TOPICS IN COMPUTATIONAL LINGUISTICS

Neural Network Models of Linguistic Structure

COURSE NOTES

Sophie Hao
Yale University
LING 380/780

FALL 2021

Syllabus

Lecture Schedule (Tentative)

- **September 2.** Introduction, syllabus, course logistics (class outside due to power outage)

Chapter 1: Semantics

What is the meaning of meaning? We start the course by learning how modern machine learning algorithms perceive the world through the language of linear algebra.

- **September 7.** Semantics: features, linear algebra, embeddings
- **September 9.** Semantics: The word2vec model, logistic regression

Chapter 2: Learning

We introduce the basic principles of machine learning and the framework of neural networks.

- **September 14.** Machine learning basics: datasets, model architectures, loss functions, optimization algorithms
- **September 16.** Introduction to neural networks: simple and multi-layer perceptrons, activation functions, linear separability, word2vec as neural networks
- **September 21.** Introduction to neural networks (continued): n-gram causal language modeling
- **September 23.** Optimization: gradient descent, stochastic gradient descent, gradient computation
- **September 28.** Optimization (continued): Training/development/testing, overfitting, generalization, regularization
- **September 30.** The backpropagation algorithm: computation graphs, automatic differentiation, the linear layer
- **October 5.** The backpropagation algorithm (continued)

- **October 7.** The PyTorch library

Chapter 3: Recurrent Models

We learn about specialized neural networks designed to model natural language.

- **October 12.** Recurrent neural networks: simple recurrent networks, long short-term memory networks, gated recurrent networks
- **October 14.** Recurrent neural networks (continued): causal language modeling
- **October 19.** Syntactic parsing: tree-structured recurrent neural networks, recurrent neural network grammars, shift–reduce parsers
- **October 26.** Sequence-to-sequence tasks: encoder–decoder networks
- **October 28.** Sequence-to-sequence tasks (continued): Bahdanau attention and Luong attention

Chapter 4: Transformer Models

We learn about the state of the art in neural network models of language: transfer learning via pre-trained Transformer networks.

- **November 2.** Transformer networks: self-attention and the Transformer architecture
- **November 4.** Transfer learning: pre-training and fine-tuning, the BERT model

Journal Club

Students will become teachers and experience one of the most important activities of computational linguistics researchers: participating in a journal club. In small groups, students will read a research paper from the literature, present what they have learned to the class, and lead a group discussion.

- **November 9 and 11.** Neural network models of linguistic structure: Do neural networks understand natural language grammar?
- **November 16 and 18.** Neural network models of cognition and language processing: How can AI help us understand ourselves?
- **November 30 and December 2.** Neural network models of language acquisition: inductive bias, poverty of the stimulus arguments
- **December 7 and December 9.** Ethics of language technology: fairness, social bias, environmental impact of deep learning

Contents

CHAPTER 1 Semantics

In linguistics, *semantics* is the study of meaning. In this chapter we will learn how word meanings are represented in neural NLP systems.

IN-CLASS EXERCISE

(1.1) EXERCISE
Is a burrito a sandwich? Consider that the following items are often referred to as "sandwiches":

a. *a peanut butter and jelly sandwich (with two slices of bread)*
b. *a submarine sandwich, also known in Connecticut as a "grinder" (consisting of a single long loaf of bread with a longitudinal slice that contains the fillings)*
c. *a Scandinavian open-faced sandwich (consisting of a single slice of bread with toppings)*
d. *a Gyro sandwich (consisting of meat and vegetables wrapped with a piece of flatbread).*

This question was litigated in the lawsuit *White City Shopping Center v. Panera Bread*.[1] Panera Bread, a fast-casual sandwich restaurant, was given the exclusive right to sell sandwiches at the White City Shopping Center in Shrewsbury, Massachusetts. When Panera found out that White City was negotiating a contract with Qdoba, a fast-casual Mexican restaurant specializing in burritos, they sued the shopping center for breach of contract. Their argument was that burritos are sandwiches, and therefore allowing Qdoba to open a restaurant at the shopping center violated Panera's exclusive right to sell sandwiches. In other words, whether or not White City was allowed to negotiate a contract with Qdoba crucially depended on the meaning of the word *sandwich*.

(1.2) EXERCISE
What do you think the verdict should be? Did White City Shopping Center breach its contract with Panera by allowing Qdoba to sell burritos?

The judge's final opinion states (emphasis added):

> Given that the term "sandwiches" is not ambiguous and [Panera's] Lease does not provide a definition of it, this court applies the **ordinary meaning** of the word. The New Webster Third International Dictionary describes a "sandwich"

> as "two thin pieces of bread, usually buttered, with a thin layer (as of meat, cheese, or savory mixture) spread between them." … Under this definition and **as dictated by common sense**, this court finds that the term "sandwich" **is not commonly understood to include** burritos, tacos, and quesadillas, which are typically made with a single tortilla and stuffed with a choice filling of meat, rice, and beans. As such, there is no viable legal basis for barring White City from leasing to [Qdoba].

(1.3) EXERCISE
What is the judge's rationale? What does it mean for a word to "mean" something?

1.1 What Is Meaning?

What exactly is meaning? Philosophers of language spend a lot of time thinking about this question, and about what certain natural language expressions mean. Linguist Leonard Bloomfield[2] believed that the nature of meaning was too complex for linguists to describe and understand. In an influential book, he wrote:

> We can define the meaning of a speech-form accurately when this meaning has to do with some matter of which we possess scientific knowledge … but we have no precise way of defining words like *love* or *hate*, which concern situations that have not been accurately classified—and these latter are in the great majority.
>
> Moreover, even where we have some scientific (that is, universally recognized and accurate) classification, we often find that the meanings of a language do not agree with this classification. The whale is in German called a 'fish': *Walfisch* … and the bat a 'mouse': *Fledermaus* …. Physicists view the color-spectrum as a continuous scale of light-waves of different lengths, ranging from 40 to 72 hundred-thousandths of a millimeter, but languages mark off different parts of this scale quite arbitrarily and without precise limits, in the meanings of such color-names as *violet*, *blue*, *green*, *yellow*, *orange*, *red*, and the color-names of different languages do not embrace the same gradations. The kinship of persons seems a simple matter, but the terminologies of kinship that are used in various languages are extremely hard to analyze.
>
> The statement of meanings is therefore the weak point in language-study, and will remain so until human knowledge advances very far beyond its present state. (Bloomfield, 1933, ch. 9)

According to Bloomfield, each word has a "true meaning," and it is the job of experts to study those meanings and describe them carefully and precisely. Pluto, for example, would not count as a planet, because astronomers have spent years thinking about what it means to be a "planet," and decided that Pluto doesn't fit the definition. But language is full of common-sense concepts like "love" or

[1] `https://casetext.com/case/white-city-v-pr-restaurants`
[2] He was a Sterling Professor at Yale from 1940 to 1949!

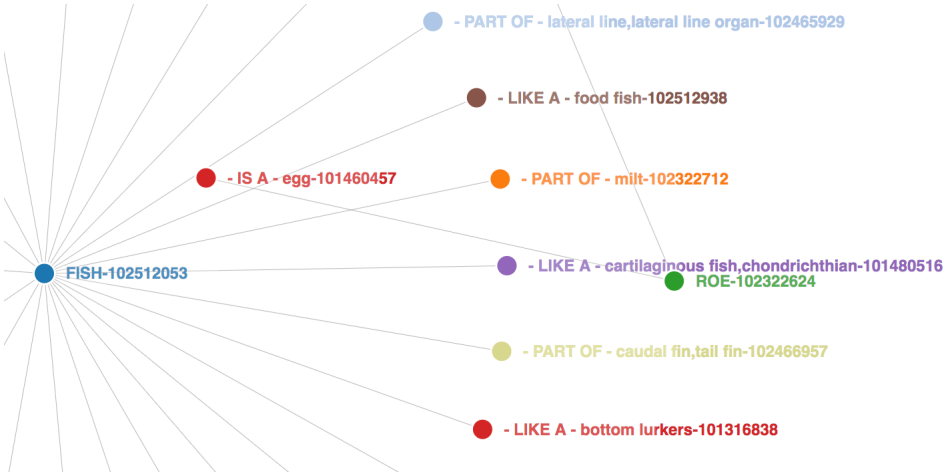

FIGURE 1.1: Relationships between word senses in the WordNet ontology.

"hate" that are difficult to define precisely even if we understand them intuitively. Bloomfield argues in the passage above that we cannot truly understand the nature of meaning until humankind has a better understanding of the concepts that we talk about on a day-to-day basis.

(1.4) EXERCISE
How would Bloomfield define a sandwich? Would Bloomfield consider a burrito to be a sandwich? Would Bloomfield agree with the judge's opinion in White City Shopping Center v. Panera Bread?

(1.5) EXERCISE
Do you agree with Bloomfield's understanding of meaning, and do you think it's appropriate for language technology? Why or why not?

In NLP, Bloomfield's idea of meaning is implemented through *ontologies*. An ontology tries to do what Bloomfield thought was impossible: it is a huge database that contains everything a typical person might know in a machine-readable format. One example of an ontology is WordNet,[3] a database that contains relationships between different word meanings. For each word, the database contains a number of *word senses*, or different meanings of that word. For example, here are the word senses for the word *fish* in WordNet.

(1.6) Word senses for *fish*

a. (noun) Any of various mostly cold-blooded aquatic vertebrates usually having scales and breathing through gills
b. (noun) The flesh of fish used as food
c. (noun, astrology) A person who is born while the sun is in Pisces
d. (noun) The twelfth sign of the zodiac; the sun is in this sign from about February 19 to March 20

[3] `https://wordnet.princeton.edu/`

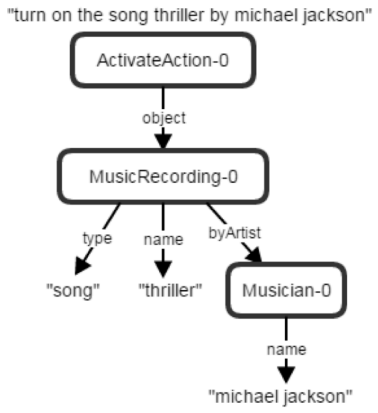

FIGURE 1.2: The abstract representation of the English sentence *turn on the song thriller by michael jackson* used by Amazon Alexa (from Kollar et al., 2018). The rectangles represent ontology objects.

e. (verb) Seek indirectly
f. (verb) Catch or try to catch fish or shellfish

The word senses are connected to one another by a graph that represents various relationships between senses. Figure 1.1 shows that according to WordNet, a caudal fin or tail fin is PART OF a fish; a cartilaginous fish or chondrichthian is LIKE A fish; and a roe IS A egg.[4]

While WordNet aims to describe the entire English language, most ontologies employed in NLP systems are limited to a narrow domain of application. For example, Amazon Alexa contains an ontology that describes the various actions (or "skills") that Alexa can do. A technical paper describes the ontology as follows:[5]

> The Alexa ontology … consists of five primary components:
> - **Classes** A hierarchy of Classes, also referred to as types, is defined in the ontology. This hierarchy is a rooted tree, with finer-grained types at deeper levels. Coarse types that are children of THING include PERSON, PLACE, INTANGIBLE, ACTION, PRODUCT, CREATIVEWORK, EVENT and ORGANIZATION. Finer-grained types include MUSICIAN and RESTAURANT.
> - **Properties** A given class contains a list of properties, which relate that class to other classes. Properties are defined in a hierarchy, with finer-grained classes inheriting the properties of its parent. There are range restrictions on the available types for both the domain and range of the property.
> - **Actions** A hierarchy of actions are defined as classes within the ontology. ACTIONS cover the core functionality of Alexa.
> - **Roles** ACTIONS operate on entities via roles. The most common role for an ACTION is the **.object** role, which is defined to be the entity on which the ACTION operates.

[4] You can look at other WordNet graphs at: `https://www.visual-thesaurus.com/wordnet.php`
[5] The documentation is available here: `https://amzn.to/3jrjR9Z`

- **Operators and Relations** A hierarchy of operators and relations represent complex relationships that cannot be expressed easily as properties. Represented as classes, these include ComparativeOperator, Equals and Coordinator ....

(Kollar et al., 2018)

The ontology objects are used to convert natural-language *requests* into an abstract representation of what the user wants Alexa to do. In Figure 1.2, the request *turn on the song thriller by michael jackson* is interpreted as an ActivateAction whose **.object** is a MusicRecording. This MusicRecording, in turn, has the **.byArtist** relation with a Musician whose **.name** is "michael jackson." The various concepts that might appear in a request, like ActivateAction, MusicRecording, and Musician, are all defined in the ontology.

1.2 The Geometry of Words

Ontologies are an important component of language-based technologies, but they are orthogonal to the material of this course. In this chapter, our goal will be to develop a representation of meaning that is compatible with neural networks. To do so, we will need to be proficient in *linear algebra*, the language of neural networks. Linear algebra is the branch of mathematics that deals with *vectors* and *matrices* and their geometric interpretations. We briefly go over some of the basic definitions here; but because knowledge of linear algebra is a pre-requisite for this course, this should all be review for you, except for the definition of the Hadamard product.

(1.7) DEFINITION

An $m \times n$ *matrix* is a collection of real numbers arranged into a grid with m rows and n columns. We denote by $\mathbb{R}^{m \times n}$ the set of all $m \times n$ matrices. We use bold, italic, uppercase letters to refer to matrices; e.g.: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \ldots$. For a matrix $\boldsymbol{A}$, the entry in row i and column j is denoted by $A_{i,j}$, where $1 \leq i \leq m$ and $1 \leq j \leq n$. We use the following notation to refer to parts of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$:

- $\boldsymbol{A}_{i,:}$ is the ith row of $\boldsymbol{A}$ and $\boldsymbol{A}_{:,j}$ is the jth column of $\boldsymbol{A}$
- $\boldsymbol{A}_{i,j:k}$ consists of row i and columns j through k of $\boldsymbol{A}$ (inclusive)
- $\boldsymbol{A}_{i,j:}$ consists of row i and columns j through n of $\boldsymbol{A}$ (inclusive)
- $\boldsymbol{A}_{i,:k}$ consists of row i and columns 1 through k of $\boldsymbol{A}$ (inclusive)
- $\boldsymbol{A}_{i:,j}$, $\boldsymbol{A}_{:k,j}$, and $\boldsymbol{A}_{i:h,j:k}$ are defined analogously for $1 \leq i < h \leq m$ and $1 \leq j < k \leq n$.

A *vector* or *column vector* is a matrix with only one column. We refer to vectors using bold, italic, lowercase letters such as $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$, and we use the notation $\mathbb{R}^m$ to refer to the set $\mathbb{R}^{m \times 1}$. A *row vector* is a matrix with only one row; i.e., an element of the set $\mathbb{R}^{1 \times n}$ for some n. We refer to row vectors using the notation $\boldsymbol{a}^\intercal$, where $\boldsymbol{a}$ is the column vector with the same entries as $\boldsymbol{a}^\intercal$. The *dimension* of a column vector $\boldsymbol{a} \in \mathbb{R}^m$ is m; the *dimension* of a row vector $\boldsymbol{b}^\intercal \in \mathbb{R}^{1 \times n}$ is n. The ith entry of a column vector $\boldsymbol{v}$ is denoted by v_i. The notations $\boldsymbol{v}_{i:}$, $\boldsymbol{v}_{:j}$, and $\boldsymbol{v}_{i:j}$ are defined analogously to matrices.

(1.8) DEFINITION

For matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{m \times n}$ and scalar $c \in \mathbb{R}$, the following operations are defined.

- *Matrix addition*:

$$\boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} A_{1,1} + B_{1,1} & A_{1,2} + B_{1,2} & \dots & A_{1,n} + B_{1,n} \\ A_{2,1} + B_{2,1} & A_{2,2} + B_{2,2} & \dots & A_{2,n} + B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} + B_{m,1} & A_{m,2} + B_{m,2} & \dots & A_{m,n} + B_{m,n} \end{bmatrix}$$

- *Matrix subtraction*:

$$\boldsymbol{A} - \boldsymbol{B} = \begin{bmatrix} A_{1,1} - B_{1,1} & A_{1,2} - B_{1,2} & \dots & A_{1,n} - B_{1,n} \\ A_{2,1} - B_{2,1} & A_{2,2} - B_{2,2} & \dots & A_{2,n} - B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} - B_{m,1} & A_{m,2} - B_{m,2} & \dots & A_{m,n} - B_{m,n} \end{bmatrix}$$

- *Hadamard product*:

$$\boldsymbol{A} \odot \boldsymbol{B} = \begin{bmatrix} A_{1,1}B_{1,1} & A_{1,2}B_{1,2} & \dots & A_{1,n}B_{1,n} \\ A_{2,1}B_{2,1} & A_{2,2}B_{2,2} & \dots & A_{2,n}B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1}B_{m,1} & A_{m,2}B_{m,2} & \dots & A_{m,n}B_{m,n} \end{bmatrix}$$

- *Hadamard quotient*:

$$\frac{\boldsymbol{A}}{\boldsymbol{B}} = \boldsymbol{A}/\boldsymbol{B} = \begin{bmatrix} A_{1,1}/B_{1,1} & A_{1,2}/B_{1,2} & \dots & A_{1,n}/B_{1,n} \\ A_{2,1}/B_{2,1} & A_{2,2}/B_{2,2} & \dots & A_{2,n}/B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1}/B_{m,1} & A_{m,2}/B_{m,2} & \dots & A_{m,n}/B_{m,n} \end{bmatrix}$$

- *Scalar multiplication*:

$$c\boldsymbol{A} = \boldsymbol{A}c = \begin{bmatrix} cA_{1,1} & cA_{1,2} & \dots & cA_{1,n} \\ cA_{2,1} & cA_{2,2} & \dots & cA_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ cA_{m,1} & cA_{m,2} & \dots & cA_{m,n} \end{bmatrix}$$

- *Transpose*:

$$\boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & \dots & A_{m,1} \\ A_{1,2} & A_{2,2} & \dots & A_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1,n} & A_{2,n} & \dots & A_{m,n} \end{bmatrix}$$

If $f : \mathbb{R} \to \mathbb{R}$ is a function mapping scalars to scalars, we extend f to matrices by applying them elementwise.

$$f(\boldsymbol{A}) = \begin{bmatrix} f(A_{1,1}) & f(A_{1,2}) & \dots & f(A_{1,n}) \\ f(A_{2,1}) & f(A_{2,2}) & \dots & f(A_{2,n}) \\ \vdots & \vdots & \ddots & \vdots \\ f(A_{m,1}) & f(A_{m,2}) & \dots & f(A_{m,n}) \end{bmatrix}$$

For vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$, we refer to the operation $\boldsymbol{u} + \boldsymbol{v}$ as *vector addition* instead of matrix addition and $\boldsymbol{u} - \boldsymbol{v}$ as *vector subtraction* instead of matrix subtraction. We define the following operations for vectors.

- *Dot product* (note that this is defined between the row vector $\boldsymbol{u}^\top$ and the column vector $\boldsymbol{v}$):

$$\boldsymbol{u}^\top \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i$$

- *Vector projection* (the *projection of* $\boldsymbol{u}$ *onto* $\boldsymbol{v}$):

$$\text{proj}_{\boldsymbol{v}}(\boldsymbol{u}) = \frac{\boldsymbol{u}^\top \boldsymbol{v}}{\|\boldsymbol{v}\|^2} \boldsymbol{v}$$

- *Vector norm* (a.k.a. *vector length*):

$$\|\boldsymbol{u}\| = \sqrt{\boldsymbol{u}^\top \boldsymbol{u}}$$

Finally, when $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, we define the operation of *matrix multiplication* by

$$\boldsymbol{AB} = \begin{bmatrix} \boldsymbol{A}_{1,:}\boldsymbol{B}_{:,1} & \boldsymbol{A}_{1,:}\boldsymbol{B}_{:,2} & \dots & \boldsymbol{A}_{1,:}\boldsymbol{B}_{:,p} \\ \boldsymbol{A}_{2,:}\boldsymbol{B}_{:,1} & \boldsymbol{A}_{2,:}\boldsymbol{B}_{:,2} & \dots & \boldsymbol{A}_{2,:}\boldsymbol{B}_{:,p} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{m,:}\boldsymbol{B}_{:,1} & \boldsymbol{A}_{m,:}\boldsymbol{B}_{:,2} & \dots & \boldsymbol{A}_{m,:}\boldsymbol{B}_{:,p} \end{bmatrix}.$$

(1.9) DEFINITION

Let $\mathbb{B} \subseteq \mathbb{R}^n$ be a set of n-dimensional vectors. The *span of* $\mathbb{B}$ is the set

$$\text{span}(\mathbb{B}) = \left\{ a_1 \boldsymbol{b}^{(1)} + a_2 \boldsymbol{b}^{(2)} + \cdots + a_k \boldsymbol{b}^{(k)} \middle| \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \dots, \boldsymbol{b}^{(k)} \in \mathbb{B} \text{ and } a_1, a_2, \dots, a_k \in \mathbb{R} \right\}.$$

A set $\mathbb{A} \subseteq \mathbb{R}^n$ is a *vector space* if $\mathbb{A} = \text{span}(\mathbb{B})$ for some $\mathbb{B} \subseteq \mathbb{R}^n$. We say that $\mathbb{B}$ is a *basis* for $\mathbb{A}$ if $\text{span}(\mathbb{B}) = \mathbb{A}$ and for every set $\mathbb{C} \subseteq \mathbb{R}^n$, if $\text{span}(\mathbb{C}) = \mathbb{A}$, then $|\mathbb{C}| \geq |\mathbb{B}|$. The *dimension* of a vector space $\mathbb{A}$ is $\dim(\mathbb{A}) = |\mathbb{B}|$, where $\mathbb{B}$ is a basis for $\mathbb{A}$.

For reasons that will become apparent in the next chapter, we will use an unusual definition of linear and bilinear maps that allows a constant "bias term" to be added.

(1.10) DEFINITION

A *linear map* is a function $f : \mathbb{R}^m \to \mathbb{R}^n$ of the form

$$f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

for some matrix $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ and vector $\boldsymbol{b} \in \mathbb{R}^n$. We refer to $\boldsymbol{W}$ as a *weight matrix* and $\boldsymbol{b}$ as a *bias vector*.

A *bilinear map* is a function $g : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ of the form

$$g(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{W}\boldsymbol{y} + b$$

for some matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ and scalar $b \in \mathbb{R}$. Again, we refer to $\boldsymbol{W}$ as a *weight matrix*, and we refer to b as a *bias term*.

	Basil	Crushed Italian Tomatoes	Fresh Basil	Fresh Clams	Fresh Mozzarella	Fresh Native Tomatoes	Garlic	Grated Pecorino Romano	Meatball	Mozzarella	Olive Oil	Oregano	Ricotta Cheese
White Clam Pizza	0	0	0	1	0	0	1	1	0	0	1	1	0
Frank Pepe's Original Tomato Pie	0	1	0	0	0	0	0	1	0	0	1	0	0
Margherita	0	1	1	0	1	0	0	1	0	0	1	0	0
Fresh Tomato Pie	1	0	0	0	0	1	1	1	0	1	1	0	0
Meatball & Ricotta	0	1	0	0	0	0	0	1	1	1	1	0	1

FIGURE 1.3: Selected menu items from Frank Pepe's Pizzeria Napoletana.[6]

(1.11) EXERCISE
Let $\mathbb{B} \subseteq \mathbb{R}^n$ be a finite set of n-dimensional vectors. Let $\boldsymbol{B} \in \mathbb{R}^{n \times |\mathbb{B}|}$ be a matrix where each column corresponds to an element of $\mathbb{B}$; i.e., $\boldsymbol{b} \in \mathbb{B}$ if and only if $\boldsymbol{b} = \boldsymbol{B}_{:,j}$ for some j. Prove that $\boldsymbol{v} \in \text{span}(\mathbb{B})$ if and only if $\boldsymbol{v} = \boldsymbol{B}\boldsymbol{a}$ for some $\boldsymbol{a} \in \mathbb{R}^{|\mathbb{B}|}$.

1.2.1 Feature Spaces

In order to apply neural networks to language, we need to be able to translate linguistic objects into the language of linear algebra. Vectors are typically interpreted geometrically as points or arrows in a vector space. Linguistic objects like words or sentences, however, are typically thought of as *discrete* objects: there is no such thing as "half of a word" or "the average of two sentences." Our challenge is to represent discrete linguistic objects using the *continuous* formalism of linear algebra.

In machine learning, discrete objects are represented linear-algebraically by *feature vectors*. To illustrate, take a look at the matrix in Figure 1.3. Let's call this matrix $\boldsymbol{P}$. This matrix is a feature representation of various menu items from Frank Pepe's Pizzeria Napoletana in Wooster Square.[7] Each column represents a pizza topping, and each row represents one of the pizzas on the menu. $P_{i,j}$ is 1 if pizza i has topping j, and $P_{i,j}$ is 0 otherwise.

IN-CLASS EXERCISE

(1.12) EXERCISE
Let $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{13}$ represent two pizzas, using the same features as in Figure 1.3. What do $\boldsymbol{p} + \boldsymbol{q}$,

[6] `https://www.pepespizzeria.com/wp-content/uploads/2021/06/Pepe-Menu.T1.0721.pdf`

[7] If you haven't been, I highly recommend it!

$\boldsymbol{p} \odot \boldsymbol{q}$, $\max(\boldsymbol{p}, \boldsymbol{q})$, $\boldsymbol{p}^\top \boldsymbol{q}$, *and* $\boldsymbol{p} - \boldsymbol{q}$ *represent? Which ones are valid pizzas?*

You might think that $\boldsymbol{p} - \boldsymbol{q}$ doesn't represent anything: if $p_i = 0$ but $q_i = 1$, then $(\boldsymbol{p} - \boldsymbol{q})_i = -1$, and it doesn't make sense for a pizza to have -1 units of topping i. But vector subtraction does have a useful interpretation—it helps us describe *analogies*.

(1.13) EXAMPLE
Define the following pizza vectors.[8]

$$\begin{aligned}
\boldsymbol{p}^\top &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} && \text{(white pie with meatballs)} \\
\boldsymbol{q}^\top &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} && \text{(plain white pie)} \\
\boldsymbol{r}^\top &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} && \text{(red pie with meatballs)} \\
\boldsymbol{s}^\top &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} && \text{(plain red pie)}
\end{aligned}$$

Observe that $\boldsymbol{p} - \boldsymbol{q} = \boldsymbol{r} - \boldsymbol{s}$. We interpret this equation as representing the following analogy: *a white pie with meatballs is to a plain white pie what a red pie with meatballs is to a plain red pie*. In NLP, we will often rewrite this equation in the form $\boldsymbol{p} - \boldsymbol{q} + \boldsymbol{s} = \boldsymbol{r}$. The vector $-\boldsymbol{q} + \boldsymbol{s}$ then represents the operation of replacing the properties of a plain white pie with those of a plain red pie.

When working with feature vectors, it is important that we be able to measure how similar or different two feature vectors are. The dot product plays a prominent role to that end. During the in-class exercise, you should have realized that the dot product between two pizzas is the number of toppings those two pizzas have in common. Certainly, the more toppings two pizzas share, the more similar the are. However, pizzas with more toppings will have a bigger dot product, and it does not seem fair to say that two pizzas with 50 toppings in common out of 100 are more similar to one another than two pizzas with 4 toppings in common out of 5. To compensate for this phenomenon, we define a measure of similarity that takes the dot product between two feature vectors, and then normalizes this dot product by the vectors' lengths.

(1.14) DEFINITION
The *cosine similarity* between two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ is defined as

$$\cos(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^\top \boldsymbol{v}}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|}.$$

Geometrically, $\cos(\boldsymbol{u}, \boldsymbol{v})$ is the cosine of the angle between the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$. This gives us a geometric interpretation of what it means for two feature vectors to share features. When the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$ is close to 0, and therefore $\boldsymbol{u}$ and $\boldsymbol{v}$ point in roughly the same direction, $\cos(\boldsymbol{u}, \boldsymbol{v}) \approx 1$; and when the angle is roughly π, and therefore $\boldsymbol{u}$ and $\boldsymbol{v}$ point in roughly opposite directions, $\cos(\boldsymbol{u}, \boldsymbol{v}) \approx -1$. Thus, when $\boldsymbol{u}$ and $\boldsymbol{v}$ share a lot of features, they point in roughly the same direction.

[8] In traditional New Haven pizzerias, a *white pie* is a pizza with tomato sauce and mozzarella cheese, while a *red pie* is a pizza with tomato sauce and no cheese.

While the cosine similarity of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ measures how similar $\boldsymbol{u}$ and $\boldsymbol{v}$ are, we can also use the *cosine distance*, defined as $1 - \cos(\boldsymbol{u}, \boldsymbol{v})$, to measure how *different* two vectors are.

(1.15) EXERCISE
Compute the cosine similarity between every two pizzas in Figure 1.3.

(1.16) EXERCISE
Consider the two most dissimilar pizzas possible: the pizza with all the toppings and the pizza with no toppings. What is the angle between their feature vectors?
Now, disregard the pizza example. What does it mean when two feature vectors point in completely opposite directions? Consider the fact that feature values don't have to always be 0 *or* 1*.*

1.2.2 Embeddings and Dimensionality Reduction

It is hard to derive any intuition from a list of numbers. If we are to understand feature vectors, it would be very helpful if we had a way of visualizing them. Unfortunately, feature spaces typically have lots and lots of features—our pizza vectors have 13 features, but in this course the smallest feature spaces will have at least 300 features—and we can't plot points in spaces of more than 2 dimensions. Therefore, we will need to *embed* our feature vectors into a 2-dimensional space.

In mathematics, an *embedding* is a function that converts objects in one space into analogous objects in another. For example, the circle $\{\boldsymbol{x} \in \mathbb{R}^2 \mid \|\boldsymbol{x}\| = 1\} \subseteq \mathbb{R}^2$ can be embedded in $\mathbb{R}^3$ as the circle $\{\boldsymbol{x} \in \mathbb{R}^3 \mid \|\boldsymbol{x}\| = 1 \text{ and } x_3 = 0\}$. Geometrically, we know that the original circle and the embedded circle have the exact same shape because the embedding is *isometric*: the distance between any two points on the original circle is the same as the distance between the analogous two points on the embedded circle.

(1.17) DEFINITION
Let $\mathbb{A}$ and $\mathbb{B}$ be two arbitrary sets. An *embedding of* $\mathbb{A}$ *into* $\mathbb{B}$ is a one-to-one mapping of $\mathbb{A}$ to $\mathbb{B}$; i.e., a function $f : \mathbb{A} \to \mathbb{B}$ such that for all $x, y \in \mathbb{A}$, if $f(x) = f(y)$, then $x = y$. We say that an embedding $f : \mathbb{A} \to \mathbb{B}$ is *isometric with respect to* d_1 *and* d_2 if $d_1 : \mathbb{A} \times \mathbb{A} \to \mathbb{R}$ and $d_2 : \mathbb{B} \times \mathbb{B} \to \mathbb{R}$ are functions such that for all $x, y \in \mathbb{A}$, $d_1(x, y) = d_2(f(x), f(y))$.

(1.18) DEFINITION
The *Euclidean distance* between two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ is defined as $\|\boldsymbol{u} - \boldsymbol{v}\|$.

(1.19) EXAMPLE
The function $f : \mathbb{R}^2 \to \mathbb{R}^3$ given by $f(\boldsymbol{x}) = \begin{bmatrix} x_1 & x_2 & 0 \end{bmatrix}^\top$ is an embedding of $\mathbb{R}^2$ into $\mathbb{R}^3$. It is isometric with respect to Euclidean distance and Euclidean distance.

[9] This embedding was computed using an algorithm called *multidimensional scaling* (MDS). You don't need to know about MDS, but if you're curious, you can learn about it here: `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html`

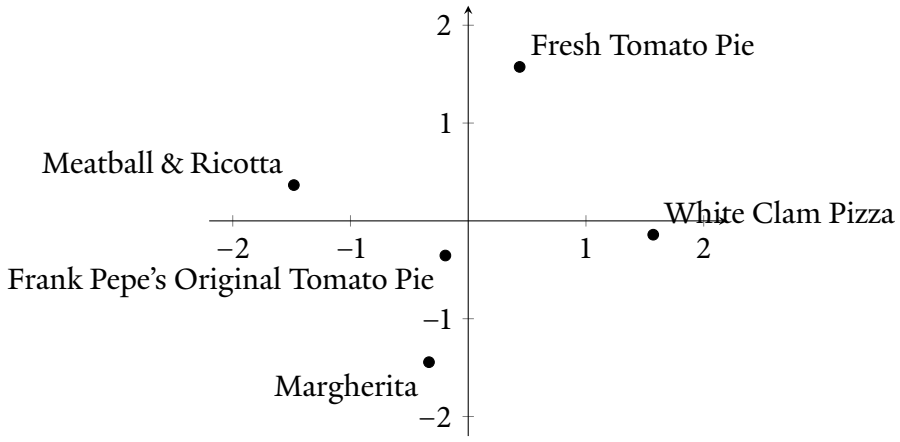

FIGURE 1.4: An embedding of our pizza vectors from $\mathbb{R}^{13}$ into $\mathbb{R}^2$. The embedding is approximately isometric with respect to the absolute value of cosine distance and Euclidean distance.[9]

Figure 1.4 shows an embedding of our pizza vectors into the 2D plane that is approximately isometric with respect to (the absolute value of) cosine distance and Euclidean distance. The closer two pizzas appear to one another within the plot, the greater their cosine similarity. From the diagram, we can quickly observe that Meatball & Ricotta, Frank Pepe's Original Tomato Pie, and Margherita are relatively similar to one another, while the Fresh Tomato Pie and White Clam Pizza are not so similar. Note that the actual coordinate values don't mean anything: the axes have no particular interpretation; the only thing that matters is the distance between points.

In data science, the task of embedding points in a high-dimensional space into a lower-dimensional space is known as *dimensionality reduction*. In this course, we will primarily use two algorithms for dimensionality reduction: *principal components analysis* (PCA) and *t-distributed stochastic neighbor embedding* (*t*-SNE). In the remainder of this section, we'll go over the basic intuition of how PCA works. A more detailed explanation of both PCA and t-SNE is available in Section 1.5.

The basic idea of PCA is that high-dimensional objects are often well-approximated by lower-dimensional objects, and therefore many of the dimensions can be dropped without losing too much information. For example, look at the data points plotted in Figure 1.5. The points, as a whole, take the form of an elogated ellipse, and a linear model along the major axis of the ellipse would serve as a good approximation of the data. The direction of this line is shown by the longer of the two arrows in the plot, and this direction is chosen in particular because it is the *direction of greatest variance*: the direction that preserves the most information if all points were projected to it, where information is quantified by the variance of the projected data points.

(1.20) EXAMPLE

Let's now use PCA to embed our pizza vectors into $\mathbb{R}^2$. We'll need to use two Python packages: NumPy, a package for doing linear algebra, and scikit-learn, a package that implements basic data science algorithms.

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
```

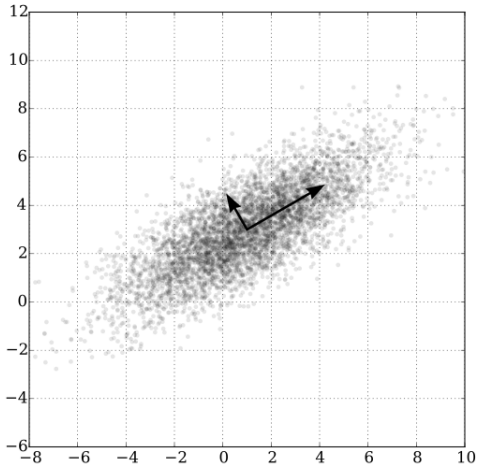

FIGURE 1.5: The two principal components of a 2-dimensional dataset. Image source: Wikimedia Commons user Nicoguaro (CC BY 4.0)

First, we enter in our pizza matrix $\boldsymbol{P}$ as a NumPy *array*. The array is a data structure that represents vectors and matrices. To create an array, we specify the entries of our matrix by making a list for each row, and then combining the rows into a list of lists.

```
>>> p = np.array(
... [[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0],
... [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
... [0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
... [1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0],
... [0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1]])
```

To perform dimensionality reduction, we create a `PCA` object and call the `fit_transform` method on our data matrix. We use the `n_components` keyword parameter to specify what vector space we want to embed our vectors into: setting `n_components=2` means we want to embed into $\mathbb{R}^2$. The return value of `fit_transform` gives us an array with the embedded coordinates, where each row contains the embedded coordinates of the corresponding row in $\boldsymbol{P}$.

```
>>> pca = PCA(n_components=2)
>>> embedded = pca.fit_transform(p)
>>> embedded
array([[ 1.01264811, -0.92395855],
       [-0.50931037, -0.23418494],
       [-0.9604    , -0.67259834],
       [ 1.21505519,  0.78969081],
       [-0.75799292,  1.04105102]])
```

These embeddings are computed by projecting our pizza vectors to the *first two principal components* of our data. The *first principal component* of our data is the direction of greatest

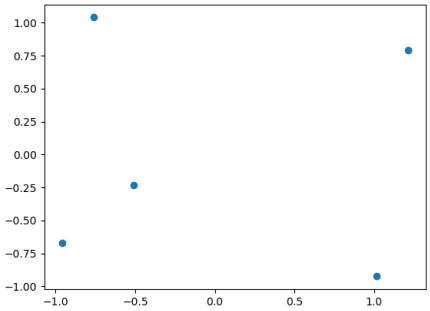

FIGURE 1.6: An embedding of our pizza vectors into $\mathbb{R}^2$ computed using PCA.

variance, while the *second principal component* is the direction orthogonal to the first principal component with the greatest variance; for $n > 2$, the nth principal component is the direction of greatest variance orthogonal to the first $n - 1$ principal components. These directions are defined mathematically as unit vectors. You can see the principal components used by the `PCA` object by looking at the `components_` attribute.

```
>>> pca.components_
array([[ 0.28534923, -0.52316423, -0.22554482,  0.23781501,
        -0.22554482,  0.28534923,  0.52316423,  0.        ,
        -0.17801059,  0.10733863,  0.        ,  0.23781501,
        -0.17801059],
       [ 0.25736834,  0.04375923, -0.2192067 , -0.30112757,
        -0.2192067 ,  0.25736834, -0.04375923,  0.        ,
         0.33928921,  0.59665754,  0.        , -0.30112757,
         0.33928921]])
```

Finally, let's visualize our embedded vectors. We do this by using the Matplotlib package, which provides tools for creating plots and figures. We first use the `scatter` function to create a scatter plot of our PCA-embedded pizza vectors, and then we use the `savefig` function to save our plot to the file `pca_pizza.png`.

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(embedded[:, 0], embedded[:, 1])
<matplotlib.collections.PathCollection object at 0x7fdda8245610>
>>> plt.savefig("pca_pizza.png")
```

Our PCA plot is shown in Figure 1.6.

(1.21) EXERCISE

What is the Euclidean distance between the PCA embeddings of each pair of pizzas? How do the Euclidean distances compare to the cosine distances between the original 13*-dimensional pizza vectors? Is the PCA embedding roughly isometric?*

Read the documentations for the `distance.euclidean`[10] *and* `distance.cosine`[11] *functions from the* `scipy.spacial` *package to find your answer.*

1.3 The Distributional Hypothesis

Our goal for the rest of this chapter is to figure out how to represent word meanings as feature vectors. Unlike pizzas, however, word meanings cannot be readily reduced to a small collection of features. However hard ontologists might try, the task of parameterizing the entirety of natural language semantics in terms of numerical features is inherently subject to the challenges that Bloomfield discusses in his book. In real life, word meanings are not precisely defined as they are in technical disciplines, and the usage of words varies substantially across individual speakers, communities, demographic groups, and languages.[12] And even if these problems were set aside, developing a compehensive representation of meaning would require an astronomical amount of engineering labor, presenting a major disadvantage in terms of scalability.

In this course we will assume that it is impossible for engineers to manually define word meanings in terms of feature vectors. Instead, we somehow need to find features for word meanings that can be automatically extracted from data. These features will be *latent*, or *hidden*: they cannot be directly observed in the data, and they do not have any *a priori* interpretation. However, we will make the assumption that word meanings exist in some abstract space where the "similarity" between two meanings can be quantified, and we will desire that our hidden features provide an embedding of word meanings from this abstract space into $\mathbb{R}^n$ that is isometric with respect to word similarity and cosine similarity. Although the individual hidden features will not have any interpretation, the cosine similarity between feature vectors will be interpreted as the similarity between word meanings, under the assumption that the embedding of words into $\mathbb{R}^n$ is isometric. Because their interpretations are based on the assumed existence of an isometric embedding, hidden feature vectors extracted from data that represent word meanings are known as *word embeddings*.

But how exactly do we find these hidden features? For that, we turn to linguist J. R. Firth (1957), who wrote (with the famous quotation in bold):

> As Wittgenstein says, 'the meaning of words lies in their use.' The day to day practice of playing language games recognizes customs and rules. It follows that a text in such established usage may contain sentences such as 'Don't be such an ass!', 'You silly ass!', 'What an ass he is!' In these examples, the word *ass* is in familiar and habitual company, commonly collocated with *you silly—*, *he is a silly—*, *don't be such an—*. **You shall know a word by the company it keeps!** One of the meanings of *ass* is its habitual collocation with such other words as those above quoted.

[10] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.euclidean.html`

[11] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cosine.html`

[12] How language might vary is an actively ongoing area of linguistics research, especially here at Yale. Check out the Yale Grammatical Diversity Project for more information: `https://ygdp.yale.edu/`

Bloomfield (1933) "[defines] the *meaning* of a [word] as the situation in which the speaker utters it and the response which it calls forth in the hearer." According to Bloomfield, we use a certain word, it is because we as speakers want to invoke a certain idea in our listeners; and ontologies assume that the right way to define a word's meaning is to describe the idea we want to invoke when we use that word. Firth, however, disagrees with Bloomfield. According to Firth, the meaning of a word is entirely captured by how we use that word. Specifically, the "usage" of a word is described by its *collocations*—statistical descriptions of what other words are used together with that word. In the quotation above, the vulgar meaning of *ass* is described by the fact that *ass* is frequently used with phrases like *he is a silly* and *don't be such an* that are commonly used with with profanity.

Firth's notion of meaning is known as the *distributional hypothesis*, and it has given rise to methods of creating word embeddings by modeling collocations. Although there are a variety of distributional methods for creating word embeddings, in the lecture-based portion of this course we will focus on just one such method.

IN-CLASS EXERCISE

(1.22) EXERCISE
Would Firth consider a burrito to be a sandwich? Why or why not?
Do you agree more with Firth's concept of meaning, or Bloomfield's? Which concept do you think the verdict from White City Shopping Center v. Panera Bread was based on?

It is interesting to note that the judge's opinion made reference to both Bloomfield's and Firth's notions of meaning: it simultaneously cited a dictionary definition of *sandwich* while also appealing to "common sense."

1.4 The word2vec Models of Distributional Semantics

The *word2vec* models (Mikolov et al., 2013a) are a family of distributional algorithms used to create word embeddings. Not only have the word2vec models been extremely influential in the development of neural NLP, they are also one of the simplest examples of how neural networks can be applied to the modeling of natural language. In the end of this chapter we will learn about one of the word2vec models, the *Skip-Gram with Negative Sampling* model (SGNS, Mikolov et al., 2013b; Goldberg and Levy, 2014), without the framework of neural networks. After we formally introduce neural networks in the next chapter, we will return to word2vec and see how the other models can be understood as neural networks.

The basic idea behind SGNS is to come up with word embeddings such that words that appear together have a high cosine similarity, while words that do not appear together have a low cosine similarity. More formally, we assume that there is a set of words $\mathbb{V}$ called the *vocabulary*, and to each word $w \in \mathbb{V}$ we assign a *word embedding* $[\![w]\!] \in \mathbb{R}^d$ and a *context embedding* $(\!|w|\!) \in \mathbb{R}^d$. These embeddings are arranged into matrices $\boldsymbol{W}, \boldsymbol{C} \in \mathbb{R}^{|\mathbb{V}| \times d}$ such that each row of $\boldsymbol{W}$ is the word embedding of some word and each row of $\boldsymbol{C}$ is the context embedding of some word. The entries of $\boldsymbol{W}$ and $\boldsymbol{C}$ are *parameters* of the model: we initialize them to random values, and a learning algorithm will adjust

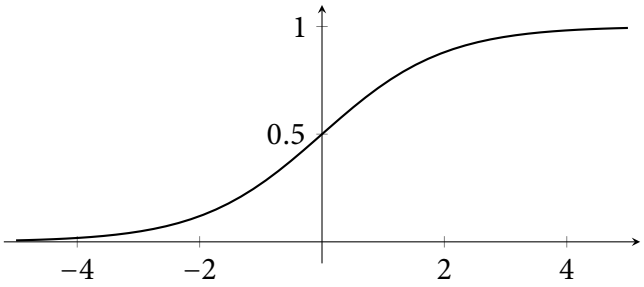

FIGURE 1.7: The sigmoid function, used in logistic regression.

these values to better reflect the data. The constant d, the dimensionality of our word embedding and context embedding vectors, is a *hyperparameter*: its value needs to be manually defined beforehand, and often times engineers and researchers will try a number of different hyperparameter values in order to see which settings work the best.

1.4.1 Binary Classification

In order to operationalize the SGNS model, we conceptualize the task of finding word embeddings as a *binary classification task*. A classification task is any task where feature vectors need to be assigned to one of two or more *classes*; a binary classification task is a classification task with exactly two possible classes. For SGNS, our binary classifier will take a word embedding and a context embedding, and classify this pair of vectors as "occurring together" or "not occurring together." We will build our classifier by fitting a *logistic model* to a dataset.

(1.23) DEFINITION

A *logistic model* is a function $f : \mathbb{R}^n \to \mathbb{R}$ of the form

$$f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

for some $\boldsymbol{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is the *sigmoid function* given by

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Note that for all $x \in \mathbb{R}$, $0 < \sigma(x) < 1$.

The heart of the logistic model is the *sigmoid curve*, shown in Figure 1.7. The sigmoid curve is an S-shaped curve that maps real numbers to numbers between 0 and 1. The sigmoid's y-intercept is $1/2$, and it is roughly linear near the y axis. When x has a large magnitude, however, the sigmoid approaches 1 as x tends to ∞ and 0 as x tends to $-\infty$. A logistic model combines the sigmoid curve with a linear map. Suppose we are classifying inputs into two classes, c_1 and c_2. The linear map computes a score that measures the degree to which the input resembles elements of one class or the other. If the score is positive, then the input is more likely to belong to class c_1, and if the score is negative, then the input is more likely to belong to class c_2. The sigmoid curve turns this score

w	c	y
dog	cat	1
dog	the	1
dog	big	1
dog	acetic	0
dog	pernicious	0
dog	spherification	0
⋮	⋮	⋮

FIGURE 1.8: An example of what a dataset for the SGNS classification task might look like.

into a number between 0 and 1, which we interpret as the probability that the input belongs to class c_1.

Now, let's design a logistic model for our word embeddings. Remember that we want $\cos(\llbracket w \rrbracket, (\!| c |\!))$ to be high when w and c occur together, and low when w and c do not occur together. Since cosine similarity is really just a normalized version of the dot product, we can easily turn the cosine similarity into a linear map by dropping the denominator and keeping the dot product. We then combine the dot product with the sigmoid curve to form a logistic model.

MODEL

(1.24) **Logistic Model for SGNS**

$$\mathbb{P}[\text{words } w \text{ and } c \text{ occur together}] = \sigma(\llbracket w \rrbracket^\top (\!| c |\!))$$

1.4.2 Logistic Regression

Now that we have our logistic model, we need to *fit* the model to a dataset in order to obtain our word and context embedding matrices. The dataset will be a table $\mathbb{D} \subseteq \mathbb{V} \times \mathbb{V} \times \{0,1\}$, with each element of $\mathbb{D}$ of the form (w, c, y), where $y = 1$ if w and c occur together and $y = 0$ if w and c do not occur together. Because the dataset indicates how each data point should be labeled by our logistic classifier, the classification task we are doing is an example of *supervised learning*. The process of fitting a logistic model to a dataset is called *logistic regression*.

To obtain the dataset $\mathbb{D}$, we take a *corpus* of text and extract *skip-grams* from it. A skip-gram is a sequence of words, with one word missing from the middle. For example, Figure 1.9 illustrates how the skip-gram *of New York said on* _____ *that he would* is extracted from a New York Times article. The missing word is flanked by 5 words to the left and 3 words to the right; the larger of these two numbers is the skip-gram's *window size*, which in this case is 5. For each word w in the corpus, we extract a skip-gram around that word, and for each c in the skip-gram, we add the tuple $(w, c, 1)$ to $\mathbb{D}$. The window size of the skip-gram is chosen randomly, with the maximum possible window size determined by a hyperparameter m. In order to add examples with a label of $y = 0$, we also randomly choose words $c_1, c_2, \ldots, c_{kn}$ from $\mathbb{V}$, where k is another hyperparameter, $n \leq 2m$

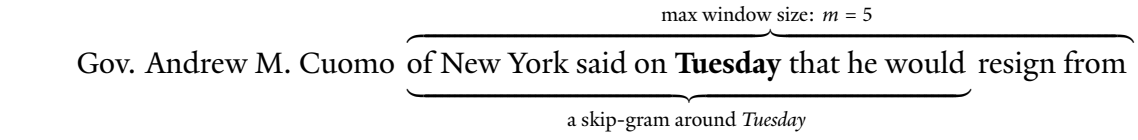

FIGURE 1.9: An example of a skip-gram with window size 5 around the word *Tuesday*.[13]

is the number of words in the skip-gram (excluding the missing word), and for each $i \leq kn$ we add the items $(w, c_i, 0)$ to $\mathbb{D}$.

After we have obtained our dataset $\mathbb{D}$, we solve the following *optimization problem*:

$$\max_{\boldsymbol{W},\boldsymbol{C}} \left(\prod_{(w,c,1)\in\mathbb{D}} \mathbb{P}[w \text{ and } c \text{ occur together}] \right) \left(\prod_{(w,c,0)\in\mathbb{D}} \mathbb{P}[w \text{ and } c \text{ do not occur together}] \right)$$

The problem above is read as follows: "find the values of $\boldsymbol{W}$ and $\boldsymbol{C}$ that result in the highest possible value of the expression." If we assume that each data point of $\mathbb{D}$ is independent and identically distributed (an assumption that might not be reasonable, but that we will make anyway just because it's convenient), then the expression being maximized is the probability that a dataset with the same (w, c) pairs as $\mathbb{D}$ would be labeled in the same way that $\mathbb{D}$ is labeled. In other words, our embedding matrices must cause our logistic model to predict that the words that actually do occur together in the real world have a high probability of doing so, while words that do not actually occur together in the real world have a low probability of occurring together. This learning strategy, in which we adjust our model parameters to maximize the probability of our dataset, is called *maximum likelihood estimation*.

Now, we make some simplifications to our optimization problem. First, plugging in our logistic model and observing that $1 - \sigma(x) = \sigma(-x)$ for all x, we rewrite the above as follows.

$$\max_{\boldsymbol{W},\boldsymbol{C}} \left(\prod_{(w,c,1)\in\mathbb{D}} \sigma(\llbracket w \rrbracket^\top (\!| c |\!)) \right) \left(\prod_{(w,c,0)\in\mathbb{D}} \sigma(-\llbracket w \rrbracket^\top (\!| c |\!)) \right)$$

Although our expression is now much simpler than it was before, it still has a subtle flaw: it is *numerically unstable*, meaning that it is prone to rounding errors when implemented in code. This is because products of probabilities become very, very small as more and more probabilities are multiplied together. Eventually, the product may accidentally get rounded to 0, at which point the entire expression is zeroed out. To alleviate this problem, we simply take the natural log of the entire expression. Since the natural log is an *increasing function* (i.e., for all $x, y > 0$ we have $x > y$ if and only if $\ln(x) > \ln(y)$), maximizing the natural log of an expression is the same thing as maximizing the original expression.

$$\max_{\boldsymbol{W},\boldsymbol{C}} \left(\sum_{(w,c,1)\in\mathbb{D}} \ln(\sigma(\llbracket w \rrbracket^\top (\!| c |\!))) \right) + \left(\sum_{(w,c,0)\in\mathbb{D}} \ln(\sigma(-\llbracket w \rrbracket^\top (\!| c |\!))) \right)$$

[13] `https://www.nytimes.com/2021/08/10/nyregion/andrew-cuomo-resigns.html`

Next, we multiply our expression by −1 and restate our optimization problem as a *minimization* problem instead of a maximization problem. There is no mathematical, statistical, or computational reason for doing this; we do it simply because regression problems such as linear regression or logistic regression are traditionally thought of as minimization problems.

$$\min_{\boldsymbol{W},\boldsymbol{C}} -\left(\sum_{(w,c,1)\in\mathbb{D}} \ln(\sigma([\![w]\!]^\top(\!|c|\!)))\right) - \left(\sum_{(w,c,0)\in\mathbb{D}} \ln(\sigma(-[\![w]\!]^\top(\!|c|\!)))\right)$$

Finally, because our dataset contains k times as many examples with a label of $y = 0$ as examples with a label of $y = 1$, we scale the terms associated with the label of $y = 0$ by $1/k$. Our final minimization problem is as follows.

$$\min_{\boldsymbol{W},\boldsymbol{C}} -\left(\sum_{(w,c,1)\in\mathbb{D}} \ln(\sigma([\![w]\!]^\top(\!|c|\!)))\right) - \frac{1}{k}\left(\sum_{(w,c,0)\in\mathbb{D}} \ln(\sigma(-[\![w]\!]^\top(\!|c|\!)))\right)$$

In the next chapter, we will learn how this minimization problem is solved. Once it is solved, we discard $\boldsymbol{C}$ and use $\boldsymbol{W}$ as our word embeddings.

To summarize, the full SGNS model is as follows.

MODEL: SKIP-GRAM WITH NEGATIVE SAMPLING

Hyperparameters:

- $\mathbb{V}$, a vocabulary
- d, the dimensionality of the word and context embeddings
- m, the maximum window size of skip-grams
- k, the number of $y = 0$ examples to include per $y = 1$ example

Parameters:

- $\boldsymbol{W} \in \mathbb{R}^{|\mathbb{V}|\times d}$, a matrix of word embeddings
- $\boldsymbol{C} \in \mathbb{R}^{|\mathbb{V}|\times d}$, a matrix of context embeddings

Data:

- A corpus of text

Procedure:

- Initialize $\mathbb{D}$ to an empty set.
- For each word w in the corpus:
  - Let $c_1, c_2, \ldots, c_n$ be a skip-gram around w with window size at most m. For each $i \le n$, add $(w, c_i, 1)$ to $\mathbb{D}$.
  - Let $c'_1, c'_2, \ldots, c'_{kn}$ be randomly sampled from $\mathbb{V}$. For each $i \le kn$, add $(w, c'_i, 0)$ to $\mathbb{D}$.
- Solve the following minimization problem:

$$\boldsymbol{W},\boldsymbol{C} \leftarrow \operatorname*{argmin}_{\boldsymbol{W},\boldsymbol{C}} -\left(\sum_{(w,c,1)\in\mathbb{D}} \ln(\sigma([\![w]\!]^\top(\!|c|\!)))\right) - \frac{1}{k}\left(\sum_{(w,c,0)\in\mathbb{D}} \ln(\sigma(-[\![w]\!]^\top(\!|c|\!)))\right)$$

- Return $\boldsymbol{W}$.

1.5 Optional Reading: Dimensionality Reduction

In this optional section, we learn about the PCA and t-SNE algorithms.

1.5.1 Principal Components Analysis

We have already seen the basic intuition behind PCA: we want to find the *principal components* of a set of feature vectors, which are defined as the *directions of greatest variance* for those vectors. More concretely, suppose we have a dataset $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, represented as a matrix in which each row contains an n-dimensional feature vector. We want to find a set of vectors

$$\mathbb{B} = \{\boldsymbol{\psi}^{(1)}, \boldsymbol{\psi}^{(2)}, \ldots, \boldsymbol{\psi}^{(n)}\}$$

with the following properties.

(1.25) Axioms for Principal Components

a. $\mathbb{B}$ forms a basis for $\mathbb{R}^n$.
b. Each element of $\mathbb{B}$ has unit length: for all i, $\|\boldsymbol{\psi}^{(i)}\| = 1$.
c. The elements of $\mathbb{B}$ are pairwise orthogonal: for all i and j where $i \neq j$, $(\boldsymbol{\psi}^{(i)})^\top(\boldsymbol{\psi}^{(j)}) = 0$. (Geometrically, this means that $\boldsymbol{\psi}^{(i)}$ is perpendicular to $\boldsymbol{\psi}^{(j)}$ in $\mathbb{R}^n$.)
d. When $\boldsymbol{X}$ is re-written using the coordinates of $\mathbb{B}$, the $\boldsymbol{\psi}^{(1)}$ direction contains the greatest possible variation of values; and for $i > 1$, the $\boldsymbol{\psi}^{(i)}$ direction contains the greatest possible variation of values subject to the constraint that $\boldsymbol{\psi}^{(i)}$ must be orthogonal to $\boldsymbol{\psi}^{(j)}$ for all $j < i$.

The elements of $\mathbb{B}$ are the principal components of $\boldsymbol{X}$; for each i, $\boldsymbol{\psi}^{(i)}$ is the *ith principal component of* $\boldsymbol{X}$. When conditions (1.25a–c) are satisfied, we refer to $\mathbb{B}$ as an *orthonormal basis for* $\mathbb{R}^n$. Axiom (1.25d), for now, is stated in a hand-wavy manner. To define the PCA algorithm, we will have to first define the concept of "variance," and then figure out how to find vectors that maximize variance.

The Covariance Matrix. The first step of PCA is to define a *covariance matrix* that measures the variance represented by the data of $\boldsymbol{X}$. The notion of "variance" we use here is based on statistical variance, from probability theory. Recall that the variance of a scalar-valued random variable is defined as follows.

(1.26) DEFINITION

Let a and b be scalar-valued random variables. The *covariance of* a *and* b is the quantity

$$\text{cov}(\text{a}, \text{b}) = \mathbb{E}[(\text{a} - \mathbb{E}[\text{a}])(\text{b} - \mathbb{E}[\text{b}])].$$

The *variance of* a is defined as $\text{var}(\text{a}) = \text{cov}(\text{a}, \text{a})$.

We extend the notion of covariance to vector-valued random variables by treating each position of each vector as a separate variable. The covariance is defined as a matrix containing the covariance between each pair of vector entries.

(1.27) DEFINITION

Let $\mathbf{a}$ and $\mathbf{b}$ be n-dimensional vector-valued random variables. The *covariance of* $\mathbf{a}$ *and* $\mathbf{b}$ is the matrix

$$\text{cov}(\mathbf{a}, \mathbf{b}) = \mathbb{E}[(\mathbf{a} - \mathbb{E}[\mathbf{a}])(\mathbf{b} - \mathbb{E}[\mathbf{b}])^\top] \in \mathbb{R}^{n \times n}.$$

Notice that $\text{cov}(\mathbf{a}, \mathbf{b})_{i,j} = \text{cov}(\mathrm{a}_i, \mathrm{b}_j)$ for all positions i and j.

Since the covariance between the two random variables is a matrix, there is no scalar value that represents the "variance" of a vector-valued random variable. Instead, the vector-valued analogue of variance is simply the covariance of a random variable $\mathbf{a}$ with itself: $\text{cov}(\mathbf{a}, \mathbf{a})$.

The covariance matrix for $\boldsymbol{X}$ is defined by pretending that $\boldsymbol{X}$ is a data sample that estimates the distribution of some n-dimensional random variable $\mathbf{x}$. The expected value of $\mathbf{x}$ is estimated using the mean of the sample, denoted by $\overline{\boldsymbol{X}}$:

$$\mathbb{E}[\mathbf{x}] = \overline{\boldsymbol{X}}^\top = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{X}_{i,:})^\top = \frac{\boldsymbol{X}^\top \mathbf{1}}{m}.$$

We then estimate $\text{cov}(\mathbf{x}, \mathbf{x})$ by plugging in the above expression into the formula for covariance:

$$\text{cov}(\mathbf{x}, \mathbf{x}) = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^\top] = \mathbb{E}[(\mathbf{x} - \overline{\boldsymbol{X}}^\top)(\mathbf{x} - \overline{\boldsymbol{X}}^\top)^\top]$$

and then using the sample mean of $(\mathbf{x} - \overline{\boldsymbol{X}}^\top)(\mathbf{x} - \overline{\boldsymbol{X}}^\top)^\top$ to estimate the expected value:

$$\mathbb{E}[(\mathbf{x} - \overline{\boldsymbol{X}}^\top)(\mathbf{x} - \overline{\boldsymbol{X}}^\top)^\top] = \frac{1}{m} \sum_{i=1}^{m} \left((\boldsymbol{X}_{i,:})^\top - \overline{\boldsymbol{X}}^\top \right) (\boldsymbol{X}_{i,:} - \overline{\boldsymbol{X}}) = \frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})}{m}.$$

We now use this formula to define the covariance matrix.

(1.28) DEFINITION

Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ be a matrix. The *covariance matrix of* $\boldsymbol{X}$ is the $n \times n$ matrix

$$\text{cov}(\boldsymbol{X}) = \frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})}{m},$$

where $\overline{\boldsymbol{X}} = \mathbf{1}^\top \boldsymbol{X}/m$.

(1.29) EXERCISE

Prove that the last part of the derivation is valid: using the definition of matrix multiplication, convince yourself that

$$\frac{1}{m} \sum_{i=1}^{m} \left((\boldsymbol{X}_{i,:})^\top - \overline{\boldsymbol{X}}^\top \right) (\boldsymbol{X}_{i,:} - \overline{\boldsymbol{X}}) = \frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})}{m}.$$

Projecting onto a Basis. When we find our orthonormal basis $\mathbb{B}$ containing the principal components of $\boldsymbol{X}$, we will want to re-write the rows of $\boldsymbol{X}$ in terms of coordinates with respect to $\mathbb{B}$. The coordinates of a vector $\boldsymbol{v} \in \mathbb{R}^n$ with respect to $\mathbb{B}$ are given by a vector $\boldsymbol{a} \in \mathbb{R}^n$ such that

$$\boldsymbol{v} = a_1\boldsymbol{\psi}^{(1)} + a_2\boldsymbol{\psi}^{(2)} + \cdots + a_n\boldsymbol{\psi}^{(n)} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\psi}^{(1)} & \boldsymbol{\psi}^{(2)} & \dots & \boldsymbol{\psi}^{(n)} \\ | & | & & | \end{bmatrix} \boldsymbol{a}.$$

By solving for $\boldsymbol{a}$, we obtain the following formula for re-writing $\boldsymbol{v}$ in terms of $\mathbb{B}$:

$$\boldsymbol{a} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\psi}^{(1)} & \boldsymbol{\psi}^{(2)} & \dots & \boldsymbol{\psi}^{(n)} \\ | & | & & | \end{bmatrix}^{-1} \boldsymbol{v};$$

in other words, to find $\boldsymbol{a}$, we simply multiply $\boldsymbol{v}$ on the left by the *change of basis matrix* for $\mathbb{B}$.

(1.30) DEFINITION

Let

$$\mathbb{B} = \{\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \dots, \boldsymbol{b}^{(n)}\}$$

be a basis for $\mathbb{R}^n$. The *change of basis matrix for* $\mathbb{B}$ is the matrix

$$\boldsymbol{B} = \begin{bmatrix} | & | & & | \\ \boldsymbol{b}^{(1)} & \boldsymbol{b}^{(2)} & \dots & \boldsymbol{b}^{(n)} \\ | & | & & | \end{bmatrix}^{-1}.$$

For a vector $\boldsymbol{v} \in \mathbb{R}^n$, the vector $\boldsymbol{Bv}$ is the *coordinates of* $\boldsymbol{v}$ *with respect to* $\mathbb{B}$.

Notice that computing the change of basis matrix requires inverting a matrix. This is a nasty operation: inverses don't always exist, and when they do, they can be computationally expensive to find. Thankfully, however, our assumption that $\mathbb{B}$ is orthonormal will make the inverse operation easy to compute.

(1.31) THEOREM

Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a square matrix. Then,

a. each column/row of $\boldsymbol{A}$ has unit length and

b. the columns/rows of $\boldsymbol{A}$ are pairwise orthogonal

if and only if $\boldsymbol{A}^{-1} = \boldsymbol{A}^\top$.

Proof. Suppose conditions a and b are satisfied by the rows of $\boldsymbol{A}$. Then, observe that

(1.32)

$$(\boldsymbol{A}\boldsymbol{A}^\top)_{i,j} = \boldsymbol{A}_{i,:}(\boldsymbol{A}_{j,:})^\top = \begin{cases} 1, & i = j \\ 0, & i \neq j, \end{cases}$$

thus $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}_n$, so $\boldsymbol{A}^\top = \boldsymbol{A}^{-1}$. Similarly, if a and b are satisfied by the columns of $\boldsymbol{A}$, then

(1.33)
$$(\boldsymbol{A}^\top \boldsymbol{A})_{i,j} = (\boldsymbol{A}_{:,i})^\top \boldsymbol{A}_{:,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j, \end{cases}$$

so again $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}_n$, thus $\boldsymbol{A}^\top = \boldsymbol{A}^{-1}$.

Now, suppose $\boldsymbol{A}^{-1} = \boldsymbol{A}^\top$. Then, $\boldsymbol{I}_n = \boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{A}^\top\boldsymbol{A}$, so equations (1.32) and (1.33) are both satisfied, which in turn implies that conditions a and b are satisfied by both the rows and columns of $\boldsymbol{A}$. □

Based on this property, the change of basis matrix for $\mathbb{B}$ is simply

$$\boldsymbol{B} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\psi}^{(1)} & \boldsymbol{\psi}^{(2)} & \cdots & \boldsymbol{\psi}^{(n)} \\ | & | & & | \end{bmatrix}^{-1} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\psi}^{(1)} & \boldsymbol{\psi}^{(2)} & \cdots & \boldsymbol{\psi}^{(n)} \\ | & | & & | \end{bmatrix}^\top = \begin{bmatrix} — & \boldsymbol{\psi}^{(1)} & — \\ — & \boldsymbol{\psi}^{(2)} & — \\ & \vdots & \\ — & \boldsymbol{\psi}^{(n)} & — \end{bmatrix}.$$

The Direction of Greatest Variance. When we re-write $\boldsymbol{X}$ using $\mathbb{B}$'s coordinates, the resulting data matrix is $(\boldsymbol{B}\boldsymbol{X}^\top)^\top = \boldsymbol{X}\boldsymbol{B}^\top$. For each i, the m-dimensional vector $\boldsymbol{X}\boldsymbol{\psi}^{(i)}$ contains the component in the $\boldsymbol{\psi}^{(i)}$ direction for each item of $\boldsymbol{X}$.

Observe that since $\boldsymbol{X}\boldsymbol{v}$ is a column vector as long as $\boldsymbol{v} \in \mathbb{R}^n$, the covariance matrix $\text{cov}(\boldsymbol{X}\boldsymbol{v})$ contains only one row and one column. We can therefore treat $\text{cov}(\boldsymbol{X}\boldsymbol{v})$ as a scalar value, allowing us to define $\boldsymbol{\psi}^{(1)}$ as the vector $\boldsymbol{v}$ that maximizes $\text{cov}(\boldsymbol{X}\boldsymbol{v})$ subject to the constraint that $\boldsymbol{v} \neq \boldsymbol{0}$:

$$\boldsymbol{\psi}^{(1)} = \underset{\boldsymbol{v}}{\text{argmax}}\, \text{cov}(\boldsymbol{X}\boldsymbol{v}) \text{ subject to } \boldsymbol{v} \neq \boldsymbol{0}.$$

For the other principal components, we do the same thing, but add in a constraint that each principal component must be orthogonal to the previous principal components:

$$\boldsymbol{\psi}^{(i)} = \underset{\boldsymbol{v}}{\text{argmax}}\, \text{cov}(\boldsymbol{X}\boldsymbol{v}) \text{ subject to } \boldsymbol{v} \neq \boldsymbol{0} \text{ and } \forall j < i [\boldsymbol{v}^\top \boldsymbol{\psi}^{(j)} = 0].$$

Our goal is now to solve these optimization problems.

The Spectral Theorem. Let's begin by doing some algebra to figure out the relationship between $\text{cov}(\boldsymbol{X}\boldsymbol{v})$ and $\text{cov}(\boldsymbol{X})$. First, we calculate the mean $\overline{\boldsymbol{X}\boldsymbol{v}}$ as follows:

$$\overline{\boldsymbol{X}\boldsymbol{v}} = \frac{\boldsymbol{1}^\top(\boldsymbol{X}\boldsymbol{v})}{m} = \frac{(\boldsymbol{1}^\top\boldsymbol{X})\boldsymbol{v}}{m} = \overline{\boldsymbol{X}}\boldsymbol{v}.$$

Plugging this into the definition of the covariance matrix, we get

$$\begin{aligned}
\operatorname{cov}(\boldsymbol{X}\boldsymbol{v}) &= \frac{(\boldsymbol{X}\boldsymbol{v} - \overline{\boldsymbol{X}\boldsymbol{v}})^\top (\boldsymbol{X}\boldsymbol{v} - \overline{\boldsymbol{X}\boldsymbol{v}})}{m} \\
&= \frac{(\boldsymbol{X}\boldsymbol{v} - \overline{\boldsymbol{X}}\boldsymbol{v})^\top (\boldsymbol{X}\boldsymbol{v} - \overline{\boldsymbol{X}}\boldsymbol{v})}{m} \\
&= \frac{(\boldsymbol{v}^\top \boldsymbol{X}^\top - \boldsymbol{v}^\top \overline{\boldsymbol{X}}^\top)(\boldsymbol{X}\boldsymbol{v} - \overline{\boldsymbol{X}}\boldsymbol{v})}{m} \\
&= \frac{\boldsymbol{v}^\top (\boldsymbol{X}^\top - \overline{\boldsymbol{X}}^\top)(\boldsymbol{X} - \overline{\boldsymbol{X}})\boldsymbol{v}}{m} \\
&= \frac{\boldsymbol{v}^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})\boldsymbol{v}}{m} \\
&= \boldsymbol{v}^\top \operatorname{cov}(\boldsymbol{X})\boldsymbol{v}.
\end{aligned}$$

Now, notice that $\operatorname{cov}(\boldsymbol{X})$ is *symmetric*; that is to say, $\operatorname{cov}(\boldsymbol{X}) = \operatorname{cov}(\boldsymbol{X})^\top$:

$$\operatorname{cov}(\boldsymbol{X})^\top = \left(\frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})}{m}\right)^\top = \frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})^\top \left((\boldsymbol{X} - \overline{\boldsymbol{X}})^\top\right)^\top}{m} = \frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})^\top (\boldsymbol{X} - \overline{\boldsymbol{X}})}{m} = \operatorname{cov}(\boldsymbol{X}).$$

The *Spectral Theorem* says that when a matrix $\boldsymbol{A}$ is symmetric, the maximum value of $\boldsymbol{v}^\top \boldsymbol{A}\boldsymbol{v}$ is always an *eigenvalue* of $\boldsymbol{A}$, and the vector $\boldsymbol{v}$ that achieves this maximum value is always an *eigenvector* corresponding to that eigenvalue. Below we review the definition of eigenvalues and eigenvectors, and then give the statement of the Spectral Theorem.

(1.34) DEFINITION

Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a square matrix. For $\boldsymbol{v} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$, we say that $\boldsymbol{v}$ *is an eigenvalue of* $\boldsymbol{A}$ *with eigenvector* λ if

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}.$$

(1.35) THEOREM (Spectral Theorem)

Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then, there is an orthonormal basis $\{\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(n)}\}$ for $\mathbb{R}^n$ that consists entirely of eigenvectors of $\boldsymbol{A}$, with associated eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, respectively. Assuming without loss of generality that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, we have

$$\left(\boldsymbol{v}^{(1)}\right)^\top \boldsymbol{A}\boldsymbol{v}^{(1)} = \lambda_1 = \max_{\boldsymbol{v}} \boldsymbol{v}^\top \boldsymbol{A}\boldsymbol{v} \text{ subject to } \boldsymbol{v} \neq \boldsymbol{0},$$

and for $i > 1$ we have

$$\left(\boldsymbol{v}^{(i)}\right)^\top \boldsymbol{A}\boldsymbol{v}^{(i)} = \lambda_i = \max_{\boldsymbol{v}} \boldsymbol{v}^\top \boldsymbol{A}\boldsymbol{v} \text{ subject to } \boldsymbol{v} \neq \boldsymbol{0} \text{ and } \forall\, j < i\left[\boldsymbol{v}^\top \boldsymbol{v}^{(j)} = 0\right].$$

Applying the Spectral Theorem to $\operatorname{cov}(\boldsymbol{X})$, we discover that the direction of greatest variance—that is, the unit vector that maximizes $\operatorname{cov}(\boldsymbol{X}\boldsymbol{v}) = \boldsymbol{v}^\top \operatorname{cov}(\boldsymbol{X})\boldsymbol{v}$—is the unit-length eigenvector for $\boldsymbol{X}$ with the greatest corresponding eigenvalue. Therefore, our principal components $\mathbb{B}$ are simply an orthonormal set of eigenvectors of $\boldsymbol{A}$, where the ith principal component $\boldsymbol{\psi}^{(i)}$ is the eigenvector with the ith greatest eigenvalue.

Finding the Eigenvectors. The final piece of the PCA algorithm is to find a way to compute the eigenvectors of the covariance matrix. Typically, this is achieved using a technique called *singular value decomposition* (SVD).

(1.36) THEOREM (Singular Value Decomposition)
Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be a matrix, not necessarily square. Then, there exist matrices $\boldsymbol{U} \in \mathbb{R}^{m \times m}$, $\boldsymbol{S} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$$

and the following hold.

a. *The columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are pairwise orthogonal and of unit length.*
b. *$\boldsymbol{S}$ is diagonal: $S_{i,j} = 0$ whenever $i \neq j$.*
c. *The diagonal entries of $\boldsymbol{S}$ are in decreasing order: if $i < j \leq \min(m, n)$, then $S_{i,i} \geq S_{j,j}$.*
d. *Each column of $\boldsymbol{V}$ is an eigenvector of $\boldsymbol{A}^\top\boldsymbol{A}$. The eigenvalue corresponding to the ith column of $\boldsymbol{V}$ if $i \leq \min(m, n)$ and 0 otherwise.*
e. *Each column of $\boldsymbol{U}$ is an eigenvector of $\boldsymbol{A}\boldsymbol{A}^\top$. The eigenvalue corresponding to the ith column of $\boldsymbol{U}$ is $S_{i,i}^2$ if $i \leq \min(m, n)$ and 0 otherwise.*

There are already algorithms for computing the SVD of a matrix. For PCA, we apply SVD to the matrix $\boldsymbol{A}$ defined as follows:

$$\boldsymbol{A} = \frac{\boldsymbol{X} - \overline{\boldsymbol{X}}}{\sqrt{m}}.$$

Observe that $\text{cov}(\boldsymbol{X}) = \boldsymbol{A}^\top\boldsymbol{A}$. If $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ is the SVD of $\boldsymbol{A}$, then the columns of $\boldsymbol{V}$ contain pairwise-orthogonal unit eigenvectors of $\text{cov}(\boldsymbol{X})$, sorted in decreasing order of their associated eigenvalues. This means that the ith column of $\boldsymbol{V}$ is in fact the ith principal component of $\boldsymbol{X}$, which makes $\boldsymbol{V}^\top$ the change of basis matrix for $\mathbb{B}$.

PCA Procedure. To summarize, the full procedure for embedding vectors into a lower-dimensional space using PCA is as follows.

ALGORITHM: DIMENSIONALITY REDUCTION VIA PRINCIPAL COMPONENTS ANALYSIS

Inputs:

- $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, a set of feature vectors arranged into a matrix of row vectors
- k, the desired dimensionality of the embedding vectors

Output:

- $\boldsymbol{X}' \in \mathbb{R}^{m \times k}$, an embedding of $\boldsymbol{X}$ into $\mathbb{R}^k$

Procedure:

- Compute the mean of $\boldsymbol{X}$:

$$\overline{\boldsymbol{X}} \leftarrow \frac{\boldsymbol{X}^\top\mathbf{1}}{m}.$$

- Compute the "square root" of the covariance matrix of $\boldsymbol{X}$:

$$\boldsymbol{A} \leftarrow \frac{(\boldsymbol{X} - \overline{\boldsymbol{X}})}{\sqrt{m}}.$$

- Compute the SVD of $\boldsymbol{A}$: let $\boldsymbol{U}$, $\boldsymbol{S}$, and $\boldsymbol{V}$ be such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$$

  is the SVD of $\boldsymbol{A}$.

- Truncate $\boldsymbol{V}$ to include only the first k principal components of $\boldsymbol{X}$:

$$\boldsymbol{V} \leftarrow \boldsymbol{V}_{:,:k}.$$

- Project $\boldsymbol{X}$ onto its first k principal components using $\boldsymbol{V}^\top$ as a change of basis matrix:

$$\boldsymbol{X}' \leftarrow \boldsymbol{X}\boldsymbol{V}.$$

- Return $\boldsymbol{X}'$.

1.5.2 t-Distributed Stochastic Neighbors Embedding

The t-SNE algorithm embeds a finite set $\mathbb{D} \subseteq \mathbb{R}^n$ of feature vectors into a lower-dimensional space $\mathbb{R}^k$ in a way that is isometric with respect to two similarity metrics, $d_1 : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and $d_2 : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$. These two similarity metrics are based on the probability that two randomly chosen points are "neighbors." In the high-dimensional space, d_1 is based on a normal distribution, while in the low-dimensional space, d_2 is based on the Student's t distribution with 1 degree of freedom. The embedding is computed by viewing d_1 and d_2 as probability distributions and maximizing their similarity.

High-Dimensional Similarity. Fix a point $\boldsymbol{x} \in \mathbb{D}$. Imagine that we want to randomly choose a "neighbor" $\boldsymbol{y}$ of $\boldsymbol{x}$, in a way such that the Euclidean distance $\|\boldsymbol{y} - \boldsymbol{x}\|$ between $\boldsymbol{y}$ and $\boldsymbol{x}$ is normally distributed with standard deviation $\sigma_{\boldsymbol{x}}$. The probability density function for $\|\boldsymbol{y} - \boldsymbol{x}\|$ is given by

$$p_{\boldsymbol{x}}(\boldsymbol{y}) = \frac{1}{\sigma_{\boldsymbol{x}}\sqrt{2\pi}} e^{-\|\boldsymbol{y}-\boldsymbol{x}\|^2/(2\sigma_{\boldsymbol{x}}^2)}.$$[14]

Observe that points closer to $\boldsymbol{x}$, which are more likely to be considered "neighbors" of $\boldsymbol{x}$, have a higher density than points that are farther away from $\boldsymbol{x}$.

Now, imagine that we are choosing a random point of $\mathbb{D}$, not including $\boldsymbol{x}$, to be a "neighbor" of $\boldsymbol{x}$. We want this random choice to be distributed according to our normal distribution. However,

[14] Technically, this implies that $\|\boldsymbol{y} - \boldsymbol{x}\|$ can be negative; we won't think about this too much.

the normal distribution is a continuous distribution, while sampling from $\mathbb{D}$ requires a discrete distribution. In order to turn our normal distribution into a discrete distribution, we convert $p_{\boldsymbol{x}}$ into a probability mass function by normalizing over $\mathbb{D}\backslash\{\boldsymbol{x}\}$:

$$\mathbb{P}[\boldsymbol{y} \in \mathbb{D}\backslash\{\boldsymbol{x} \text{ is chosen as a "neighbor" of } \boldsymbol{x}] = p(\boldsymbol{y}, \boldsymbol{x}) = \frac{p_{\boldsymbol{x}}(\boldsymbol{y})}{\sum_{\boldsymbol{z} \in \mathbb{D}\backslash\{\boldsymbol{x}\}} p_{\boldsymbol{x}}(\boldsymbol{z})}.$$

Our similarity metric on $\mathbb{R}^n$ is defines the similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$ as the average of $p(\boldsymbol{x}, \boldsymbol{y})$ and $p(\boldsymbol{y}, \boldsymbol{x})$, scaled by the cardinality of $\mathbb{D}$:

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \frac{p(\boldsymbol{x}, \boldsymbol{y}) + p(\boldsymbol{y}, \boldsymbol{x})}{2|\mathbb{D}|}.$$

Assuming that $d(\boldsymbol{x}, \boldsymbol{y}) = 0$ when $\boldsymbol{x} = \boldsymbol{y}$, observe that d_1 is a probability mass function over $\mathbb{D} \times \mathbb{D}$:

$$\sum_{\boldsymbol{x}, \boldsymbol{y} \in \mathbb{D}} d_1(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2|\mathbb{D}|}\left(\left(\sum_{\boldsymbol{x}, \boldsymbol{y} \in \mathbb{D}} p(\boldsymbol{x}, \boldsymbol{y})\right) + \left(\sum_{\boldsymbol{x}, \boldsymbol{y} \in \mathbb{D}} p(\boldsymbol{y}, \boldsymbol{x})\right)\right) = \frac{1}{2|\mathbb{D}|}(|\mathbb{D}| + |\mathbb{D}|) = 1.$$

Low-Dimensional Similiarity. To define d_2, we do essentially the same thing, except we use a Student's t-distribution with 1 degree of freedom centered around $\boldsymbol{x}$. The probability density function for this distribution is given by

$$q_{\boldsymbol{x}}(\boldsymbol{y}) = \frac{1}{\pi(1 + \|\boldsymbol{y} - \boldsymbol{x}\|^2)}.$$

Unlike $p_{\boldsymbol{x}}$, $q_{\boldsymbol{x}}$ is "symmetric" in the sense that $q_{\boldsymbol{x}}(\boldsymbol{y}) = q_{\boldsymbol{y}}(\boldsymbol{x})$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Therefore, rather than viewing $q_{\boldsymbol{x}}(\boldsymbol{y})$ as the process of choosing a neighbor $\boldsymbol{y}$ for a pre-determined point $\boldsymbol{x}$, we can view $q_{\boldsymbol{x}}(\boldsymbol{y})$ as simultaneously choosing two points, $\boldsymbol{x}$ and $\boldsymbol{y}$, to be neighbors. Thus, given a set of embeddings $\mathbb{E}$, we define a probability mass function over $\mathbb{E}$ by

$$\mathbb{P}[\boldsymbol{x} \text{ and } \boldsymbol{y} \text{ are "neighbors"}] = q(\boldsymbol{x}, \boldsymbol{y}) = \frac{q_{\boldsymbol{x}}(\boldsymbol{y})}{\sum_{\boldsymbol{u}, \boldsymbol{v} \in \mathbb{E}, \boldsymbol{u} \neq \boldsymbol{v}} q_{\boldsymbol{u}}(\boldsymbol{v})},$$

and we simply set $d_2(\boldsymbol{x}, \boldsymbol{y}) = q(\boldsymbol{x}, \boldsymbol{y})$ when $\boldsymbol{x} \neq \boldsymbol{y}$ and $d_2(\boldsymbol{x}, \boldsymbol{y}) = 0$ otherwise. Note that d_2 is also a valid probability mass function.

Finding an Optimal Embedding. Let $f^* : \mathbb{R}^n \to \mathbb{R}^k$ be our t-SNE embedding. We would like f^* to embed $\mathbb{D}$ as isometrically as possible with respect to d_1 and d_2. The degree of isometricity is measured by the *Kullback–Leibler divergence* between the probability distribution d_1 and the probability distribution defined by d_2.

(1.37) DEFINITION

Let $p, q : \mathbb{A} \to [0, 1]$ be probability mass functions over a set $\mathbb{A}$. The *Kullback–Leibler divergence of p from q*, or *KL-divergence* for short, is defined as

$$\mathrm{KL}(p \| q) = \sum_{a \in \mathbb{A}} p(a) \ln\left(\frac{p(a)}{q(a)}\right).$$

Note that the definition of d_2 depends on how $\mathbb{D}$ is embedded into $\mathbb{R}^k$. Therefore, different embeddings $f : \mathbb{R}^n \to \mathbb{R}^k$ will result in different definitions of d_2. We define the *error* of an embedding f to be the KL-divergence between d_1 and the version of d_2 given by f:

$$E(f) = \sum_{\boldsymbol{x},\boldsymbol{y}\in\mathbb{D}} d_1(\boldsymbol{x},\boldsymbol{y}) \ln\left(\frac{d_1(\boldsymbol{x},\boldsymbol{y})}{d_2(f(\boldsymbol{x}),f(\boldsymbol{y}))}\right).$$

We then compute our t-SNE embedding by minimizing $E(f)$:

$$f^* = \operatorname*{argmin}_f E(f) = \operatorname*{argmin}_f \sum_{\boldsymbol{x},\boldsymbol{y}\in\mathbb{D}} d_1(\boldsymbol{x},\boldsymbol{y}) \ln\left(\frac{d_1(\boldsymbol{x},\boldsymbol{y})}{d_2(f(\boldsymbol{x}),f(\boldsymbol{y}))}\right).$$

Bibliography

Leonard Bloomfield. 1933. *Language*, first edition. Henry Holt and Company, New York, NY.

J. R. Firth. 1957. A Synopsis of Linguistic Theory, 1930– 1955. In *Studies in Linguistic Analysis*, first edition, Special Volume of the Philological Society, pages 1–31. Blackwell Publishers, Oxford, United Kingdom.

Yoav Goldberg and Omer Levy. 2014. Word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. *Computing Research Repository*, arXiv:1402.3722.

Thomas Kollar, Danielle Berry, Lauren Stuart, Karolina Owczarzak, Tagyoung Chung, Lambert Mathias, Michael Kayser, Bradford Snow, and Spyros Matsoukas. 2018. The Alexa Meaning Representation Language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 3 (industry papers), pages 177–184, New Orleans, LA. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *ICLR 2013 Workshop Proceedings*, Scottsdale, AZ. OpenReview.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, Montreal, Canada. Curran Associates, Inc.