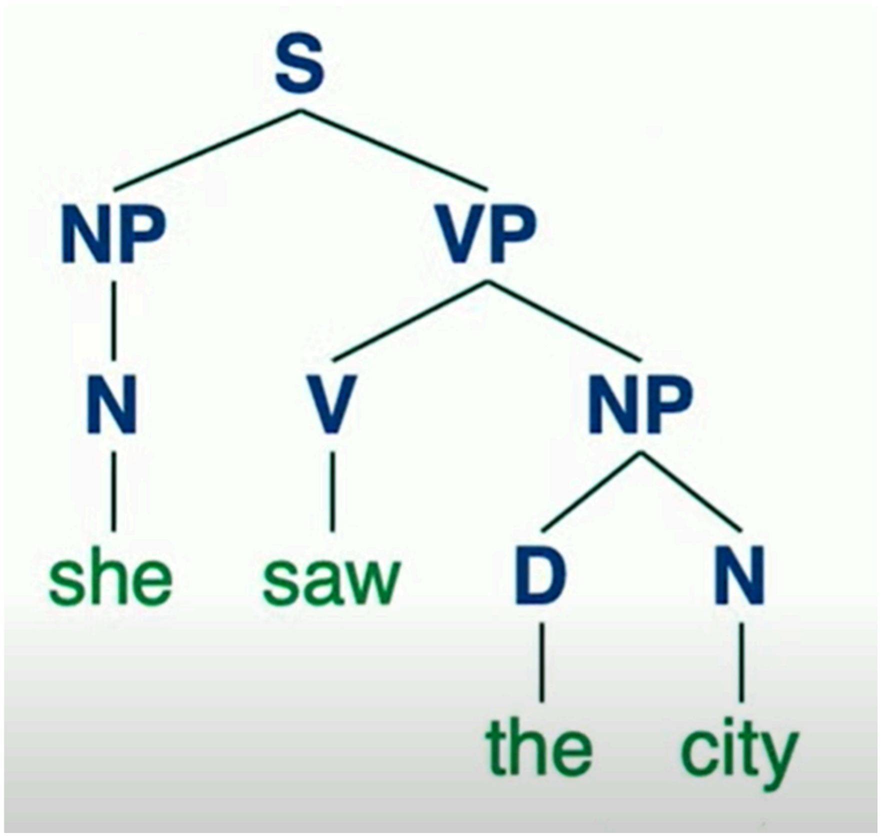
```
sentence = input("Sentence: ").split()
try:
    for tree in parser.parse(sentence):
        tree.pretty_print()
        tree.draw()
except ValueError:
    print("No parse tree possible.")
```

After giving the algorithm an input sentence split into a list of words, the function prints the resulting syntactic tree (pretty_print) and also generates a graphic representation (draw).

给算法提供一个被拆分为单词列表的输入句子后，函数打印出结果的句法树（pretty_print）并生成图形表示（draw）。



```
             S
     ________|___
    |            VP
    |    ________|___
    NP  |            NP
    |   |         ___|___
    N   V        D       N
    |   |        |       |
   she saw      the     city
```



n-grams n-grams

An *n*-gram is a sequence of *n* items from a sample of text. In a **character *n*-gram**, the items are characters, and in a **word *n*-gram** the items are words. A *unigram*, *bigram*, and *trigram* are sequences of one, two, and three items. In the following sentence, the first three *n*-grams are "how often have," "often have I," and "have I said."

n-元组是从文本样本中获取的 n 个项目的序列。在字符 n-元组中，项目是字符，而在词 n-元组中，项目是词。单元组、双元组和三元组分别是包含一个、两个和三个项目的序列。在以下句子中，前三个 n-元组是"how often have"、"often have I"和"have I said"。

"How often have I said to you that when you have eliminated the impossible whatever remains, however improbable, must be the truth?"

"我曾对你们说过多少次，当你排除了所有不可能的情况后，剩下的无论多么不可能，都必须是真相。"

n-grams are useful for text processing. While the AI hasn't necessarily seen the whole sentence before, it sure has seen parts of it, like "have I said." Since some words occur together more often than others, it is possible to also predict the next word with some probability. For example, your smartphone suggests words to you based on a probability distribution derived from the last few words you typed. Thus, a helpful step in natural language processing is breaking the sentence into n-grams.

n-grams 对文本处理很有用。虽然 AI 并不一定见过整句，但它肯定见过其中的部分，比如"我说过"。由于一些词比其他词更常一起出现，因此有可能根据一定的概率预测下一个词。例如，你的智能手机会根据你最近输入的几个词的概率分布来为你建议单词。因此，在自然语言处理中，将句子分解为 n-grams 是一个有帮助的步骤。

Tokenization 分词

Tokenization is the task of splitting a sequence of characters into pieces (tokens). Tokens can be words as well as sentences, in which case the task is called **word tokenization** or **sentence tokenization**. We need tokenization to be able to look at *n*-grams, since those rely on sequences of tokens. We start by splitting the text into words based on the space character. While this is a good start, this method is imperfect because we end up with words with punctuation, such as "remains,". So, for example, we can remove punctuation. However, then we face additional challenges, such as words with apostrophes (e.g. "o'clock") and hyphens (e.g. "pearl-grey). Additionally, some punctuation is important for sentence structure, like periods. However, we need to be able to tell apart between a period at the end of the word "Mr." and a period in the end of the sentence. Dealing with these questions is the process of tokenization. In the end, once we have our tokens, we can start looking at *n*-

grams.

分词是将字符序列分割成片段（称为令牌）的任务。令牌可以是单词，也可以是句子，这时任务称为词分词或句分词。我们需要分词以便查看 n-grams，因为这些依赖于令牌序列。我们首先根据空格字符将文本分割成单词。虽然这是一个良好的开始，但这种方法并不完美，因为我们最终会得到包含标点符号的单词，如“remains”。因此，例如，我们可以移除标点符号。然而，这会带来额外的挑战，例如包含撇号的单词（例如“o'clock”）和破折号（例如“pearl-grey”）。此外，一些标点符号对于句子结构很重要，比如句点。然而，我们需要能够区分单词“Mr.”末尾的句点和句子末尾的句点。处理这些问题就是分词的过程。最终，一旦我们有了令牌，我们就可以开始查看 n-grams。

Markov Models 马尔可夫模型

As discussed in previous lectures, Markov models consist of nodes, the value of each of which has a probability distribution based on a finite number of previous nodes. Markov models can be used to generate text. To do so, we train the model on a text, and then establish probabilities for every *n*-th token in an *n*-gram based on the *n* words preceding it. For example, using trigrams, after the Markov model has two words, it can choose a third one from a probability distribution based on the first two. Then, it can choose a fourth word from a probability distribution based on the second and third words. To see an implementation of such a model using nltk, refer to generator.py in the source code, where our model learns to generate Shakespeare-sounding sentences. Eventually, using Markov models, we are able to generate text that is often grammatical and sounding superficially similar to human language output. However, these sentences lack actual meaning and purpose.

如在之前的讲座中讨论的那样，马尔可夫模型由节点组成，每个节点的值基于有限数量的先前节点具有概率分布。马尔可夫模型可以用于生成文本。为此，我们通过训练模型来使用文本，然后根据前 n 个单词建立 n-gram 中每个第 n 个令牌的概率。例如，使用三元组，当马尔可夫模型有两个单词时，它可以基于前两个单词从概率分布中选择第三个单词。然后，它可以基于第二和第三个单词从概率分布中选择第四个单词。要查看使用 nltk 实现此类模型的示例，请参阅源代码中的 generator.py，其中我们的模型学习生成听起来像莎士比亚风格的句子。最终，使用马尔可夫模型，我们能够生成语法正确且听起来与人类语言输出相似的文本。然而，这些句子缺乏实际意义和目的。

Bag-of-Words Model 词袋模型

Bag-of-words is a model that represents text as an unordered collection of words. This model ignores syntax and considers only the meanings of the words in the sentence. This approach is helpful in some classification tasks, such as sentiment analysis (another classification task would be distinguishing regular email from spam email). Sentiment analysis can be used, for instance, in product reviews, categorizing reviews as positive or negative. Consider the following sentences:

词袋模型是一种将文本表示为无序单词集合的模型。此模型忽略了语法，仅考虑句子中单词的意义。这种方法在某些分类任务中很有帮助，例如情感分析（另一个分类任务是区分普通邮件与垃圾邮件）。在产品评论中，可以使用情感分析，将评论分类为正面或负面。考虑以下句子：

1. “My grandson loved it! So much fun!”

   "我的孙子喜欢它！太好玩了！"

2. “Product broke after a few days.”

   产品在几天后就坏了。

3. “One of the best games I've played in a long time.”

   “这是我很久以来玩过的最好的游戏之一。”

4. “Kind of cheap and flimsy, not worth it.”

   "有点廉价和轻薄，不值得。"

Based only on the words in each sentence and ignoring the grammar, we can see that sentences 1 and 3 are positive (“loved,” “fun,” “best”) and sentences 2 and 4 are negative (“broke,” “cheap,” “flimsy”).

仅根据每句话中的单词，不考虑语法，我们可以看出第 1 句和第 3 句是积极的（“被爱”，“有趣”，“最好”），而第 2 句和第 4 句是消极的（“破裂”，“便宜”，“轻薄”）。

Naive Bayes 朴素贝叶斯

Naive Bayes is a technique that can be used in sentiment analysis with the bag-of-words model. In sentiment analysis, we are asking “What is the probability that the sentence is positive/negative given the words in the sentence.” Answering this question requires computing conditional probability, and it is helpful to recall Bayes' rule from lecture 2:

朴素贝叶斯是一种可以在词袋模型中用于情感分析的技术。在情感分析中，我们询问的是“给定句子中的单词，句子是正面/负面的概率是多少。”回答这个问题需要计算条件概率，并且复习第 2 讲中的贝叶斯法则会很有帮助：

$$P(b \mid a) = \frac{P(b)\ P(a \mid b)}{P(a)}$$

Now, we would like to use this formula to find P(sentiment | text), or, for example, P(positive | “my grandson loved it”). We start by tokenizing the input, such that we end up with P(positive | “my”, “grandson”, “loved”, “it”). Applying Bayes’ ruled directly, we get the following expression: P(“my”, “grandson”, “loved”, “it” | positive)*P(positive)/P(“my”, “grandson”, “loved”, “it”). This complicated expression will give us the precise answer to P(positive | “my”, “grandson”, “loved”, “it”).

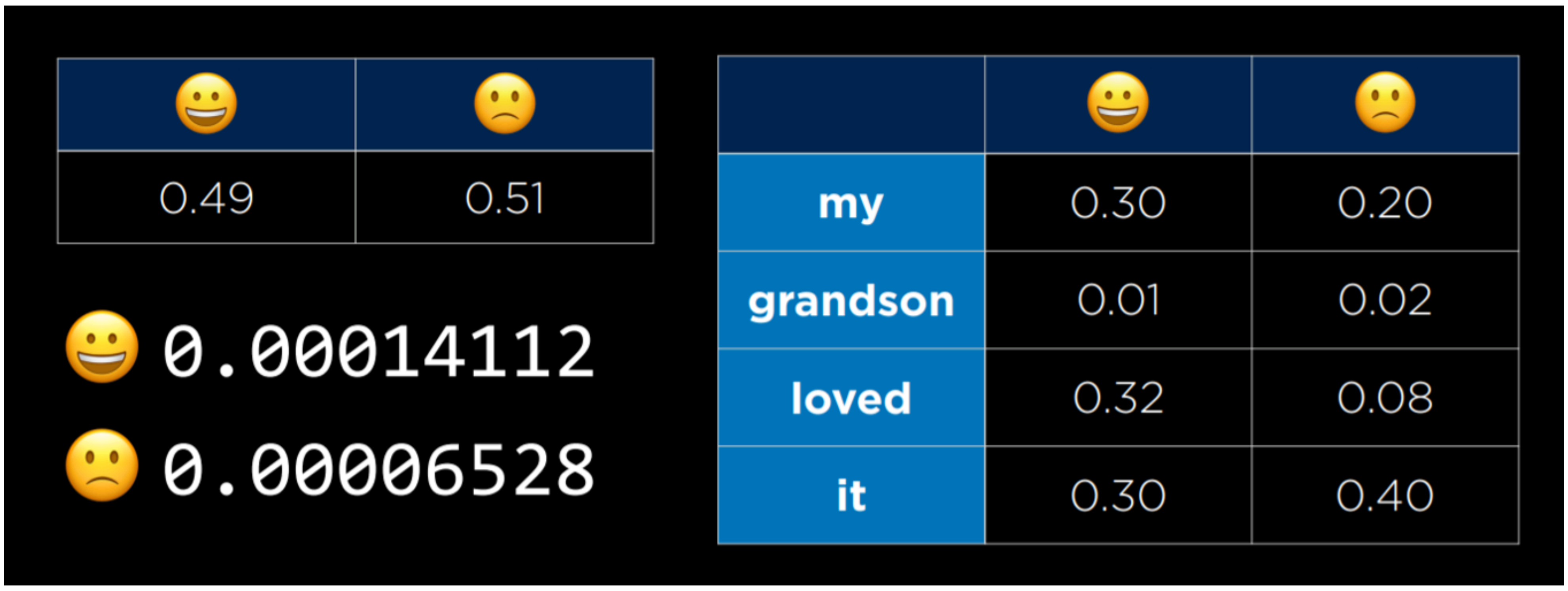
现在，我们想要使用这个公式来找到 P(sentiment | text)，例如，P(positive | “my grandson loved it”）。我们首先对输入进行分词，使得我们得到 P(positive | “my”，“grandson”，“loved”，“it”）。直接应用贝叶斯法则，我们得到以下表达式：P(“my”，“grandson”，“loved”，“it” | positive)*P(positive)/P(“my”，“grandson”，“loved”，“it”）。这个复杂的表达式将给出 P(positive | “my”，“grandson”，“loved”，“it”）的确切答案。

However, we can simplify the expression if we are willing to get an answer that’s not equal, but proportional to P(positive | “my”, “grandson”, “loved”, “it”). Later on, knowing that the probability distribution needs to sum up to 1, we can normalize the resulting value into an exact probability. This means that we can simplify the expression above to the numerator only: P(“my”, “grandson”, “loved”, “it” | positive)*P(positive). Again, we can simplify this expression based on the knowledge that a conditional probability of *a* given *b* is proportional to the joint probability of *a* and *b*. Thus, we get the following expression for our probability: P(positive, “my”, “grandson”, “loved”, “it”)*P(positive). Calculating this joint probability, however, is complicated, because the probability of each word is conditioned on the probabilities of the words preceding it. It requires us to compute P(positive)*P(“my” | positive)*P(“grandson” | positive, “my”)*P(loved | positive, “my”, “grandson”)*P(“it” | positive, “my”, “grandson”, “loved”).

然而，如果我们愿意接受一个不等于 P(positive | “my”, “grandson”, “loved”, “it”)但与其成比例的答案，我们可以简化表达式。稍后，当我们知道概率分布需要总和为 1 时，我们可以将得到的值归一化为一个精确的概率。这意味着我们可以将上述表达式简化为分子：P(“my”, “grandson”, “loved”, “it” | positive)*P(positive)。再次，基于给定 b 的条件概率与 a 和 b 的联合概率成比例的知识，我们可以将此表达式简化为：P(positive, “my”, “grandson”, “loved”, “it”)*P(positive)。然而，计算这个联合概率是复杂的，因为每个单词的概率都依赖于其前一个单词的概率。这需要我们计算 P(positive)*P(“my” | positive)*P(“grandson” | positive, “my”)*P(loved | positive, “my”, “grandson”)*P(“it” | positive, “my”, “grandson”, “loved”)。

Here is where we use Bayes’ rules naively: we assume that the probability of each word is independent from other words. This is not true, but despite this imprecision, Naive Bayes’ produces a good sentiment estimate. Using this assumption, we end up with the following probability: P(positive)*P(“my” | positive)*P(“grandson” | positive)*P(“loved” | positive)*P(“it” | positive), which is not that difficult to calculate. P(positive) = the number of all positive samples divided by the number of total samples. P(“loved” | positive) is equal to the number of positive samples with the word “loved” divided by the number of positive samples. Let’s consider the example below, with smiling and frowning emojies substituting the words “positive” and “negative”:

在这里，我们粗略地使用了贝叶斯规则：我们假设每个单词的概率与其他单词独立。这并不正确，但尽管如此，朴素贝叶斯仍然能产生一个不错的情感估计。基于这个假设，我们得到以下概率：P(正面)*P("我" | 正面)*P("孙子" | 正面)*P("爱" | 正面)*P("它" | 正面)，这并不难计算。P(正面) = 所有正面样本的数量除以总样本数量。P("爱" | 正面)等于包含单词“爱”的正面样本数量除以正面样本总数。让我们考虑以下例子，用微笑和皱眉的表情符号代替“正面”和“负面”： 正面样本数量 = 100 负面样本数量 = 100 包含“爱”的正面样本数量 = 50 包含“孙子”的正面样本数量 = 30 包含“它”的正面样本数量 = 40 P(正面) = 100 / (100 + 100) = 0.5 P("爱" | 正面) = 50 / 100 = 0.5 P("孙子" | 正面) = 30 / 100 = 0.3 P("它" | 正面) = 40 / 100 = 0.4 因此，P(正面)*P("爱" | 正面)*P("孙子" | 正面)*P("它" | 正面) = 0.5 * 0.5 * 0.3 * 0.4 = 0.03



😀	🙁
0.49	0.51

😀 0.00014112

🙁 0.00006528

	😀	🙁
my	0.30	0.20
grandson	0.01	0.02
loved	0.32	0.08
it	0.30	0.40

On the right we are seeing a table with the conditional probabilities of each word on the left occurring in a sentence given that the sentence is positive or negative. In the small table on the left we are seeing the probability of a positive or a negative sentence. On the bottom left we are seeing the resulting probabilities following the computation. At this point, they are in proportion to each other, but they don't tell us much in terms of probabilities. To get the probabilities, we need to normalize the values, arriving at P(positive) = 0.6837 and P(negative) = 0.3163. The strength of naive Bayes is that it is sensitive to words that occur more often in one type of sentence than in the other. In our case, the word "loved" occurs much more often in positive sentences, which makes the whole sentence more likely to be positive than negative. To see an implementation of sentiment assessment using Naive Bayes with the nltk library, refer to sentiment.py.

在右侧，我们看到一个表格，其中列出了左侧每个单词在句子为正面或负面情况下发生的条件概率。在左侧的小表格中，我们看到正面或负面句子的概率。在左下角，我们看到计算后的结果概率。此时，它们彼此成比例，但就概率而言，它们并没有告诉我们太多信息。为了得到概率，我们需要对值进行归一化，得到 P(正面) = 0.6837 和 P(负面) = 0.3163。朴素贝叶斯的强大之处在于它对在一种类型的句子中比另一种类型句子中出现更频繁的单词敏感。在我们的情况下，"loved"这个词在正面句子中出现的频率要高得多，这使得整个句子更有可能是正面而不是负面。要查看使用 nltk 库实现情感评估的朴素贝叶斯实现，请参阅 sentiment.py。

One problem that we can run into is that some words may never appear in a certain type of sentence. Suppose none of the positive sentences in our sample had the word "grandson." Then, P("grandson" | positive) = 0, and when computing the probability of the sentence being positive we will get 0. However, this is not the case in reality (not all sentences mentioning grandsons are negative). One way to go about this problem is with **Additive Smoothing**, where we add a value α to each value in our distribution to smooth the data. This way, even if a certain value is 0, by adding α to it we won't be multiplying the whole probability for a positive or negative sentence by 0. A specific type of additive smoothing, **Laplace Smoothing** adds 1 to each value in our distribution, pretending that all values have been observed at least once.

我们可能会遇到的一个问题是，某些词可能在某种类型的句子中从未出现。假设我们样本中的所有正例句子都没有使用"孙子"这个词。那么，P("孙子" | 正例) = 0，当我们计算句子为正例的概率时，我们会得到 0。然而，在实际情况中并非如此（并非所有提到孙子的句子都是负面的）。解决这个问题的一种方法是加性平滑，我们向分布中的每个值添加一个值α，以平滑数据。这样，即使某个值为 0，通过向它添加α，我们不会将整个正例或负例句子的概率乘以 0。加性平滑的一种特定类型，拉普拉斯平滑，将 1 添加到我们分布中的每个值，假装所有值至少被观察过一次。

Word Representation 词表示

We want to represent word meanings in our AI. As we've seen before, it is convenient to provide input to the AI in the form of numbers. One way to go about this is by using **One-Hot Representation**, where each word is represented with a vector that consists of as many values as we have words. Except for a single value in the vector that is equal to 1, all other values are equal to 0. How we can differentiate words is by which of the values is 1, ending up with a unique vector per word. For example, the sentence "He wrote a book" can be represented as four vectors:

我们希望在我们的 AI 中表示单词含义。正如我们之前所见，以数字形式向 AI 提供输入是方便的。一种方法是使用 One-Hot 表示法，其中每个单词用一个向量表示，该向量包含我们拥有的单词数量。除了向量中的一个值等于 1 外，所有其他值都等于 0。通过确定哪个值为 1，我们可以区分单词，最终得到每个单词的独特向量。例如，"他写了一本书"可以用四个向量表示： 1. 他: [1, 0, 0, ...] 2. 写: [0, 1, 0, ...] 3. 一: [0, 0, 1, ...] 4. 本: [0, 0, 0, ...]

- [1, 0, 0, 0] (he)

  [1, 0, 0, 0] (他)
- [0, 1, 0, 0] (wrote)

  [0, 1, 0, 0]（写）
- [0, 0, 1, 0] (a)

- [0, 0, 0, 1] (book)

  [0, 0, 0, 1]（书）

However, while this representation works in a world with four words, if we want to represent words from a dictionary, when we can have 50,000 words, we will end up with 50,000 vectors of length 50,000. This is incredibly inefficient. Another problem in this kind of representation is that we are unable to represent similarity between words like “wrote” and “authored.” Instead, we turn to the idea of **Distributed Representation**, where meaning is distributed across multiple values in a vector. With distributed representation, each vector has a limited number of values (much less than 50,000), taking the following form:

然而，在一个只有四个词的世界里，这种表示方式可以工作，但如果我们要表示字典中的词，当我们可能有 5 万个词时，我们将得到 5 万个长度为 5 万个向量。这极其低效。这种表示方式的另一个问题是，我们无法表示“wrote”和“authored”等词之间的相似性。相反，我们转向分布式表示的概念，其中意义分布在向量中的多个值上。在分布式表示中，每个向量的值数量有限（远少于 5 万个），形式如下：

- [-0.34, -0.08, 0.02, -0.18, …] (he)

  [-0.34, -0.08, 0.02, -0.18, …] (他)
- [-0.27, 0.40, 0.00, -0.65, …] (wrote)

  [-0.27, 0.40, 0.00, -0.65, …]（写）
- [-0.12, -0.25, 0.29, -0.09, …] (a)
- [-0.23, -0.16, -0.05, -0.57, …] (book)

  [-0.23, -0.16, -0.05, -0.57, …]（书）

This allows us to generate unique values for each word while using smaller vectors. Additionally, now we are able to represent similarity between words by how different the values in their vectors are.

这使我们能够在使用较小的向量的同时为每个单词生成唯一的值。此外，我们现在可以通过单词向量中的值有多不同来表示单词之间的相似性。

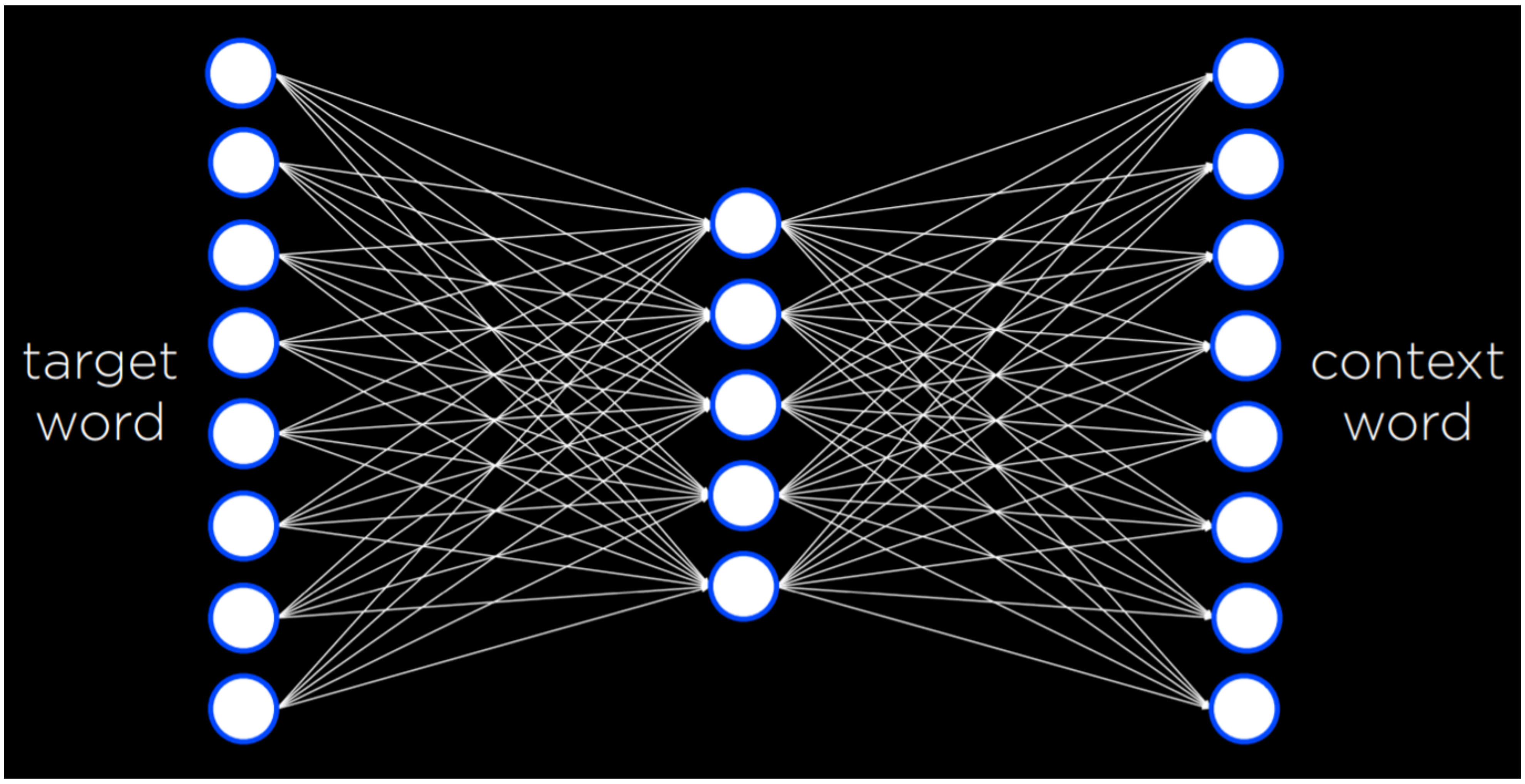
“You shall know a word by the company it keeps” is an idea by J. R. Firth, an English linguist. Following this idea, we can come to define words by their adjacent words. For example, there are limited words that we can use to complete the sentence “for ___ he ate.” These words are probably words like “breakfast,” “lunch,” and “dinner.” This brings us to the conclusion that by considering the environment in which a certain word tends to appear, we can infer the meaning of the word.

“你通过它所与之相伴的词就能知道一个词的意思”是英国语言学家 J. R. Firth 提出的一个观点。遵循这一观点，我们可以根据一个词的相邻词来定义这个词。例如，在完成句子“为了___他吃了。”时，我们能使用的词可能只有“早餐”、“午餐”和“晚餐”等。这让我们得出结论，通过考虑某个词倾向于出现的环境，我们可以推断出这个词的意思。

word2vec

word2vec is an algorithm for generating distributed representations of words. It does so by **Skip-Gram Architecture**, which is a neural network architecture for predicting context given a target word. In this architecture, the neural network has an input unit for every target word. A smaller, single hidden layer (e.g. 50 or 100 units, though this number is flexible) will generate values that represent the distributed representations of words. Every unit in this hidden layer is connected to every unit in the input layer. The output layer will generate words that are likely to appear in a similar context as the target words. Similar to what we saw in last lecture, this network needs to be trained with a training dataset using the backpropagation algorithm.

word2vec 是一种生成词的分布式表示的算法。它通过跳过词架构来实现，这是一种用于给定目标词预测上下文的神经网络架构。在这一架构中，神经网络为每个目标词有一个输入单元。一个较小的单隐藏层（例如，50 或 100 个单元，但这个数字是灵活的）将生成表示词的分布式表示的值。这个隐藏层中的每个单元都与输入层中的每个单元相连。输出层将生成可能出现在与目标词相似上下文中的词。类似于上一节课我们看到的，这个网络需要使用反向传播算法通过训练数据集进行训练。



This neural network turns out to be quite powerful. At the end of the process, every word ends up being just a vector, or a sequence of numbers. For example,

这个神经网络实际上非常强大。在过程结束时，每个词最终都变成了一个向量，或者说是一系列数字。例如，

book: [-0.226776 -0.155999 -0.048995 -0.569774 0.053220 0.124401 -0.091108 -0.606255 -0.114630 0.473384 0.061061 0.551323 -0.245151 -0.014248 -0.210003 0.316162 0.340426 0.232053 0.386477 -0.025104 -0.024492 0.342590 0.205586 -0.554390 -0.037832 -0.212766 -0.048781 -0.088652 0.042722 0.000270 0.356324 0.212374 -0.188433 0.196112 -0.223294 -0.014591 0.067874 -0.448922 -0.290960 -0.036474 -0.148416 0.448422 0.016454 0.071613 -0.078306 0.035400 0.330418 0.293890 0.202701 0.555509 0.447660 -0.361554 -0.266283 -0.134947 0.105315 0.131263 0.548085 -0.195238 0.062958 -0.011117 -0.226676 0.050336 -0.295650 -0.201271 0.014450 0.026845 0.403077 -0.221277 -0.236224 0.213415 -0.163396 -0.218948 -0.242459 -0.346984 0.282615 0.014165 -0.342011 0.370489 -0.372362 0.102479 0.547047 0.020831 -0.202521 -0.180814 0.035923 -0.296322 -0.062603 0.232734 0.191323 0.251916 0.150993 -0.024009 0.129037 -0.033097 0.029713 0.125488 -0.018356 -0.226277 0.437586 0.004913]

书籍：[-0.226776 -0.155999 -0.048995 -0.569774 0.053220 0.124401 -0.091108 -0.606255 -0.114630 0.473384 0.061061 0.551323 -0.245151 -0.014248 -0.210003 0.316162 0.340426 0.232053 0.386477 -0.025104 -0.024492 0.342590 0.205586 -0.554390 -0.037832 -0.212766 -0.048781 -0.088652 0.042722 0.000270 0.356324 0.212374 -0.188433 0.196112 -0.223294 -0.014591 0.067874 -0.448922 -0.290960 -0.036474 -0.148416 0.448422 0.016454 0.071613 -0.078306 0.035400 0.330418 0.293890 0.202701 0.555509 0.447660 -0.361554 -0.266283 -0.134947 0.105315 0.131263 0.548085 -0.195238 0.062958 -0.011117 -0.226676 0.050336 -0.295650 -0.201271 0.014450 0.026845 0.403077 -0.221277 -0.236224 0.213415 -0.163396 -0.218948 -0.242459 -0.346984 0.282615 0.014165 -0.342011 0.370489 -0.372362 0.102479 0.547047 0.020831 -0.202521 -0.180814 0.035923 -0.296322 -0.062603 0.232734 0.191323 0.251916 0.150993 -0.024009 0.129037 -0.033097 0.029713 0.125488 -0.018356 -0.226277 0.437586 0.004913]

By themselves, these numbers don't mean much. But by finding which other words in the corpus have the most similar vectors, we can run a function that will generate the words that are the most similar to the word *book*. In the case of this network it will be: book, books, essay, memoir, essays, novella, anthology, blurb, autobiography, audiobook. This is not bad for a computer! Through a bunch of numbers that don't carry any specific meaning themselves, the AI is able to generate words that really are very similar to *book* not in letters or sounds, but in meaning! We can also compute the difference between words based on how different their vectors are. For example, the difference between *king* and *man* is similar to the difference between *queen* and *woman*. That is, if we add the difference between *king* and *man* to the vector for *woman*, the closest word to the resulting vector is *queen*! Similarly, if we add the difference between *ramen* and *japan* to *america*, we get *burritos*. By using neural networks and distributed representations for words, we get our AI to understand semantic similarities between words in the language, bringing us one step closer to AIs that can understand and produce human language.

这些数字本身意义不大。但是，通过找到语料库中与这些词向量最相似的其他单词，我们可以运行一个函数，生成与单词"书"最相似的单词。在这个网络中，它将是：书，书籍，论文，回忆录，论文，短篇小说，选集，简介，自传，有声读物。对于计算机来说，这已经相当不错了！通过一堆本身没有特定意义的数字，AI 能够生成与"书"在意义上非常相似的单词，而不仅仅是字母或声音上的相似。我们还可以根据单词向量之间的差异来计算单词之间的差异。例如，国王和男人之间的差异类似于王后和女人之间的差异。也就是说，如果我们向女人的向量中添加国王和男人之间的差异，与结果向量最接近的单词是王后！同样，如果我们向美国中添加拉面和日本之间的差异，我们得到的是墨西哥卷饼。 通过使用神经网络和分布式表示法来处理单词，我们让 AI 理解语言中单词的语义相似性，使我们更接近能够理解和生成人类语言的 AI。

Neural Networks 神经网络

Recall that a **neural network** takes some input, passes it to the network, and creates some output. By providing the network with training data, it can do more and more of an accurate job of translating the input into an output. Commonly, machine translation uses neural networks. In practice, when we are translating words, we want to translate a sentence or paragraph. Since a sentence is a fixed size, we run into the problem of translating a sequence to another sequence where sizes are not fixed. If you have ever had a conversation with an AI chatbot, it needs to understand a sequence of words and generate an appropriate sequence as output.

回想一下，神经网络接收一些输入，将其传递给网络，并生成一些输出。通过向网络提供训练数据，它可以越来越准确地将输入转换为输出。通常，机器翻译使用神经网络。在实践中，当我们进行翻译时，我们希望翻译整个句子或段落。由于句子的大小是固定的，我们遇到了将一个序列翻译为另一个大小不固定的序列的问题。如果你曾经与 AI 聊天机器人进行过对话，它需要理解一系列单词并生成适当的输出序列。

Recurrent neural networks can re-run the neural network multiple times, keeping track of a state that holds all relevant information. Input is taken into the network, creating a hidden state. Passing a second input into the encoder, along with the first hidden state, produces a new hidden state. This process is repeated until an end token is passed. Then, a decoding state begins, creating hidden state after hidden state until we get the final word and another end token. Some problems, however, arise. One problem in the encoder stage where all the information from the input stage must be stored in one final state. For large sequences, it's very challenging to store all that information into a single state value. It would be useful to somehow combine all the hidden states. Another problem is that some of the hidden states in the input sequence are more important than others. Could there be some way to know what states (or words) are more important than others?

循环神经网络可以多次运行神经网络，跟踪一个包含所有相关信息的状态。输入被输入到网络中，创建一个隐藏状态。将第二个输入输入到编码器中，同时带上第一个隐藏状态，产生一个新的隐藏状态。这个过程重复直到传递一个结束标记。然后开始解码状态，创建一个接一个的隐藏状态，直到得到最终的单词和另一个结束标记。然而，也存在一些问题。在编码器阶段，所有输入阶段的信息都必须存储在一个最终状态中。对于长序列，将所有信息存储到单个状态值中是非常具有挑战性的。有某种方法将所有隐藏状态组合起来会很有用。另一个问题是输入序列中的某些隐藏状态比其他状态更重要。是否有可能知道哪些状态（或单词）比其他状态更重要?

Attention 注意

Attention refers to the neural network's ability to decide what values are more important than others. In the sentence "What is the capital of Massachusetts," attention allows the neural network to decide what values it will pay attention to at each stage of generating the output sentence. Running such a calculation, the neural network will show that when generating the final word of the answer, "capital" and "Massachusetts" are the most important to pay attention to. By taking the attention score, multiplying them by the hidden state values generated by the network, and adding them up, the neural network will create a final context vector that the decoder can use to calculate the final word. A challenge that arises in calculations such as these is that recurrent neural networks require sequential training of word after word. This takes a lot of time. With the growth of large language models, they take longer and longer to train. A desire for parallelism has steadily grown as larger and larger datasets need to be trained. Hence, a new architecture has been introduced.

注意力是指神经网络决定哪些值比其他值更重要的能力。在句子"马萨诸塞州的首都是什么"中，注意力允许神经网络决定在生成输出句子的每个阶段会关注哪些值。在进行这样的计算时，神经网络会显示，在生成答案的最终单词"首都"和"马萨诸塞州"时，它们是最需要关注的。通过使用注意力分数，将它们与网络生成的隐藏状态值相乘，然后相加，神经网络将创建一个最终上下文向量，解码器可以使用它来计算最终单词。在这样的计算中出现的一个挑战是，循环神经网络需要对一个接一个的单词进行顺序训练。这需要大量的时间。随着大型语言模型的发展，它们的训练时间越来越长。随着需要训练的大型数据集的增加，对并行性的需求稳步增长。因此，引入了一种新的架构。

Transformers 变换器

Transformers is a new type of training architecture whereby each input word is passed through a neural network simultaneously. An input word goes into the neural network and is captured as an encoded representation. Because all words are fed into the neural network at the same time, word order could easily be lost. Accordingly, **position encoding** is added to the inputs. The neural network, therefore, will use both the word and the position of the word in the encoded representation. Additionally, a **self-attention** step is added to help define the context of the word being inputted. In fact, neural networks will often use multiple self-attention steps such that they can further understand the context. This process is repeated multiple times for each of the words in the sequence. What results are encoded representations that will be useful when it's time to decode the information.

Transformer 是一种新的训练架构，其中每个输入单词同时通过神经网络。输入单词进入神经网络并被编码为表示。由于所有单词同时被输入到神经网络中，单词顺序很容易丢失。因此，添加位置编码到输入中。因此，神经网络将使用单词及其在编码表示中的位置。此外，添加了一个自我注意力步骤来帮助定义输入单词的上下文。事实上，神经网络通常会使用多个自我注意力步骤，以便进一步理解上下文。这个过程对序列中的每个单词重复多次。结果是编码表示，当解码信息时将非常有用。

In the decoding step, the previous output word and its positional encoding are given to multiple self-attention steps and the neural network. Additionally, multiple attention steps are fed the encoded representation from the encoding process and provided to the neural network. Hence, words are able to pay attention to each other. Further, parallel processing is possible, and the calculations are fast and accurate.

解码步骤中，神经网络接收前一个输出单词及其位置编码，并经过多个自我注意力步骤。此外，多个注意力步骤接收编码过程中的编码表示，并提供给神经网络。因此，单词能够相互注意。进一步地，可以进行并行处理，计算快速且准确。

Summing Up 总结

We have looked at artificial intelligence in a wide array of contexts. We looked at search problems in how AI can look for solutions. We looked at how AI represents knowledge and create knowledge. We looked at uncertainty when it does not know things for sure. We looked at optimization, maximizing and minimizing function. We looked at machine learning, finding patterns by looking at training data. We learned about neural networks and how they use weights to go from input to output. Today, we looked at language itself and how we can get the computer to understand our language. We have just scratched the surface of this process. We truly hope you enjoyed this journey with us. This was Introduction to Artificial Intelligence with Python.

我们在各种背景下探讨了人工智能。我们研究了 AI 如何寻找解决方案的搜索问题。我们探讨了 AI 如何表示和创造知识。我们研究了不确定性，即在 AI 不一定知道事情的时候。我们研究了优化，最大化和最小化函数。我们研究了机器学习，通过查看训练数据来发现模式。我们了解了神经网络以及它们如何使用权重从输入到输出。今天，我们探讨了语言本身以及如何让计算机理解我们的语言。我们只是触及了这个过程的表面。我们真心希望您喜欢与我们的旅程。这是使用 Python 的人工智能入门。