Word Vectors (word2vec)

Ling 282/482: Deep Learning for Computational Linguistics

C.M. Downey

Fall 2025

From Numbers to Language

From Numbers to Language

- In past lectures:
  - Vectors, spaces, linear transformations between spaces
  - Perceptron models, which learn to linearly separate input vectors
  - Gradient Descent, Backpropagation algorithms to learn ideal weights

From Numbers to Language

- In past lectures:
  - Vectors, spaces, linear transformations between spaces
  - Perceptron models, which learn to linearly separate input vectors
  - Gradient Descent, Backpropagation algorithms to learn ideal weights
- Starting today:
  - How do we use these tools to **represent language?**
  - First step: **word vectors** (representing language in **high-dimensional space**)
  - Algorithms like **word2vec** learn these representations based on data

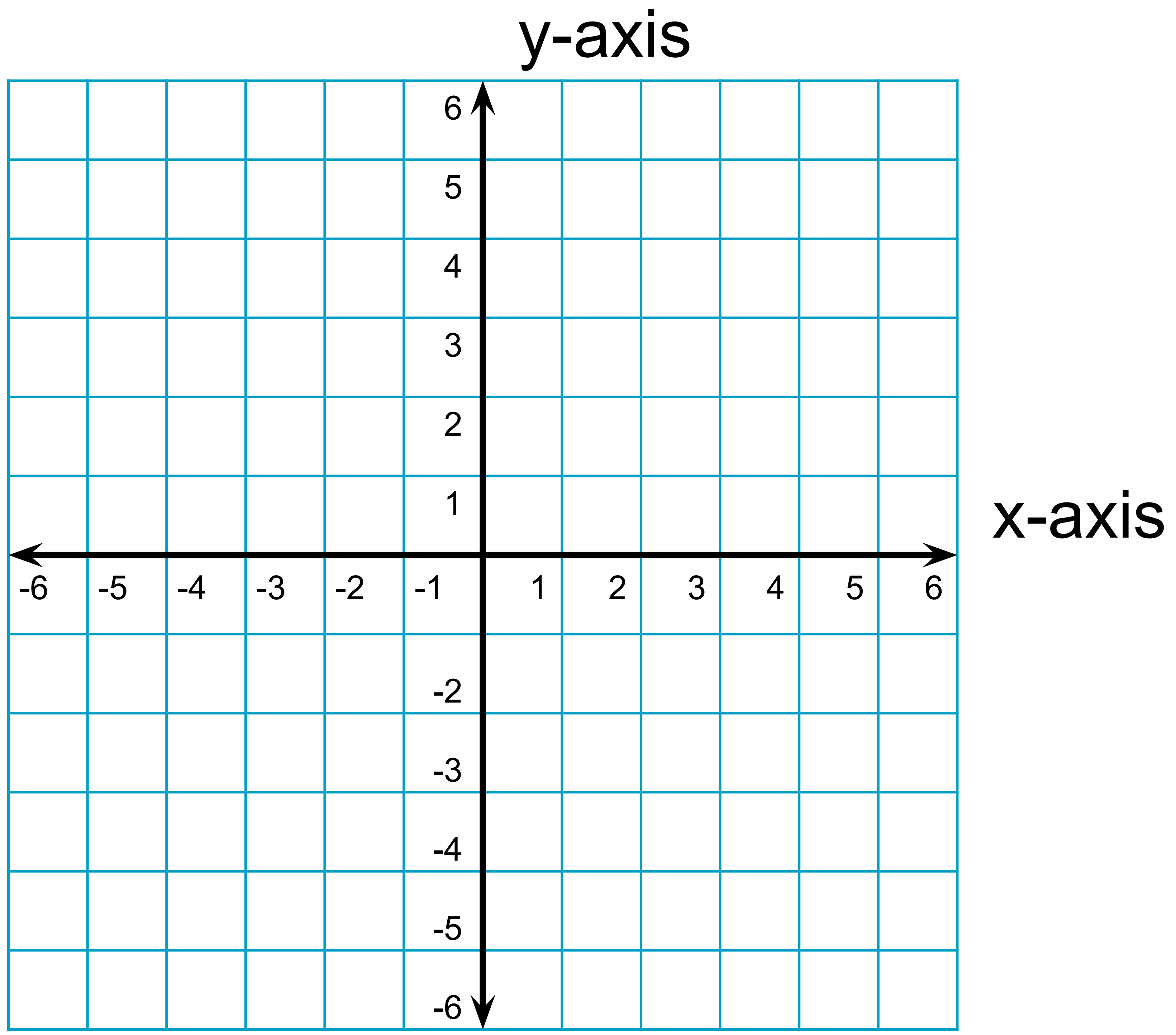
Word Vectors, Intro

Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a **dimension**

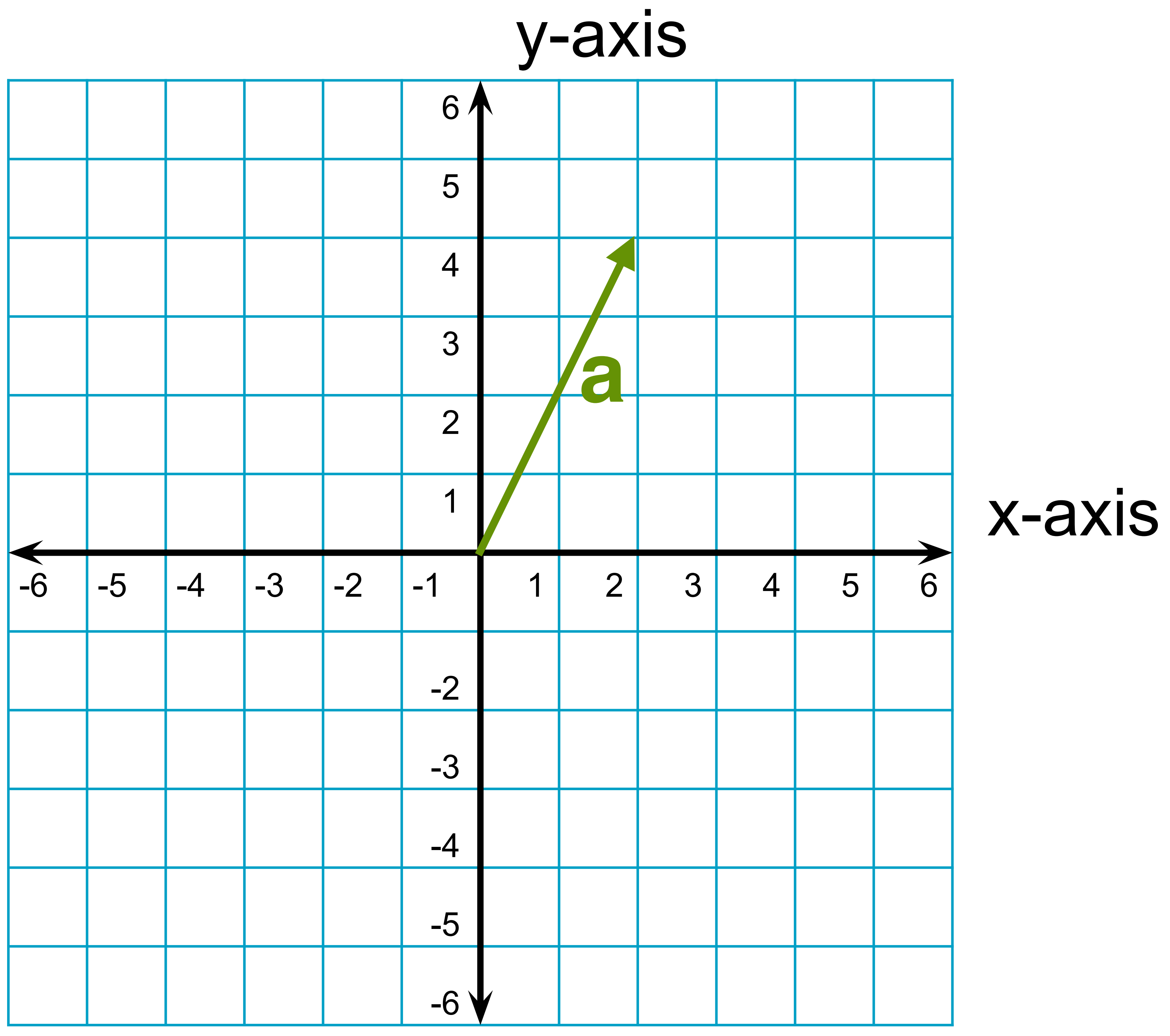
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a **dimension**



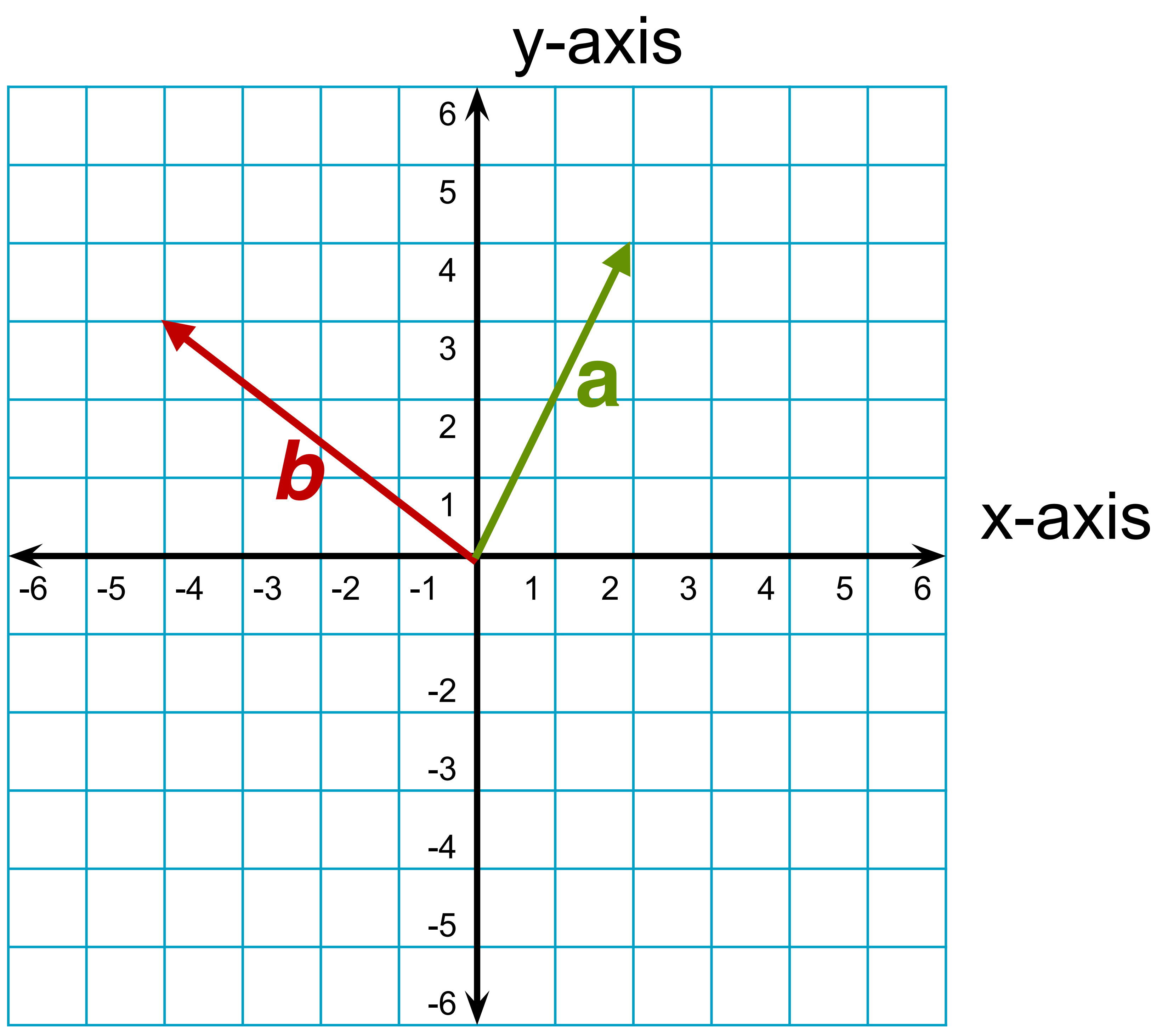
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a **dimension**
  - $\vec{a} = \langle 2,4 \rangle$



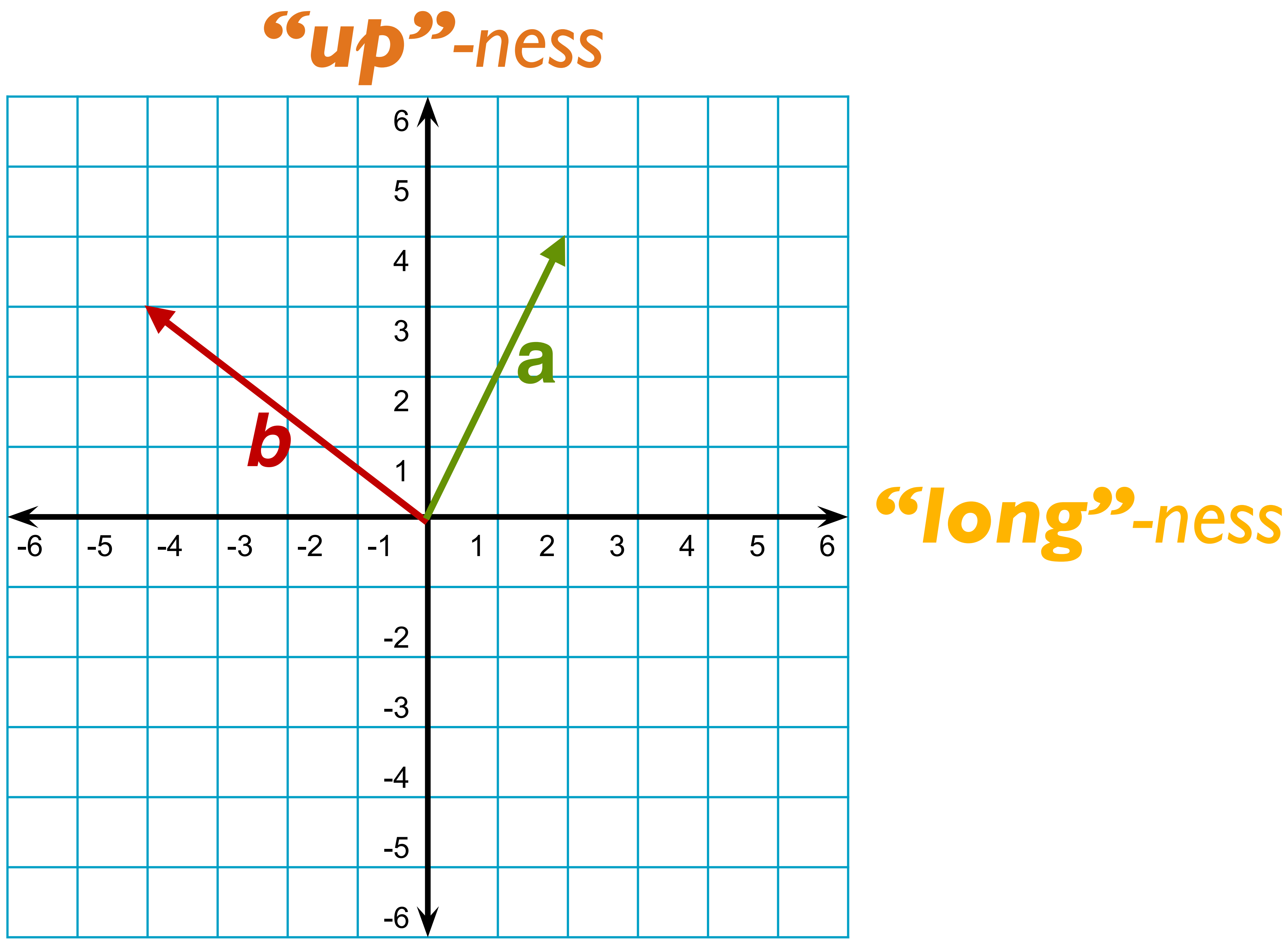
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a **dimension**
    - $\vec{a} = \langle 2,4 \rangle$
    - $\vec{b} = \langle -4,3 \rangle$



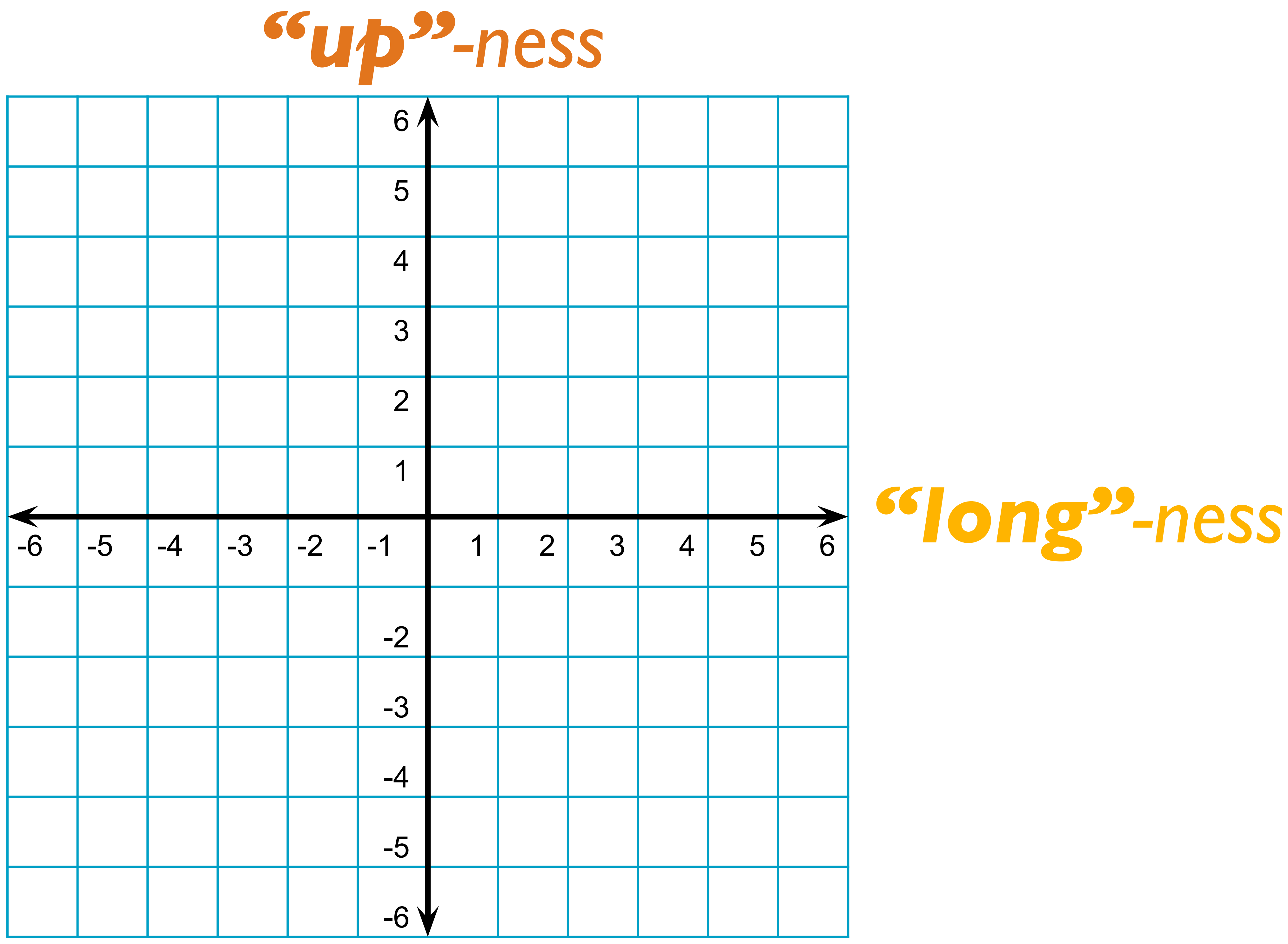
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a “dimension”
  - $\vec{a} = \langle 2,4 \rangle$
  - $\vec{b} = \langle -4,3 \rangle$



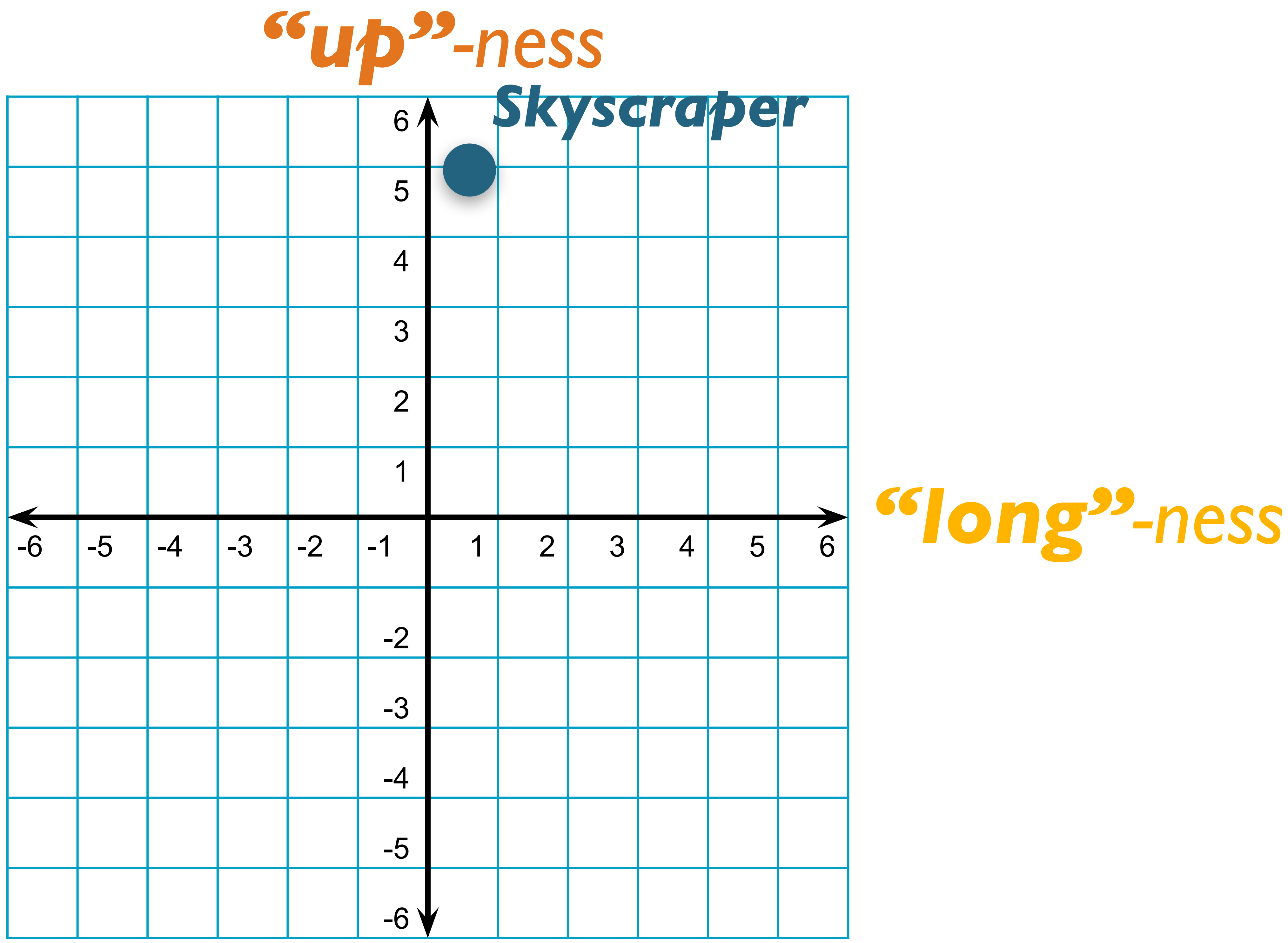
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a “dimension”
    - $\vec{a}$ = ⟨2,4⟩
    - $\vec{b}$ = ⟨-4,3⟩



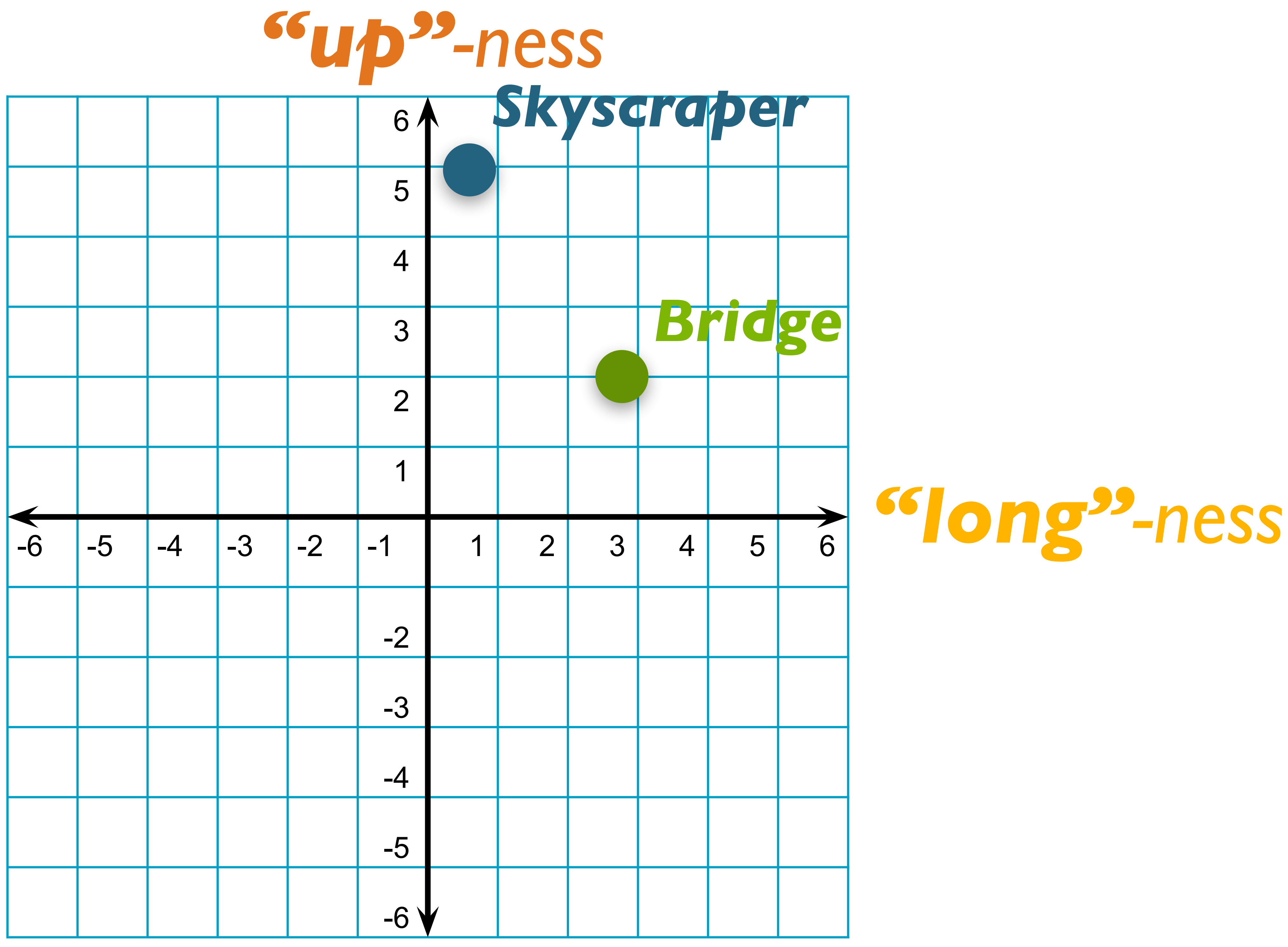
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a "dimension"
  - $\vec{a} = \langle 2,4 \rangle$
  - $\vec{b} = \langle -4,3 \rangle$



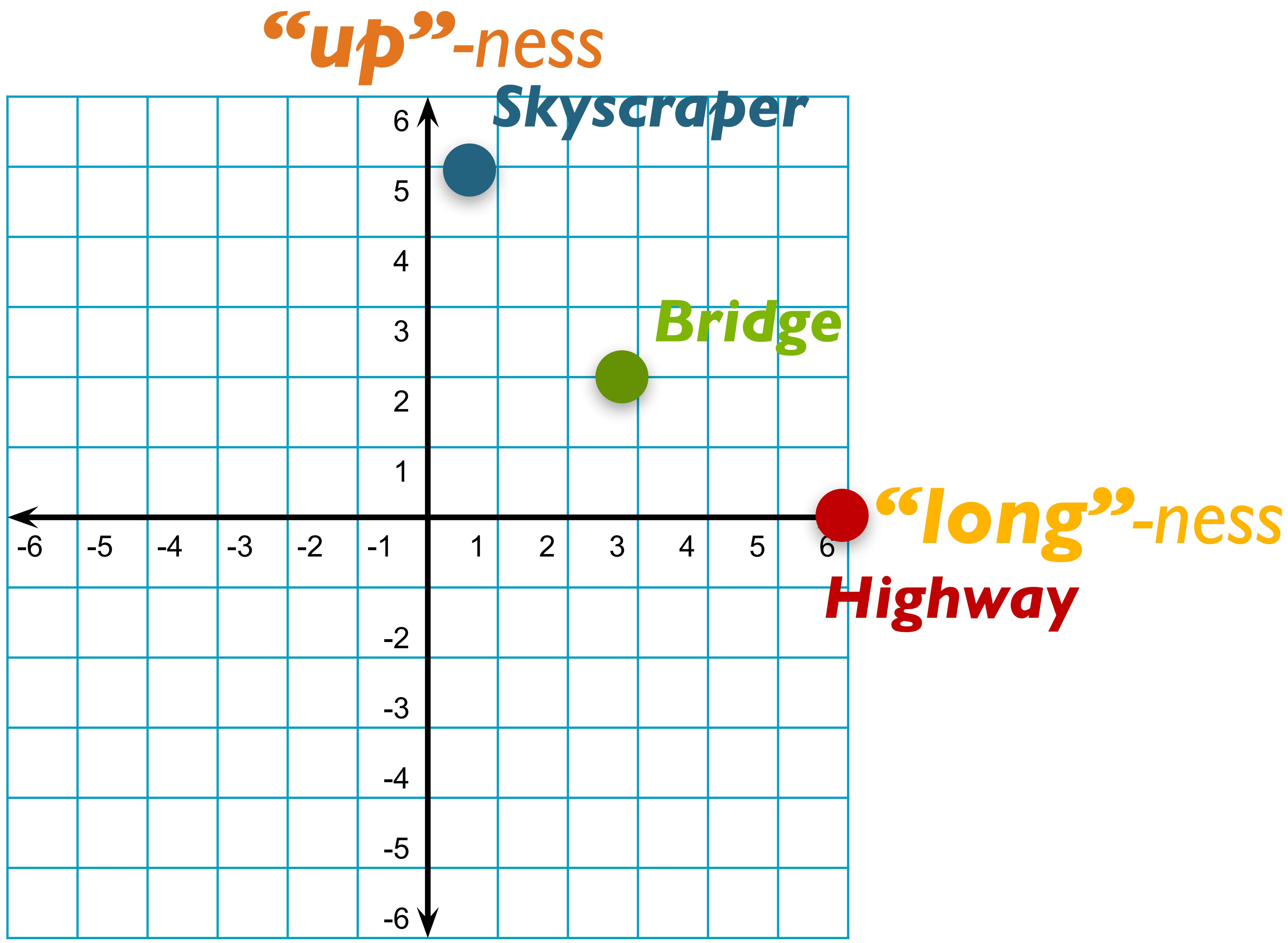
Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a "dimension"
  - $\vec{a} = \langle 2,4 \rangle$
  - $\vec{b} = \langle -4,3 \rangle$

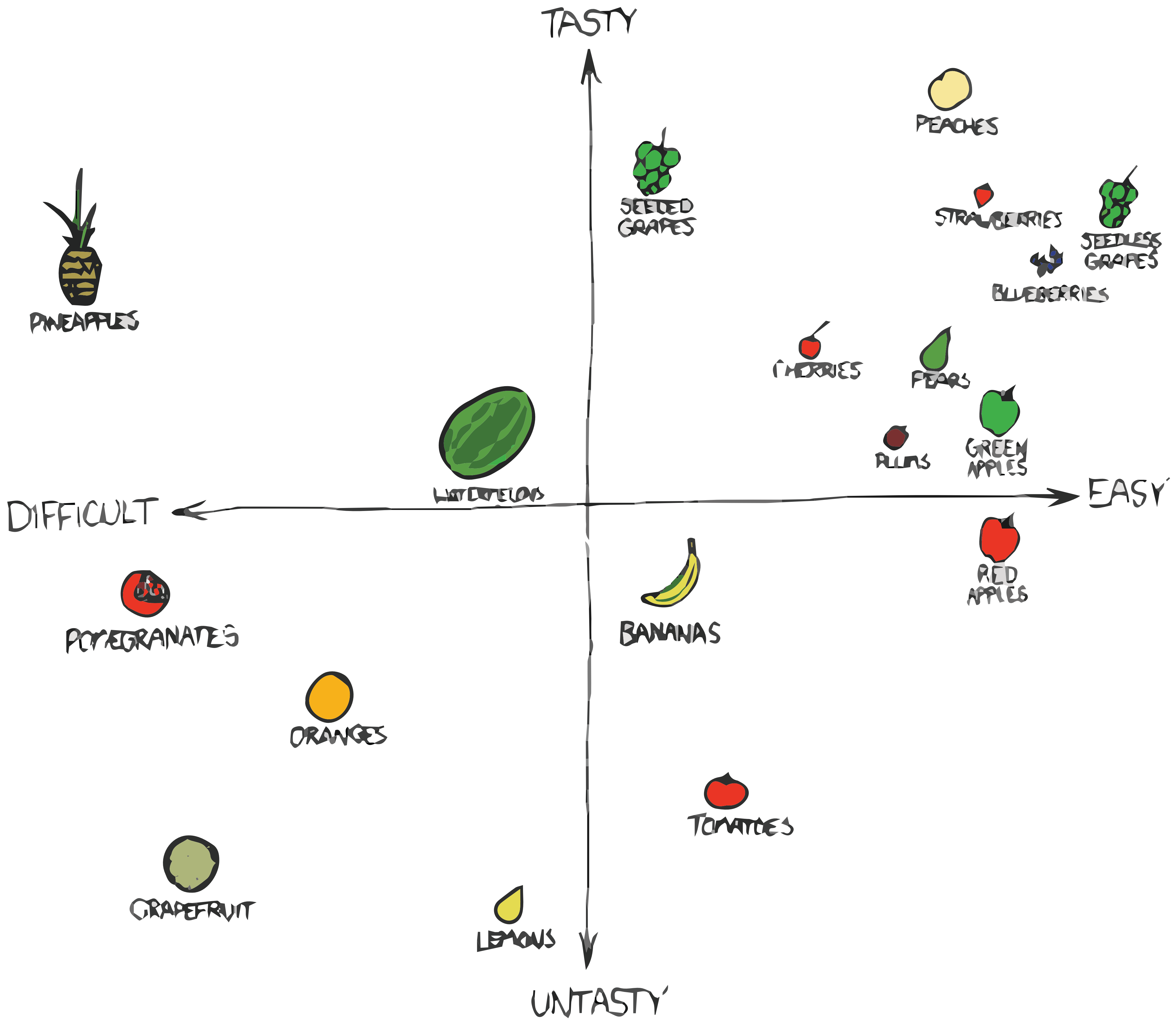


Vectors as information

- A vector is a list of numbers
- Each number can be thought of as representing a "dimension"
  - $\vec{a} = \langle 2,4 \rangle$
  - $\vec{b} = \langle -4,3 \rangle$



Vectors as information



xkcd.com/388

Comparing Vectors

Comparing Vectors

- Any theory of meaning needs to capture **relatedness** between words
  - e.g. "cat" and "kitten" are more **closely related** than "scratch" and "ministry"
  - (You can learn to categorize these relationships in a Formal Semantics course)

Comparing Vectors

- Any theory of meaning needs to capture **relatedness** between words
  - e.g. "cat" and "kitten" are more **closely related** than "scratch" and "ministry"
  - (You can learn to categorize these relationships in a Formal Semantics course)
- How do we capture relationships between **word vectors?**
  - Luckily, we have all kinds of ways to **measure** vector relationships
  - These are mostly divided into metrics of **vector similarity** and **vector distance/ dissimilarity**
  - Which metrics are **closest to human intuitions?**

Vector Length

- A vector's **length** is equal to the *square root of the dot product with itself*
- This is an extension of the **Pythagorean Theorem** for right triangles
- Notice that **dot product** is closely related to length

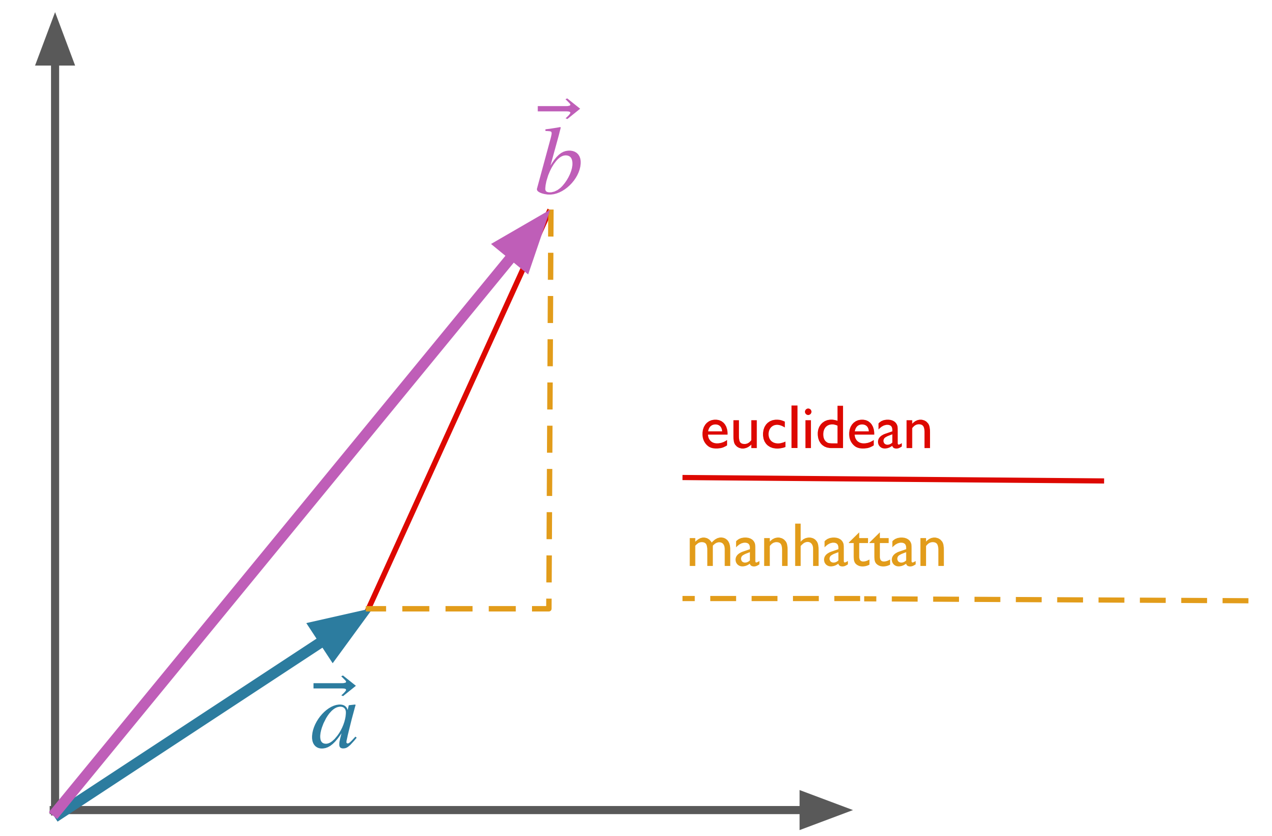
$$\text{length}(x) = \|x\| = \sqrt{x \cdot x}$$

Vector Distances: Manhattan & Euclidean

- **Manhattan Distance**
  - Distance as **cumulative horizontal + vertical moves**
  - Inspired by walking in a **city on a grid**
- **Euclidean Distance**
  - Our normal notion of distance
  - Length of a **straight line** between
- Both are sensitive to **extreme values**

$$d_{\text{manhattan}}(x, y) = \sum_i |x_i - y_i|$$

$$d_{\text{euclidian}}(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$



Vector Similarity: Dot Product

$$\text{sim}_{\text{dot}}(x, y) = x \cdot y = \sum_i x_i y_i$$

Vector Similarity: Dot Product

- Recall: I defined dot product as the "**strength** with which two vectors **go in the same direction**"
  - This can be used as a **similarity metric**
  - Produces a **scalar value**

$$\text{sim}_{\text{dot}}(x, y) = x \cdot y = \sum_i x_i y_i$$

Vector Similarity: Dot Product

- Recall: I defined dot product as the "**strength** with which two vectors **go in the same direction**"
  - This can be used as a **similarity metric**
  - Produces a **scalar value**
- Equals **zero** when the vectors are **orthogonal** (perpendicular)

$$\mathrm{sim}_{\mathrm{dot}}(x, y) = x \cdot y = \sum_i x_i y_i$$

Vector Similarity: Dot Product

- Recall: I defined dot product as the "**strength** with which two vectors **go in the same direction**"
  - This can be used as a **similarity metric**
  - Produces a **scalar value**
- Equals **zero** when the vectors are **orthogonal** (perpendicular)
- **Negative** when the vectors point in **opposite directions**

$$\text{sim}_{\text{dot}}(x, y) = x \cdot y = \sum_i x_i y_i$$

Vector Similarity: Dot Product

- Recall: I defined dot product as the "**strength** with which two vectors **go in the same direction**"
  - This can be used as a **similarity metric**
  - Produces a **scalar value**
- Equals **zero** when the vectors are **orthogonal** (perpendicular)
- **Negative** when the vectors point in **opposite directions**
- Problem: gives **higher similarity** to **longer vectors**

$$\text{sim}_{\text{dot}}(x, y) = x \cdot y = \sum_i x_i y_i$$

Vector Similarity: Cosine

$$\mathrm{sim}_{\mathrm{cosine}}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}}$$

Vector Similarity: Cosine

- The skew towards longer vectors can be solved by **normalizing** the dot product by the **length** of the vectors

$$\text{sim}_{\text{cosine}}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}}$$

Vector Similarity: Cosine

- The skew towards longer vectors can be solved by **normalizing** the dot product by the **length** of the vectors
- This happens to be the same as the **cosine of the angle** between them

$$\text{sim}_{\text{cosine}}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}}$$

Vector Similarity: Cosine

- The skew towards longer vectors can be solved by **normalizing** the dot product by the **length** of the vectors
- This happens to be the same as the **cosine of the angle** between them
  - Equals **-1** if going in **opposite directions**

$$\mathrm{sim}_{\mathrm{cosine}}(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Vector Similarity: Cosine

- The skew towards longer vectors can be solved by **normalizing** the dot product by the **length** of the vectors
- This happens to be the same as the **cosine of the angle** between them
  - Equals **-1** if going in **opposite directions**
  - **Zero** if the vectors are **orthogonal**

$$\text{sim}_{\text{cosine}}(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Vector Similarity: Cosine

- The skew towards longer vectors can be solved by **normalizing** the dot product by the **length** of the vectors
- This happens to be the same as the **cosine of the angle** between them
  - Equals **-1** if going in **opposite directions**
  - **Zero** if the vectors are **orthogonal**
  - **+1** if going in the **same direction** (regardless of length)

$$\text{sim}_{\text{cosine}}(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Vector Similarity: Cosine

- The skew towards longer vectors can be solved by **normalizing** the dot product by the **length** of the vectors
- This happens to be the same as the **cosine of the angle** between them
  - Equals **-1** if going in **opposite directions**
  - **Zero** if the vectors are **orthogonal**
  - **+1** if going in the **same direction** (regardless of length)
- This metric **correlates with human intuitions** of semantic relatedness (demonstrated through experiments)

$$\mathrm{sim}_{\mathrm{cosine}}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

One-hot Vectors

$$\text{cat}: \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{dog}: \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \text{bird}: \begin{bmatrix}0\\0\\1\end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**

$$\text{cat}: \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{dog}: \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \text{bird}: \begin{bmatrix}0\\0\\1\end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)

$$\text{cat} : \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{dog} : \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{bird} : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**

$$\text{cat}: \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{dog}: \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{bird}: \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**
  - (Below is a one-hot representation of a vocabulary with three words)

cat : $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ dog : $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ bird : $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**
  - (Below is a one-hot representation of a vocabulary with three words)
- For a vocabulary with **size** $|V|$, the one-hots need **as many numbers**

$$\text{cat} : \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{dog} : \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{bird} : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**
  - (Below is a one-hot representation of a vocabulary with three words)
- For a vocabulary with **size** $|V|$, the one-hots need **as many numbers**
  - Tens of thousands of words $\rightarrow$ giant vectors!

$$\text{cat}: \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{dog}: \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \text{bird}: \begin{bmatrix}0\\0\\1\end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**
  - (Below is a one-hot representation of a vocabulary with three words)
- For a vocabulary with **size** $|V|$, the one-hots need **as many numbers**
  - Tens of thousands of words $\rightarrow$ giant vectors!
  - The vectors are **sparse**: contain **mostly zeros**

$$\text{cat}: \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{dog}: \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{bird}: \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**
  - (Below is a one-hot representation of a vocabulary with three words)
- For a vocabulary with **size** $|V|$, the one-hots need **as many numbers**
  - Tens of thousands of words $\rightarrow$ giant vectors!
  - The vectors are **sparse**: contain **mostly zeros**
- Each word is **equally similar to all others**

cat : $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ dog : $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ bird : $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

One-hot Vectors

- The simplest possible word vectors are called **one-hot vectors**
  - **Each word** in the vocabulary is given an **"index"** (integer identification)
  - Word is represented by a vector where $x_i = 1$, with **zeros in all other indices**
  - (Below is a one-hot representation of a vocabulary with three words)
- For a vocabulary with **size** $|V|$, the one-hots need **as many numbers**
  - Tens of thousands of words $\rightarrow$ giant vectors!
  - The vectors are **sparse**: contain **mostly zeros**
- Each word is **equally similar to all others**
- How do we get **better representations?**

$$\text{cat}: \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{dog}: \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \text{bird}: \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Distributional Similarity

- How do we create **semantically-structured** word vectors?

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**
- "You shall know a word by the company it keeps!" (*Firth, 1957*)

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**
- "You shall know a word by the company it keeps!" ([Firth, 1957](#))
  - A bottle of *tezgüino* is on the table.

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**
- "You shall know a word by the company it keeps!" (*Firth, 1957*)
    - A bottle of *tezgüino* is on the table.
    - Everybody likes *tezgüino*.

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**
- "You shall know a word by the company it keeps!" (*Firth, 1957*)
  - A bottle of *tezgüino* is on the table.
  - Everybody likes *tezgüino*.
  - *Tezgüino* makes you drunk.

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**
- "You shall know a word by the company it keeps!" ([Firth, 1957](#))
  - A bottle of *tezgüino* is on the table.
  - Everybody likes *tezgüino*.
  - *Tezgüino* makes you drunk.
  - We make *tezgüino* from corn.

Distributional Similarity

- How do we create **semantically-structured** word vectors?
- **"Distributional Hypothesis"**: similar words have **similar contexts**
- "You shall know a word by the company it keeps!" (*Firth, 1957*)
  - A bottle of *tezgüino* is on the table.
  - Everybody likes *tezgüino*.
  - *Tezgüino* makes you drunk.
  - We make *tezgüino* from corn.
- Tezgüino: corn-based alcoholic beverage.

Bag of Words Vectors

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Bag of Words Vectors

- We want words with **similar contexts** to have **similar vectors**
  - What if we directly represent the context as a vector?

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Bag of Words Vectors

- We want words with **similar contexts** to have **similar vectors**
  - What if we directly represent the context as a vector?
- Idea: for each word, **count** the number of times that any other word **appeared "close-by"**
  - Define "close-by" as appearing within a **sliding window** around the target word (i.e. $\pm n$ words, where n is chosen by the engineer)
  - Do this counting over a **corpus** (dataset) of language data

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

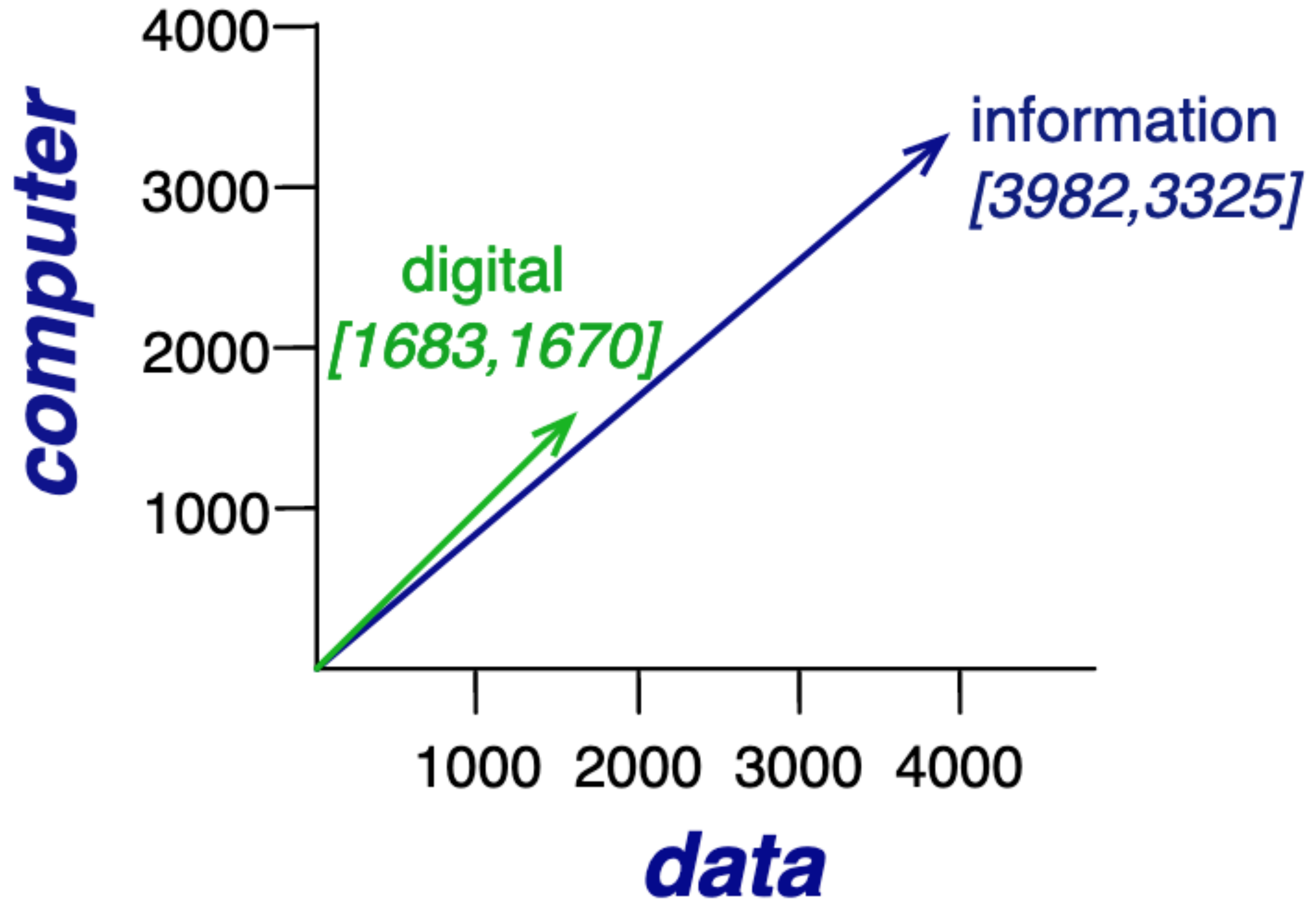
Bag of Words Vectors

- We want words with **similar contexts** to have **similar vectors**
  - What if we directly represent the context as a vector?
- Idea: for each word, **count** the number of times that any other word **appeared "close-by"**
  - Define "close-by" as appearing within a **sliding window** around the target word (i.e. $\pm n$ words, where n is chosen by the engineer)
  - Do this counting over a **corpus** (dataset) of language data
- What is the **size** of these vectors?
  - (Still $|V|$)

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Bag of Words Vectors

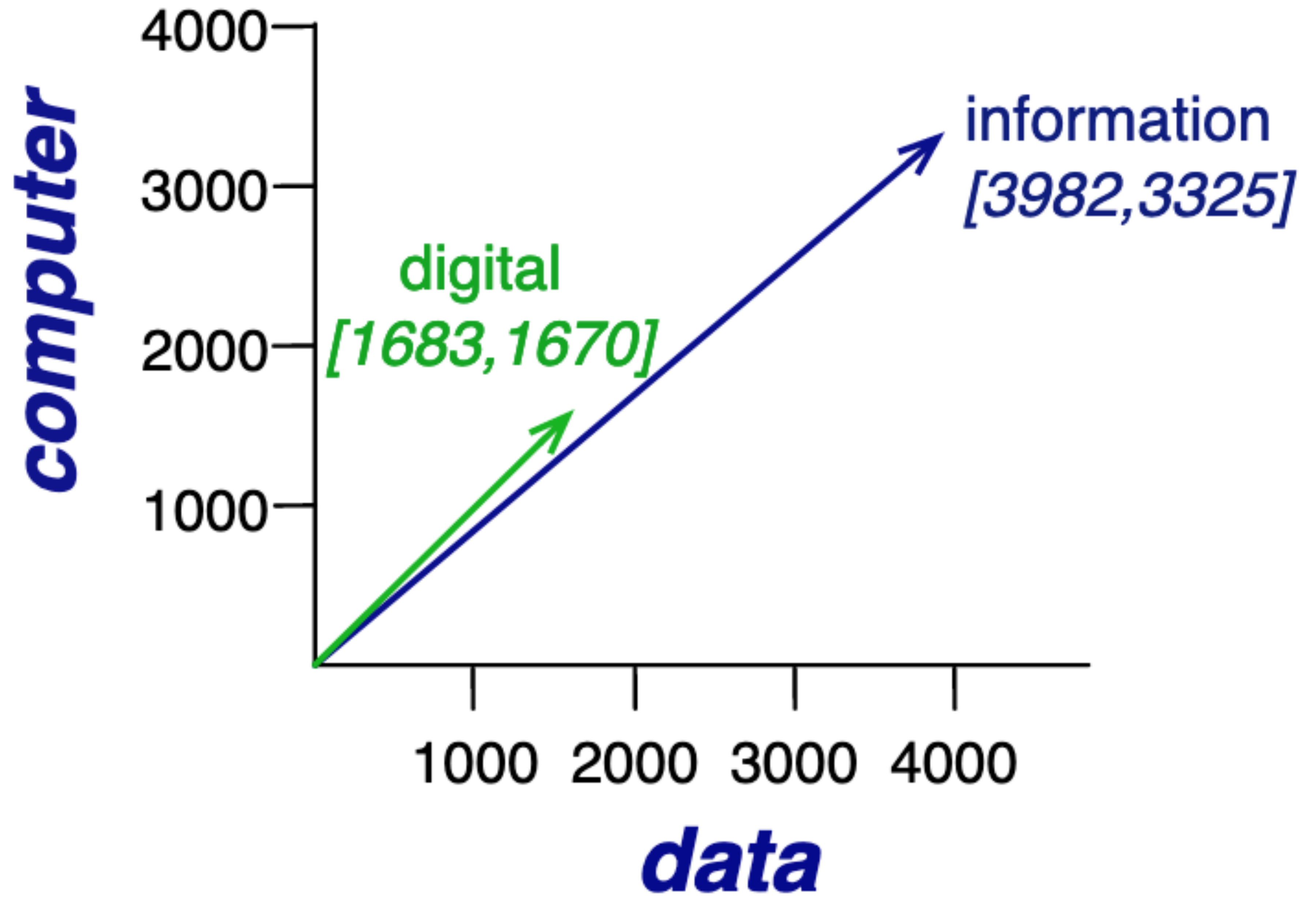
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Bag of Words Vectors

- Can capture **basic similarities** between words

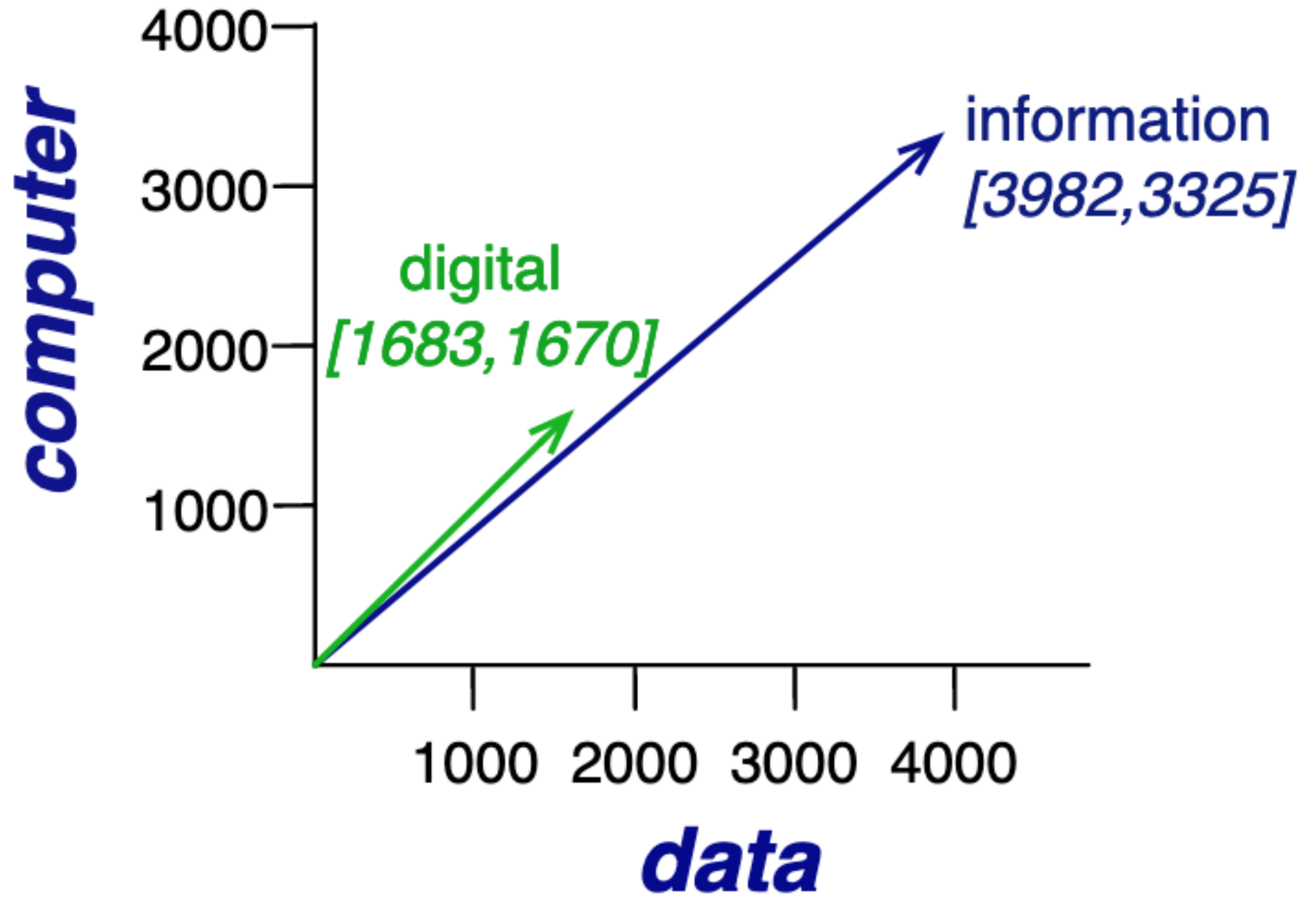
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Bag of Words Vectors

- Can capture **basic similarities** between words
- Usually **re-weighted** by some more sophisticated algorithm
  - (e.g. tf-idf, ppmi)

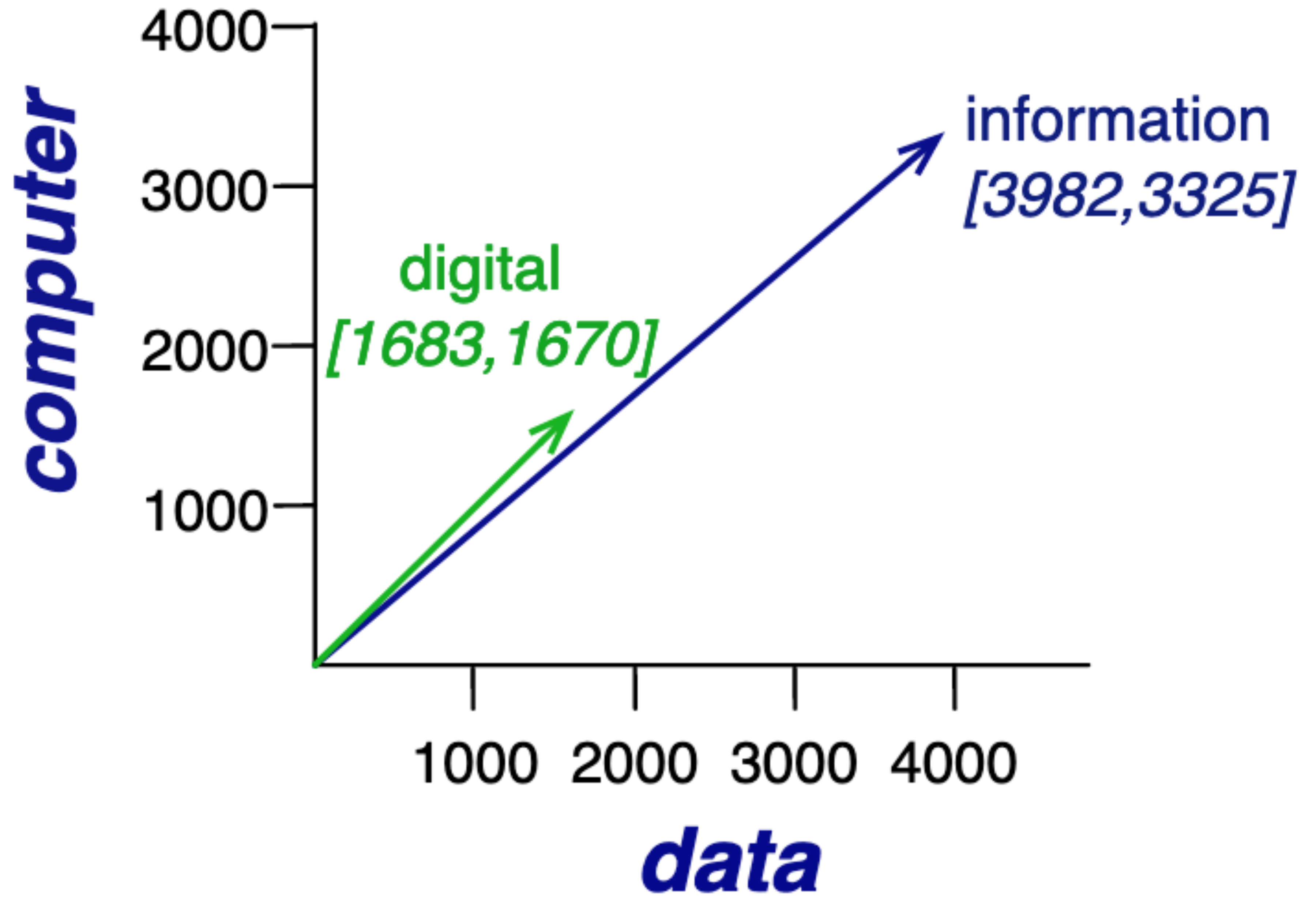
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Bag of Words Vectors

- Can capture **basic similarities** between words
- Usually **re-weighted** by some more sophisticated algorithm
  - (e.g. tf-idf, ppmi)
- Problems:
  - Still very **high-dimensional** ($|V|$)
  - Still very **sparse** (most pairs of words **never co-occur**)

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Prediction-Based Models (word2vec)

Prediction-based Embeddings

Prediction-based Embeddings

- **Skip-gram** and **Continuous Bag of Words** (CBOW) models

Prediction-based Embeddings

- **Skip-gram** and **Continuous Bag of Words** (CBOW) models
- Words with **similar meanings** share **similar contexts**

Prediction-based Embeddings

- **Skip-gram** and **Continuous Bag of Words** (CBOW) models
- Words with **similar meanings** share **similar contexts**
- Instead of counting, train models to **predict context words**

Prediction-based Embeddings

- **Skip-gram** and **Continuous Bag of Words** (CBOW) models
- Words with **similar meanings** share **similar contexts**
- Instead of counting, train models to **predict context words**
- The **embeddings** (word vectors) used to accomplish the prediction become **semantically structured**

Skip-Gram vs. Continuous Bag of Words

Skip-Gram vs. Continuous Bag of Words

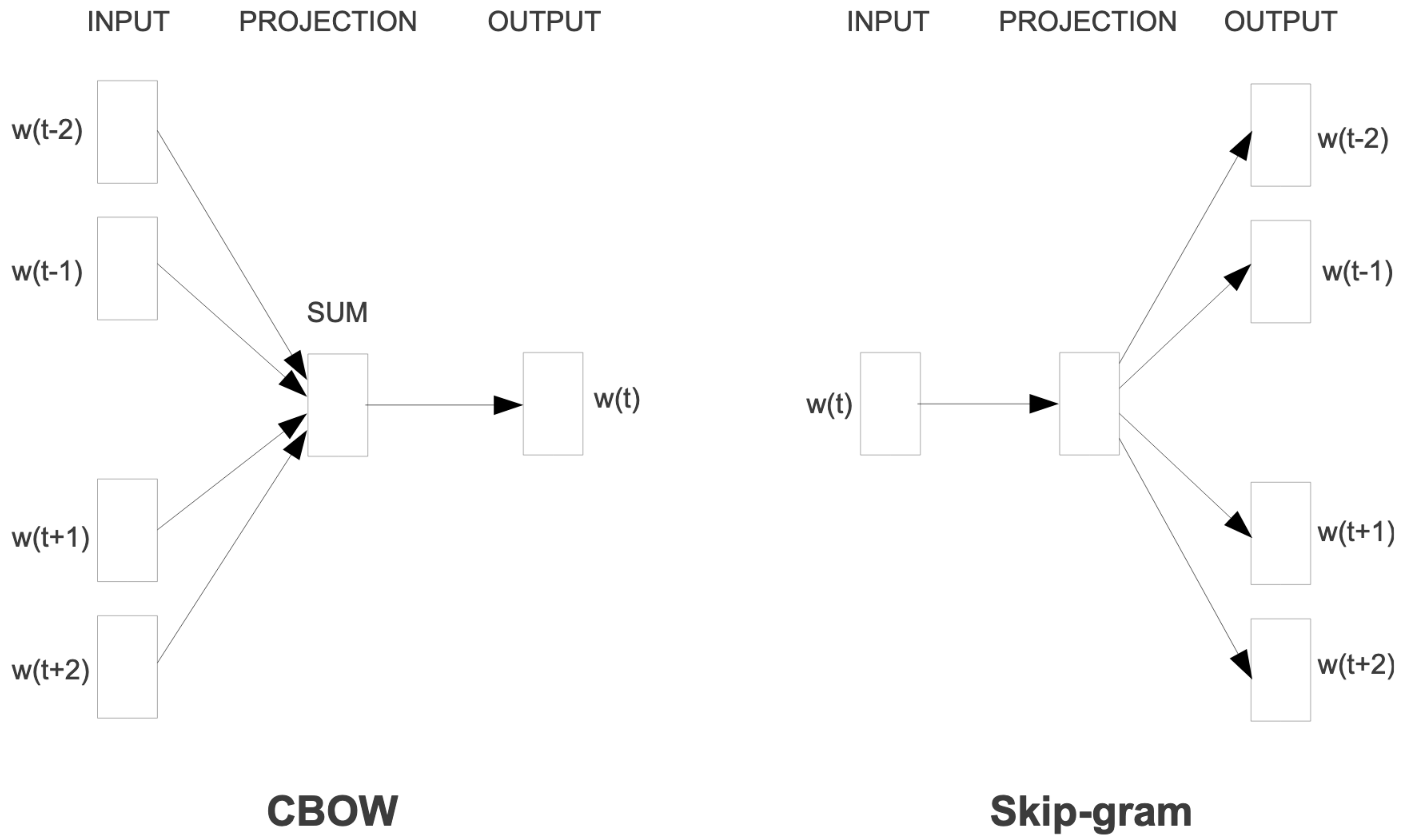
- Continuous Bag of Words (CBOW):
  - $P(\textit{word} \mid \textit{context})$
  - Input: $(w_{t-1}, w_{t-2}, w_{t+1}, wt_{+2} \ldots)$
  - Output: $p(w_t)$

Skip-Gram vs. Continuous Bag of Words

- Continuous Bag of Words (CBOW):
  - ***P*(*word* | *context*)**
  - Input: (w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} …)
  - Output: $p(w_t)$
- **Skip-gram:**
  - ***P*(*context* | *word*)**
  - Input: w_t
  - Output: $p(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \ldots)$

Skip-Gram vs. Continuous Bag of Words

- Continuous Bag of Words (CBOW)
  - $P(\textit{word} \mid \textit{context})$
  - Input: $(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \ldots)$
  - Output: $p(w_t)$
- **Skip-gram:**
  - $P(\textit{context} \mid \textit{word})$
  - Input: w_t
  - Output: $p(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \ldots)$



Mikolov et al 2013a (the OG word2vec paper)

Skip-Gram Model

Skip-Gram Model

- Learns **two embedding matrices**
  - W : **word**, matrix of shape [vocab_size, embedding_dimension]
  - C : **context** embedding, matrix of same shape

Skip-Gram Model

- Learns **two embedding matrices**
  - W : **word**, matrix of shape [vocab_size, embedding_dimension]
  - C : **context** embedding, matrix of same shape
- Prediction task:
  - Given a word, **predict each neighbor** word in window
  - Compute $p(w_k \mid w_j)$ as proportional to $C_k \cdot W_j$
  - Convert to probability via Softmax
    - (Softmax is a version of Sigmoid we'll discuss more later)

$$p(w_k \mid w_j) = \frac{e^{\mathbf{C}_k \cdot \mathbf{W}_j}}{\sum_i e^{\mathbf{C}_i \cdot \mathbf{W}_j}}$$

Parameters and Hyper-parameters

Parameters and Hyper-parameters

- The embedding dimension and window size are **hyper-parameters**
  - Chosen by the modeler/practitioner
  - **Not updated** by the learning algorithm

Parameters and Hyper-parameters

- The embedding dimension and window size are **hyper-parameters**
  - Chosen by the modeler/practitioner
  - **Not updated** by the learning algorithm
- Other hyper-parameters we've seen so far
  - **Learning rate** for GD
  - **Batch size** for mini-batch GD

Parameters and Hyper-parameters

- The embedding dimension and window size are **hyper-parameters**
  - Chosen by the modeler/practitioner
  - **Not updated** by the learning algorithm
- Other hyper-parameters we've seen so far
  - **Learning rate** for GD
  - **Batch size** for mini-batch GD
- Will talk more about **how to choose** hyper-parameters later

Parameters and Hyper-parameters

- The embedding dimension and window size are **hyper-parameters**
  - Chosen by the modeler/practitioner
  - **Not updated** by the learning algorithm
- Other hyper-parameters we've seen so far
  - **Learning rate** for GD
  - **Batch size** for mini-batch GD
- Will talk more about **how to choose** hyper-parameters later
- **Parameters**: parts of the model that are **updated by the learning algorithm**

Power of Prediction-based Embeddings

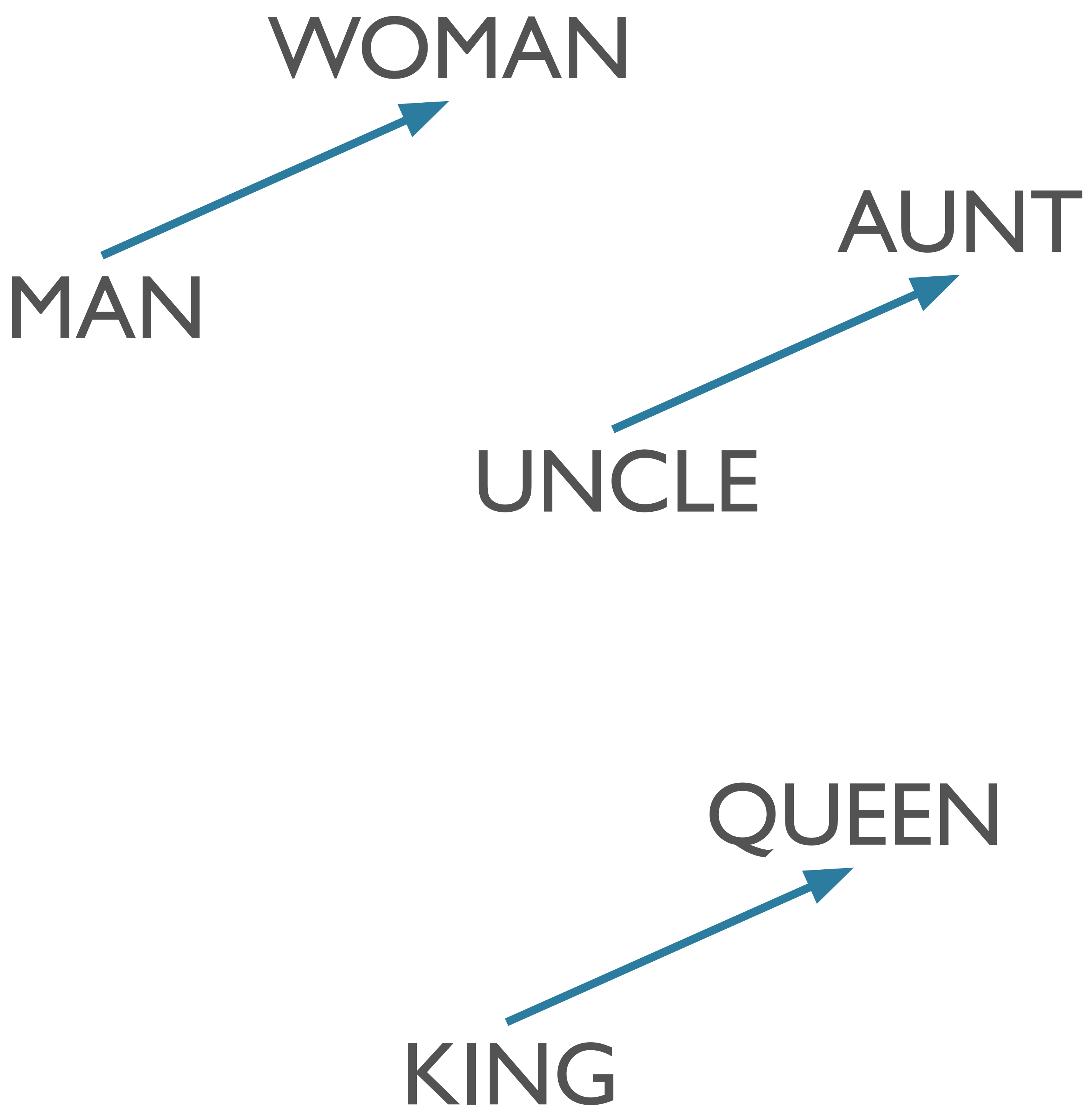
Power of Prediction-based Embeddings

- Count-based embeddings
  - Very high-dimensional ($|V|$)
  - Sparse
  - Pro: features are interpretable (“occurred with word W, N times in corpus”)

Power of Prediction-based Embeddings

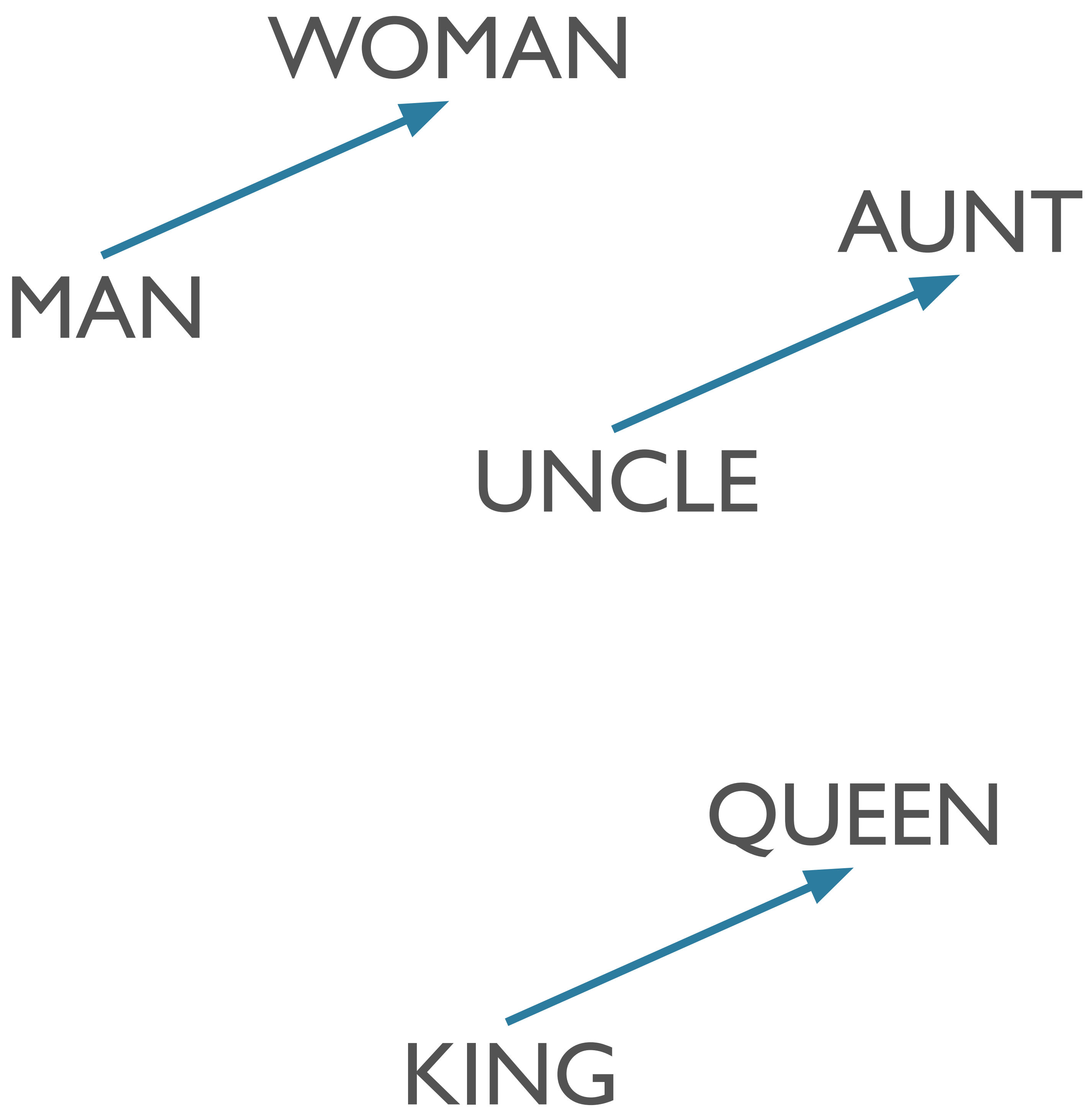
- Count-based embeddings
  - Very high-dimensional ($|V|$)
  - Sparse
  - Pro: features are interpretable ("occurred with word W, N times in corpus")
- Prediction-based embeddings
  - **"Low"-dimensional** (typically ~256-2048)
  - **Dense**
  - Con: features are **not immediately interpretable**
    - What does "dimension 36 has value -9.63" mean?

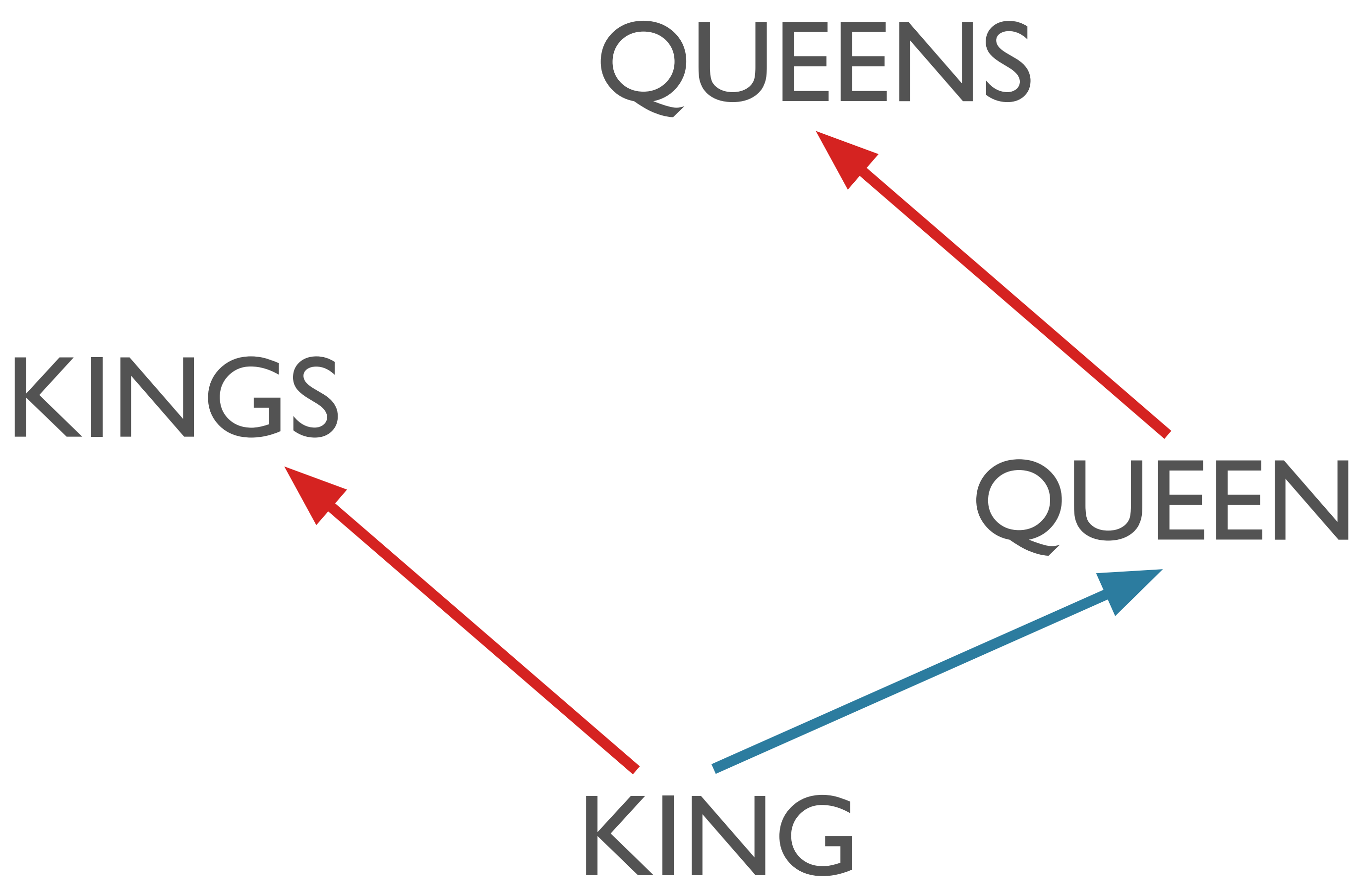
Relationships via Offsets



Mikolov et al 2013b

Relationships via Offsets





[Mikolov et al 2013b]

One More Example

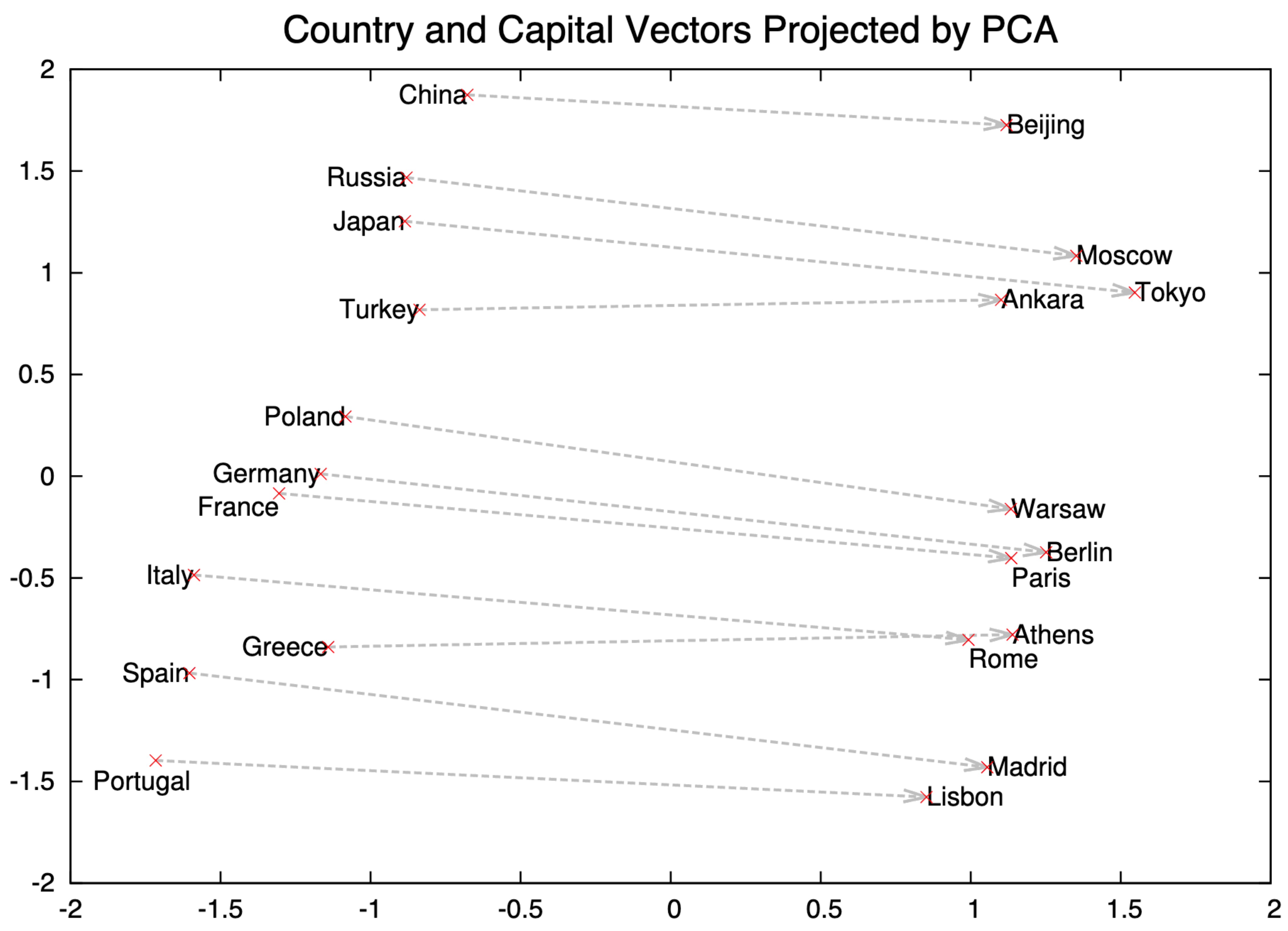
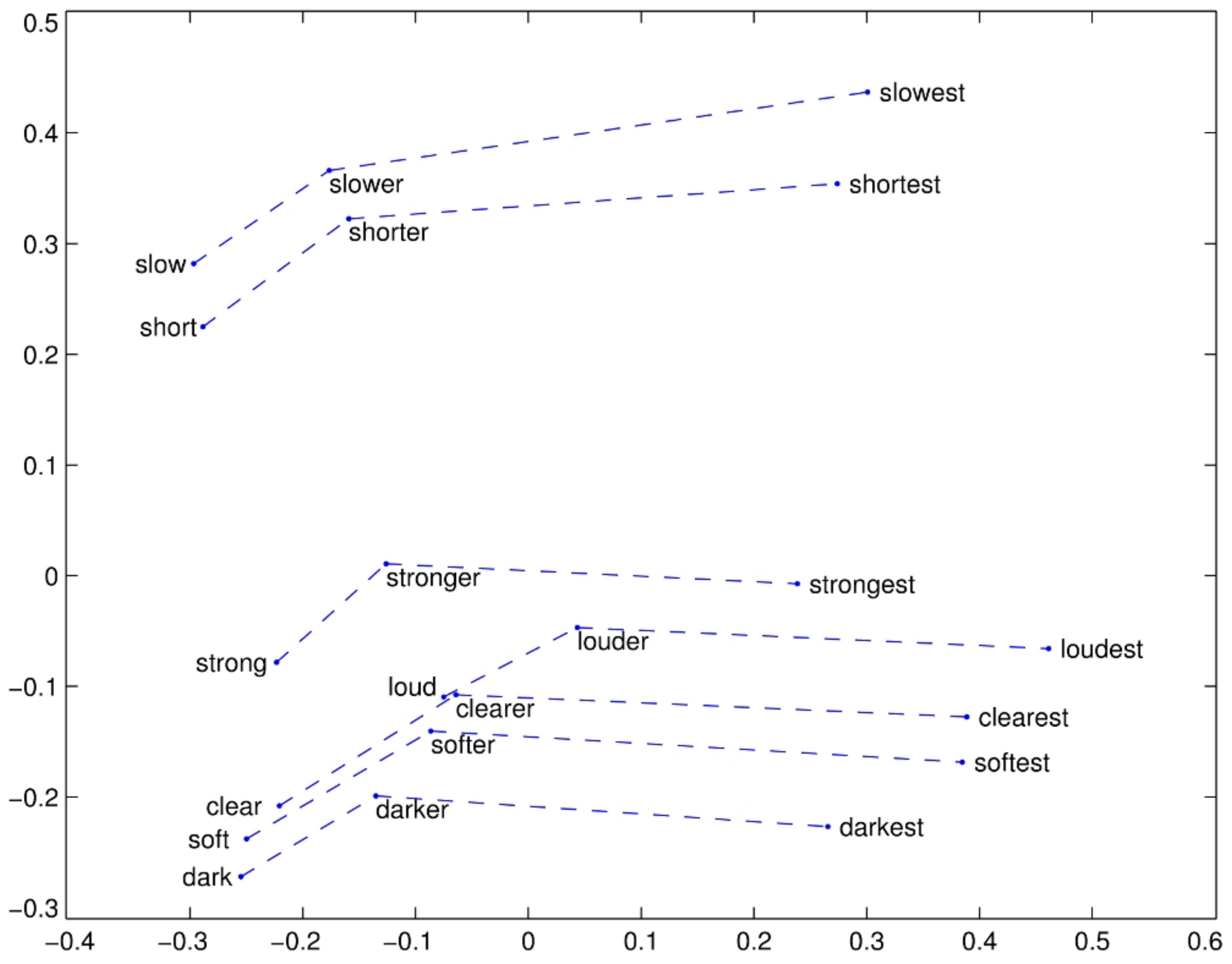


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

[Mikolov et al 2013c]

One More Example



Caveat Emptor

Issues in evaluating semantic spaces using word analogies

Tal Linzen
LSCP & IJN
École Normale Supérieure
PSL Research University
tal.linzen@ens.fr

Abstract

The offset method for solving word analogies has become a standard evaluation tool for vector-space semantic models: it is considered desirable for a space to represent semantic relations as consistent vector offsets. We show that the method's reliance on cosine similarity conflates offset consistency with largely irrelevant neighborhood structure, and propose simple baselines that should be used to improve the utility of the method in vector space evaluation.

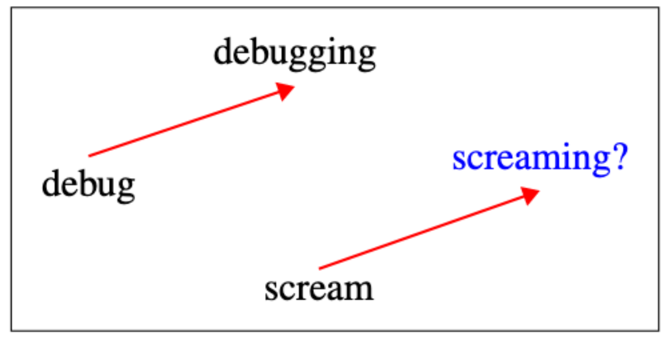


Figure 1: Using the vector offset method to solve the analogy task (Mikolov et al., 2013c).

cosine similarity to the landing point. Formally, if the analogy is given by

$$a : a^* :: b : \_\_\_ \qquad (1)$$

Linzen 2016, a.o.

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi[1], Kai-Wei Chang[2], James Zou[2], Venkatesh Saligrama[1,2], Adam Kalai[2]
[1]Boston University, 8 Saint Mary's Street, Boston, MA
[2]Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

Abstract

The blind application of machine learning runs the risk of amplifying biases present in data. Such a danger is facing us with *word embedding*, a popular framework to represent text data as vectors which has been used in many machine learning and natural language processing tasks. We show that even word embeddings trained on Google News articles exhibit female/male gender stereotypes to a disturbing extent.

Bolukbasi et al 2016

Skip-Gram with Negative Sampling (SGNS)

Training The Skip-Gram Model

- Issue: **denominator** computation is **very expensive**
- Alternative: approximate by **negative sampling**
  - Use **"fake" examples**, and make their probability **small**
  - + example: true context word
  - – examples: *k* other words, randomly sampled

$$p(w_k \mid w_j) = \frac{e^{C_k \cdot W_j}}{\sum_i e^{C_i \cdot W_j}}$$

Negative Sampling, Idea

Negative Sampling, Idea

- Skip-Gram
  - $p(w_k \mid w_j)$: what is the probability that w_k occurred in the context of w_j
  - Classifier with $|V|$ classes

Negative Sampling, Idea

- Skip-Gram
  - $p(w_k \mid w_j)$: what is the probability that w_k occurred in the context of w_j
  - Classifier with $|V|$ classes
- Negative sampling
  - $p(+ \mid w_k, w_j)$: what is the probability that (w_k, w_j) was a **true co-occurrence**?
  - $p(- \mid w_k, w_j) = 1 - p(+ \mid w_k, w_j)$
    - Probability that (w_k, w_j) was **not** a true co-occurrence
    - Examples of **“fake” co-occurrences** = **negative samples**
  - Binary classifier

Generating Positive Examples

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1          c2     w       c3         c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Generating Positive Examples

- Iterate through the corpus

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1          c2    w      c3         c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Generating Positive Examples

- Iterate through the corpus
- For each word: add **all words within *window_size*** of the current word as **positive** pairs
  - *window_size* is a **hyper-parameter**

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1          c2     w      c3         c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Negative Samples

- For each positive (w, c) sample, **generate *num_negatives* samples**
  - $(w, c\_)$, where $c\_$ is different from c
  - *num_negatives* is another hyper-parameter

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

The Data

The Data

- X = pairs of words

The Data

- X = pairs of words
- Y = {0, 1}
  - 1 = + (positive example), 0 = - (negative example)

The Data

- X = pairs of words
- Y = {0, 1}
  - 1 = + (positive example), 0 = - (negative example)
- Example (x, y) pairs:
  - ((“apricot”, “tablespoon”), 1)
  - ((“apricot”, “jam”), 1)
  - ((“apricot”, “aardvark”), 0)
  - ((“apricot”, “my”), 0)

The Model

The Model

- So what is $p(1 \mid w, c)$?

The Model

- So what is $p(1 \mid w, c)$?
  - More specifically: $p(1 \mid w, c; \theta)$

The Model

- So what is $p(1 \mid w, c)$?
  - More specifically: $p(1 \mid w, c; \theta)$
- As before, learns **two embedding matrices**

The Model

- So what is $p(1 \mid w, c)$?
  - More specifically: $p(1 \mid w, c; \theta)$
- As before, learns **two embedding matrices**
  - E **: word embeddings**, matrix of shape [vocab_size, embedding_dimension]

The Model

- So what is $p(1 \mid w, c)$?
  - More specifically: $p(1 \mid w, c; \theta)$
- As before, learns **two embedding matrices**
  - E **: word embeddings**, matrix of shape [vocab_size, embedding_dimension]
    - E_w: embedding for word *w* (row of the matrix)

The Model

- So what is $p(1 \mid w, c)$?
  - More specifically: $p(1 \mid w, c; \theta)$
- As before, learns **two embedding matrices**
  - E **: word embeddings**, matrix of shape [vocab_size, embedding_dimension]
    - E_w: embedding for word *w* (row of the matrix)
  - C **: context embeddings**, matrix of same shape

The Model

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

The Model

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word
embedding

The Model

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word embedding

Context word embedding

The Model

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word embedding

Context word embedding

Similarity (dot-product)

The Model

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

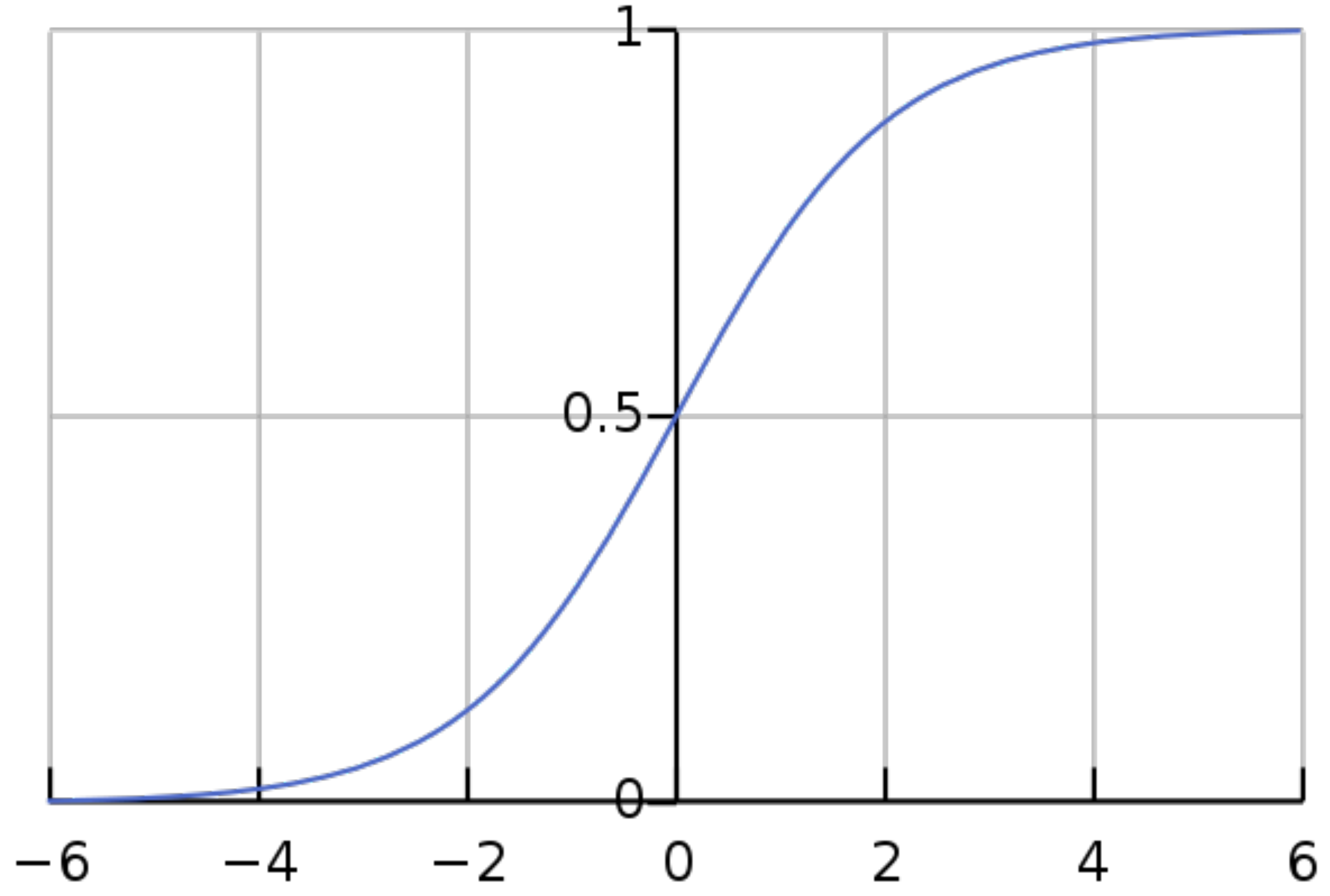
sigmoid

Target word embedding

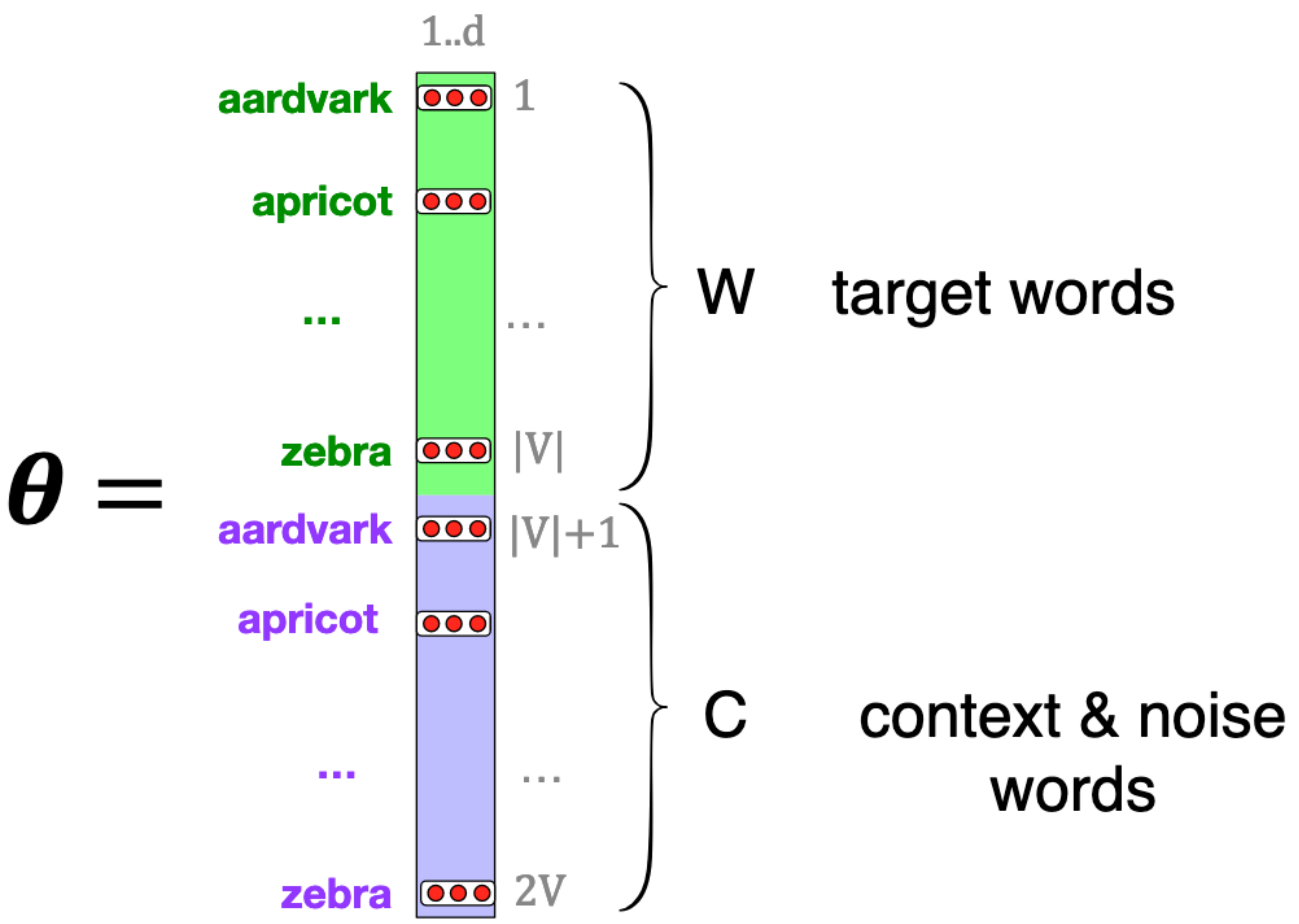
Similarity (dot-product)

Context word embedding

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



Learning: Parameters



The Model

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

The Model

- Target and context words that are **more similar** to each other (have more similar embeddings) have a **higher probability** of being a positive example

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

The Model

- Target and context words that are **more similar** to each other (have more similar embeddings) have a **higher probability** of being a positive example
- We want our model to:
  - Assign high $p(1 \mid w, c_+)$ (c+ is a positive context word)
  - Assign low $p(1 \mid w, c_-)$ (c- is a negative context word)
    - Equivalently: assign high $p(0 \mid w, c_-)$

$$p(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Aside: Log Probabilities

Aside: Log Probabilities

- In machine learning, we often want to **multiply many probabilities**
  - I.e. to represent the probability of **many events** happening jointly

Aside: Log Probabilities

- In machine learning, we often want to **multiply many probabilities**
  - I.e. to represent the probability of **many events** happening jointly
- Problem: multiplying probabilities **quickly rounds to zero** (especially for computer programs)

Aside: Log Probabilities

- In machine learning, we often want to **multiply many probabilities**
  - I.e. to represent the probability of **many events** happening jointly
- Problem: multiplying probabilities **quickly rounds to zero** (especially for computer programs)
- Solution: **sum log probabilities** instead of multiplying regular ones
  - $log(p_1 \cdot p_2) = log(p_1) + log(p_2)$
  - $log(1.0) = 0.0;\ log(0.0) = -\infty$
  - **Numerically stable** (i.e. less likely to get a **rounding error**)

Aside: Log Probabilities

- In machine learning, we often want to **multiply many probabilities**
  - I.e. to represent the probability of **many events** happening jointly
- Problem: multiplying probabilities **quickly rounds to zero** (especially for computer programs)
- Solution: **sum log probabilities** instead of multiplying regular ones
  - $log(p_1 \cdot p_2) = log(p_1) + log(p_2)$
  - $log(1.0) = 0.0;\ log(0.0) = -\infty$
  - **Numerically stable** (i.e. less likely to get a **rounding error**)
- To make it a **loss function**, take the **negative log probability** $-log(p)$

Loss: Binary Cross-Entropy

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label, $\hat{y}$ is the model’s prediction

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label, $\hat{y}$ is the model’s prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log p(1 \mid w, c)$

Loss: Binary Cross-Entropy

- Recall: y is our "gold standard" label, $\hat{y}$ is the model's prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log p(1 \mid w, c)$
- When y = 0
  - minimize: $-\log(1 - \hat{y}) = -\log(1 - p(1 \mid w, c)) = -\log p(0 \mid w, c)$

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label, $\hat{y}$ is the model’s prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log p(1 \mid w, c)$
- When y = 0
  - minimize: $-\log(1 - \hat{y}) = -\log(1 - p(1 \mid w, c)) = -\log p(0 \mid w, c)$
- I.e. the **negative log probability** that the model assigns to the true label

Loss: Binary Cross-Entropy

- Recall: y is our "gold standard" label, $\hat{y}$ is the model's prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log p(1 \mid w, c)$
- When y = 0
  - minimize: $-\log(1 - \hat{y}) = -\log(1 - p(1 \mid w, c)) = -\log p(0 \mid w, c)$
- I.e. the **negative log probability** that the model assigns to the true label
- BCE loss **incorporates both** into the closed form (y is either 1 or 0)

$$\ell_{BCE}(\hat{y}, y) := -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

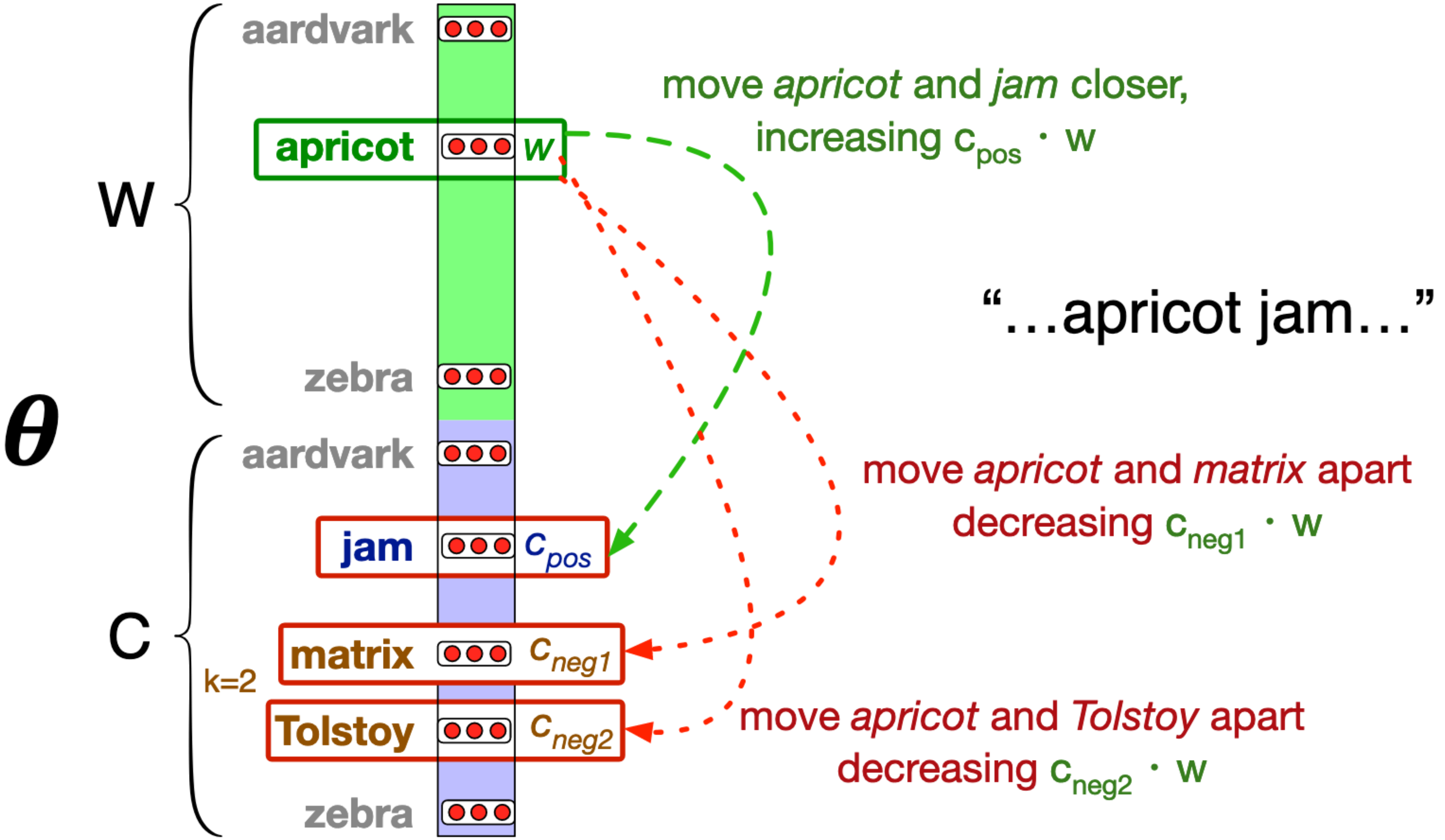
Training Loop w/ Negative Samples

```
initialize parameters / build model

for each epoch:

 positives = shuffle(positives)

 for each example in positives:

  positive_output = model(example)
  generate k negative samples
  negative_outputs = [model(negatives)]
  compute gradients
  update parameters
```

Combo Loss

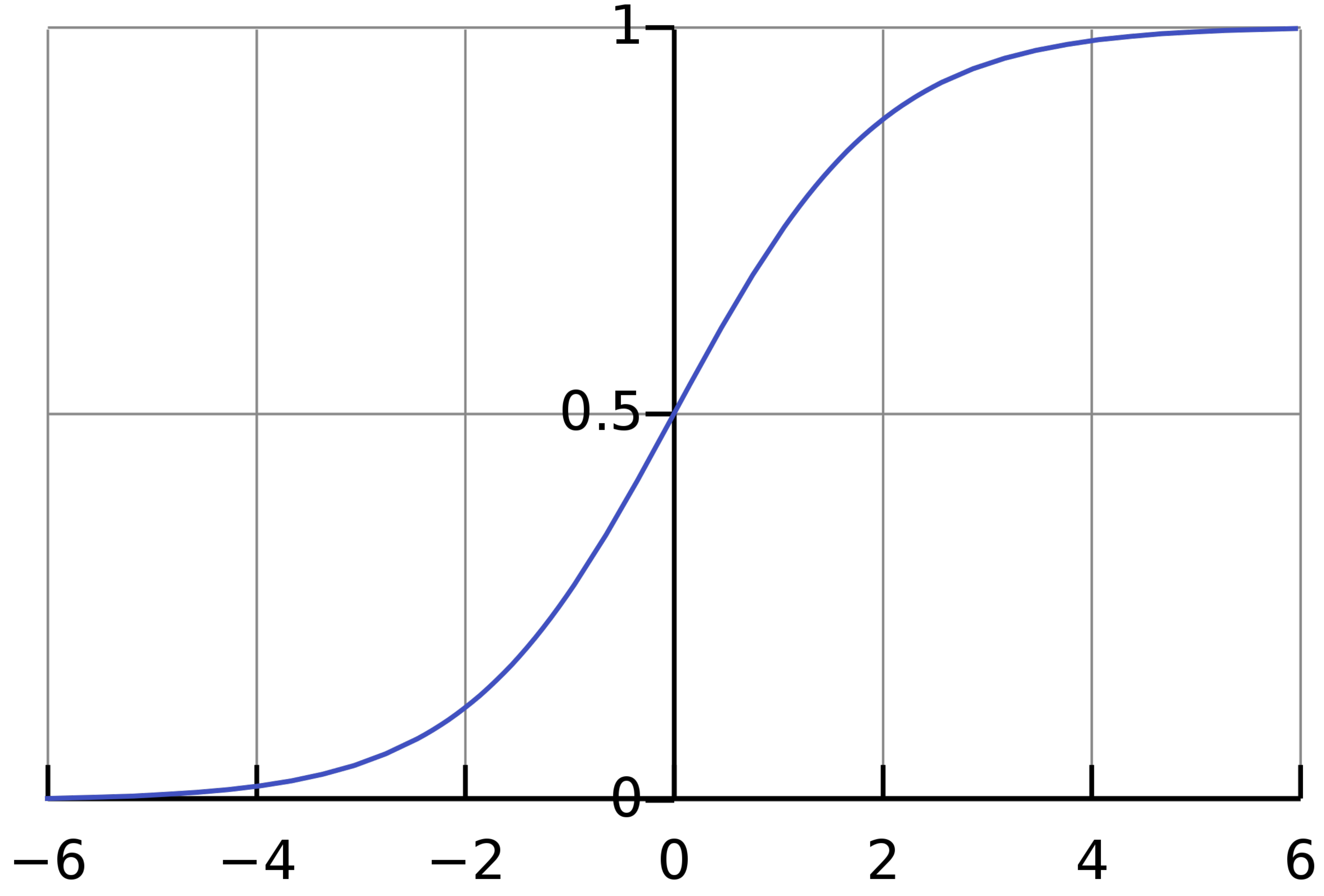
$$
\begin{aligned}
L_{CE} &= -\log P(1,0,0,\ldots,0 \mid w, c_+, c_{-1}, c_{-2}, \ldots, c_{-k}) \\
&= -\log\left(P(1 \mid w, c_+) \prod_{i=1}^{k} P(0 \mid w, c_{-i})\right) \\
&= -\log P(1 \mid w, c_+) - \sum_{i=1}^{k} \log P(0 \mid w, c_{-i})
\end{aligned}
$$

Learning: Intuitively



Appendix: Sigmoid and Softmax

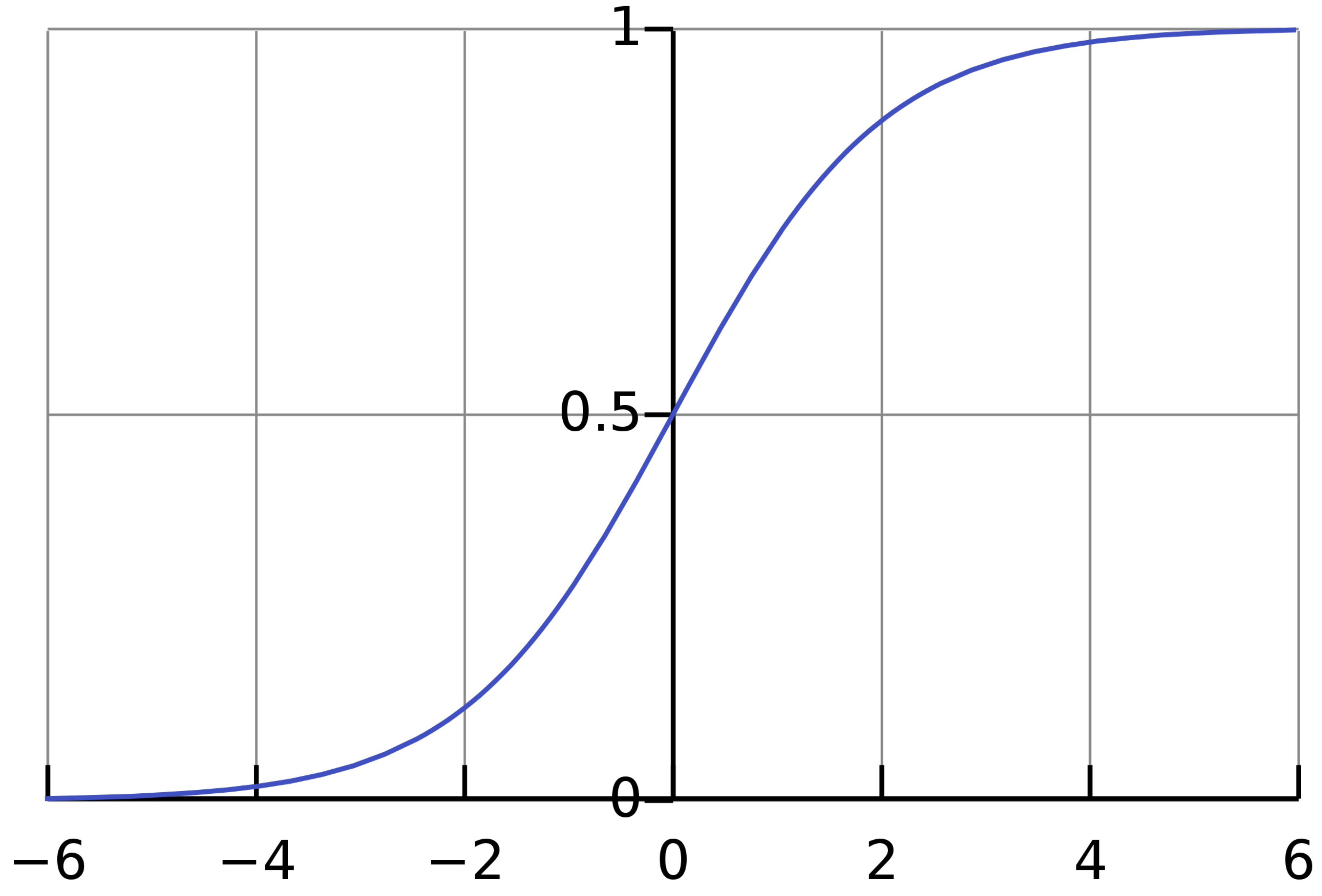
Sigmoid



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

Sigmoid

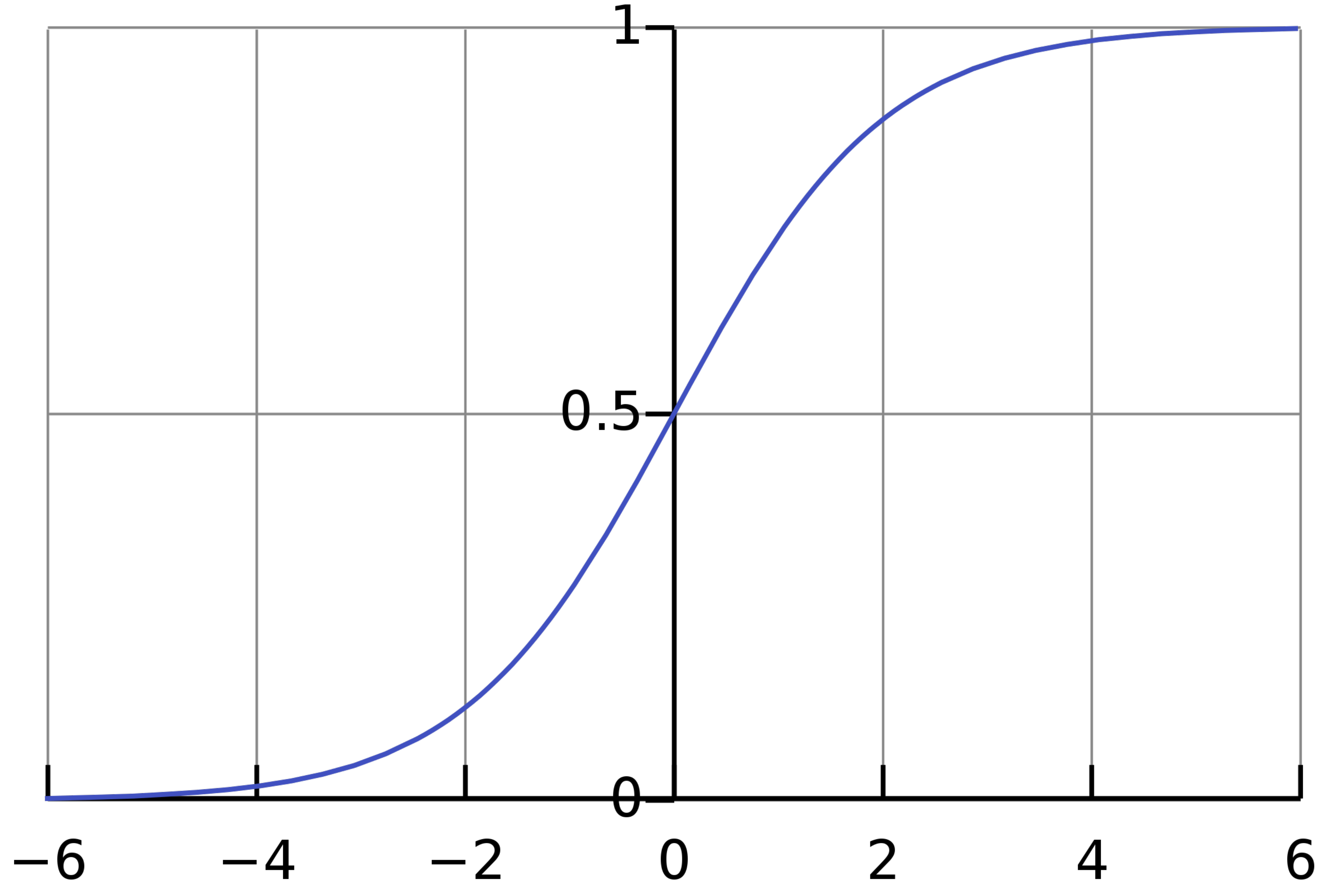
- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0, 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

Sigmoid

- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0,\ 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$
- e^x represents the **odds of "True"** in the binary classification



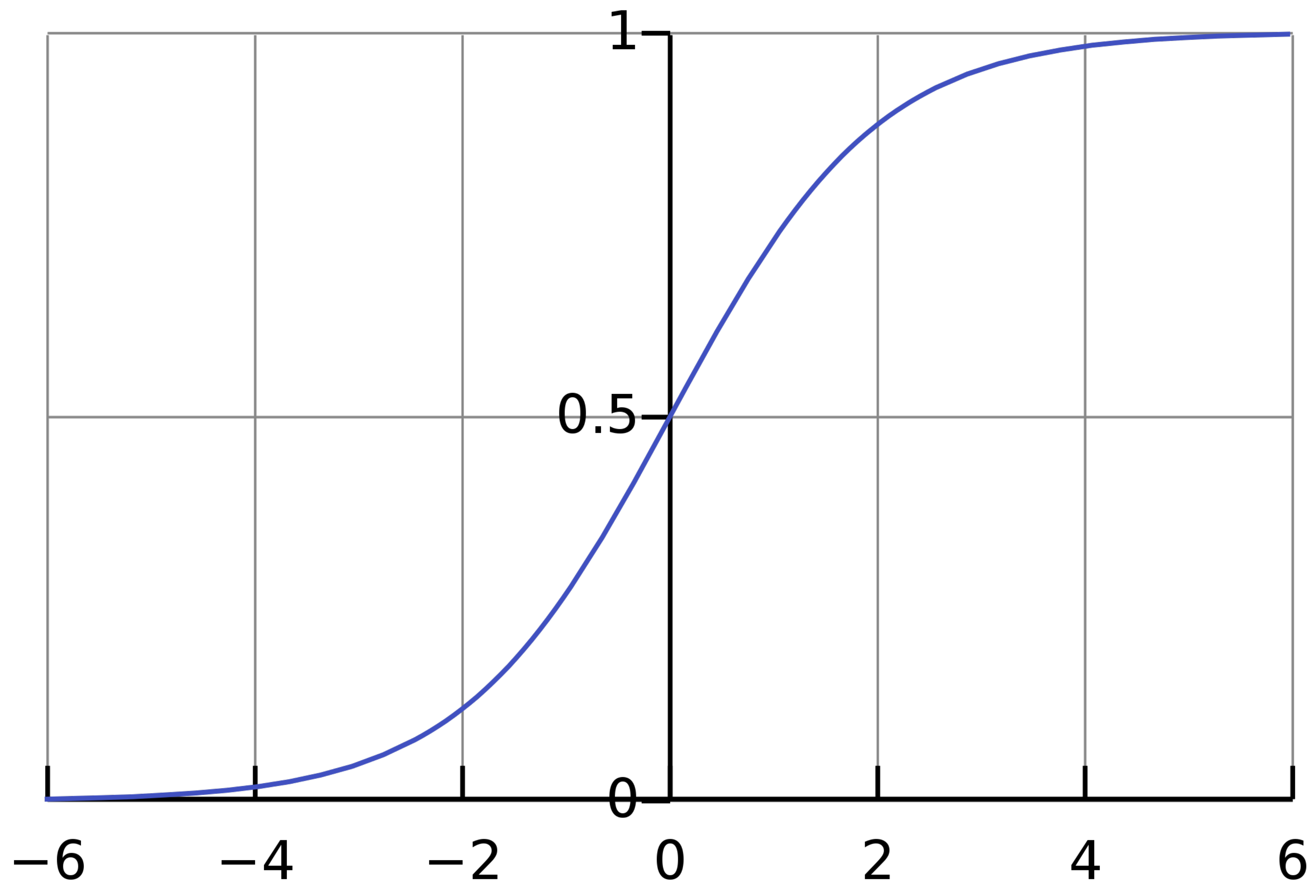
$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

"True" odds

"True" odds

Sigmoid

- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0, 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$
- e^x represents the **odds of "True"** in the binary classification
- e^0 represents the **odds of "False"**



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

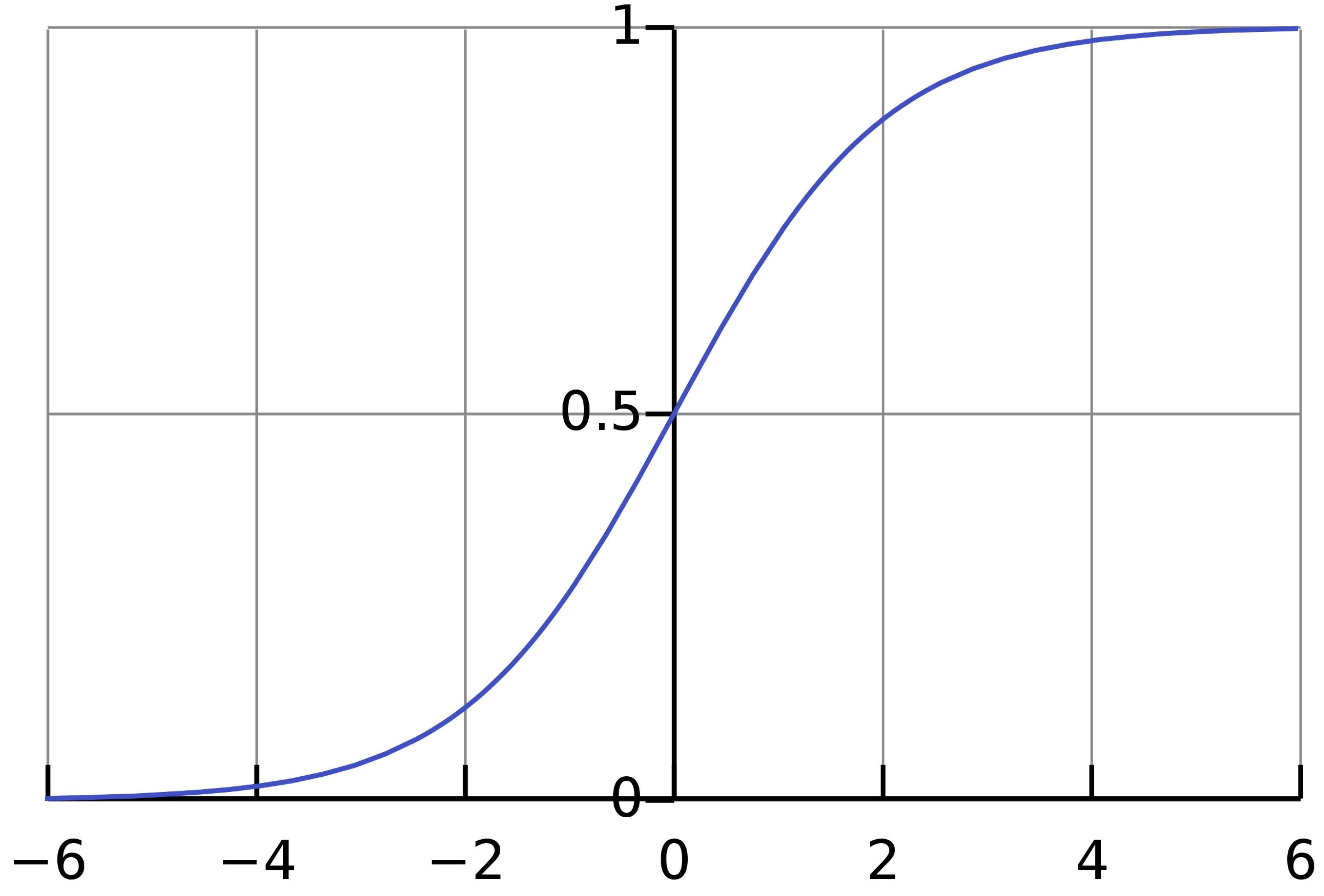
"True" odds

"True" odds

"False" odds

Sigmoid

- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0, 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$
- e^x represents the **odds of "True"** in the binary classification
- e^0 represents the **odds of "False"**
- $\sigma(x)$ is the "True" odds **divided by the sum of the odds**
  - (i.e. the **probability of True**)



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

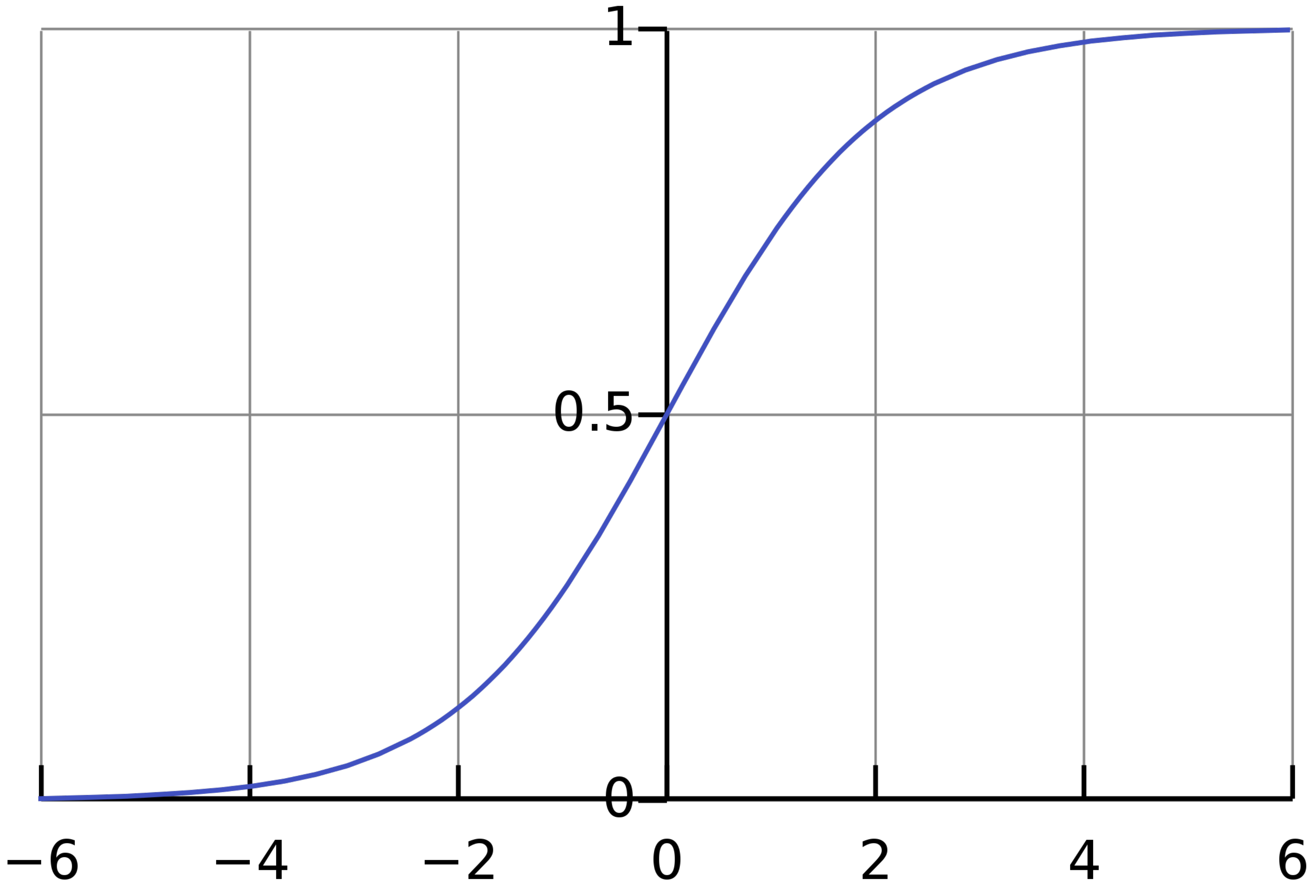
"True" odds

"True" odds

"False" odds

Sigmoid

- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0, 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$
- e^x represents the **odds of "True"** in the binary classification
- e^0 represents the **odds of "False"**
- $\sigma(x)$ is the "True" odds **divided by the sum of the odds**
  - (i.e. the **probability of True**)
- $x = 0$ represents the **default case**
  - $\sigma(0) = 1/(1+1) = 1/2$
  - This is the **"threshold"** of the Sigmoid function



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

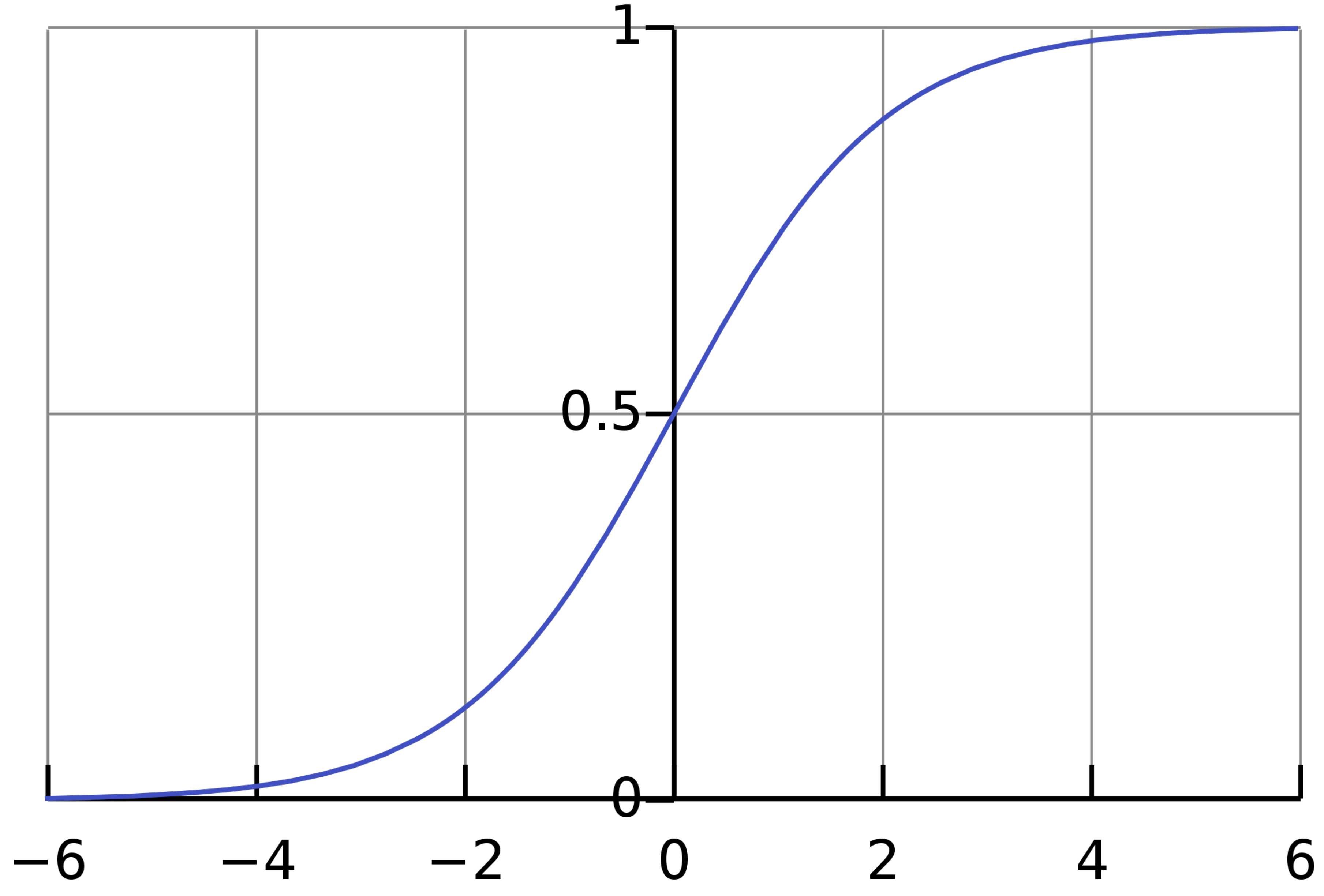
"True" odds

"True" odds

"False" odds

Sigmoid

- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0,\ 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$
- e^x represents the **odds of "True"** in the binary classification
- e^0 represents the **odds of "False"**
- $\sigma(x)$ is the "True" odds **divided by the sum of the odds**
  - (i.e. the **probability of True**)
- $x = 0$ represents the **default case**
  - $\sigma(0) = 1/(1+1) = 1/2$
  - This is the **"threshold"** of the Sigmoid function
- $x < 0 \rightarrow 0 \leq e^x < 1$ (True odds **less than** False)



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

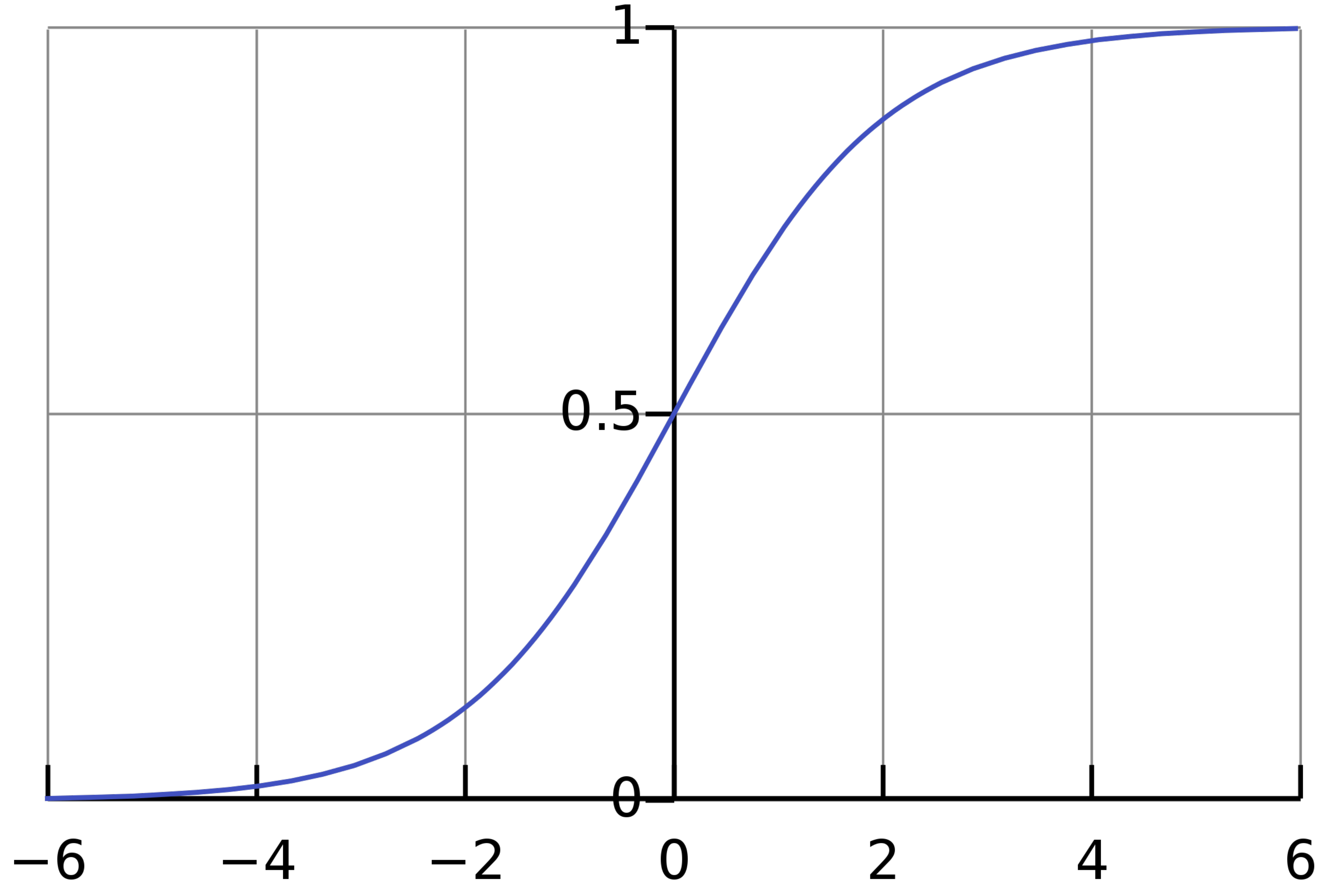
"True" odds

"True" odds

"False" odds

Sigmoid

- Recall:
  - Sigmoid takes in a **number** $[-\infty, \infty]$ and outputs a **probability** $[0.0,\ 1.0]$
  - Any number taken to the **power of zero** equals **one** $(a^0 = 1)$
- e^x represents the **odds of "True"** in the binary classification
- e^0 represents the **odds of "False"**
- $\sigma(x)$ is the "True" odds **divided by the sum of the odds**
  - (i.e. the **probability of True**)
- $x = 0$ represents the **default case**
  - $\sigma(0) = 1/(1+1) = 1/2$
  - This is the **"threshold"** of the Sigmoid function
- $x < 0 \rightarrow 0 \leq e^x < 1$ (True odds **less than** False)
- $x > 0 \rightarrow 1 < e^x \leq \infty$ (True odds **greater than** False)



$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{e^x}{e^x + e^0}$$

"True" odds

"True" odds

"False" odds

Softmax

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

odds of class i

summed odds of
all classes

Softmax

- Softmax is the **extension of Sigmoid** for classification into **more than 2 classes**
  - Input: a **vector** x
    - x_i is the **"score"** for class i
  - Output: a **vector of probabilities** p
    - p is the **same size** as x
    - p_i is the **probability** of class i
    - The probabilities **sum to 1.0** (this is a valid **probability distribution**)

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

odds of class i

summed odds of all classes

Softmax

- Softmax is the **extension of Sigmoid** for classification into **more than 2 classes**
  - Input: a **vector** x
    - x_i is the **"score"** for class i
  - Output: a **vector of probabilities** p
    - p is the **same size** as x
    - p_i is the **probability** of class i
    - The probabilities **sum to 1.0** (this is a valid **probability distribution**)
- When x is **all zeros**, the probability for each class is 1/N (N = size of x)

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

odds of class i

summed odds of all classes

Softmax

- Softmax is the **extension of Sigmoid** for classification into **more than 2 classes**
  - Input: a **vector** x
    - x_i is the **"score"** for class i
  - Output: a **vector of probabilities** p
    - p is the **same size** as x
    - p_i is the **probability** of class i
    - The probabilities **sum to 1.0** (this is a valid **probability distribution**)
- When x is **all zeros**, the probability for each class is 1/N (N = size of x)
- The input "scores" x are often called the **logits**

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

odds of class i

summed odds of all classes