Word Vectors (Word2Vec)

Ling 282/482: Deep Learning for Computational Linguistics
C.M. Downey
Fall 2025

Word Vectors, Intro

Distributional Similarity

- "You shall know a word by the company it keeps!" (*Firth, 1957*)

Distributional Similarity

- “You shall know a word by the company it keeps!” (*Firth, 1957*)
  - A bottle of *tezgüino* is on the table.

Distributional Similarity

- "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

- "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

- "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

- "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.
- Tezguino; corn-based alcoholic beverage.

Distributional Similarity

- How can we represent the “company” of a word?

Distributional Similarity

- How can we represent the “company” of a word?
- How can we make similar words have similar representations?

Why use word vectors?

- With words, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires exact same word to be in training and test
- One-hot vectors:
  - “terrible”: [0 0 0 0 0 0 1 0 0 0 … 0]
  - “Sparse” vectors
  - length = size of vocabulary
  - All words are as different from each other
    - e.g. “terrible” is as different from “bad” as from “awesome”

Why use word vectors?

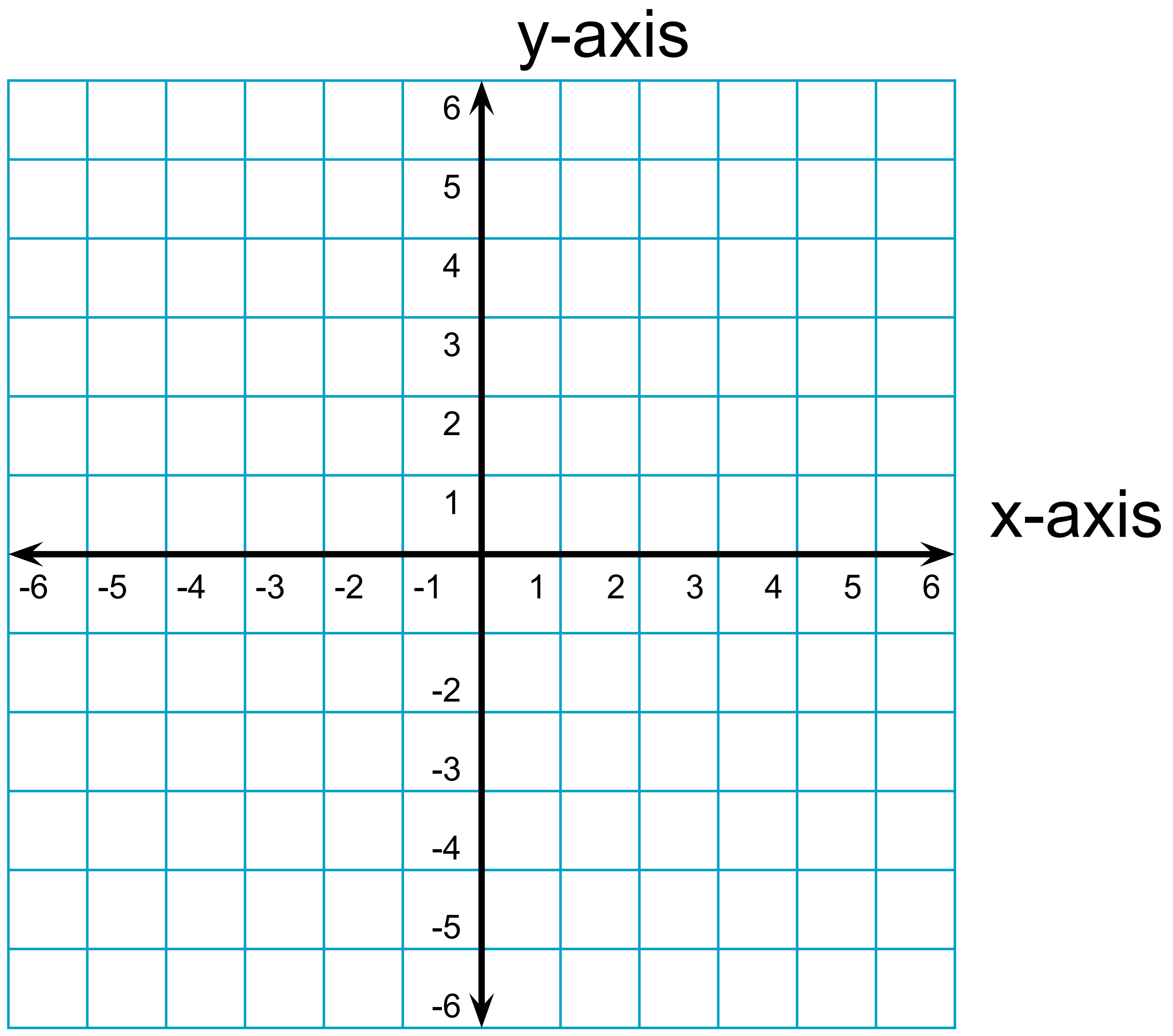
- With embeddings:
  - “Dense” vectors
  - “The previous word was vector [35,22,17, …]”
  - Now in the test set we might see a similar vector [34,21,14, …]
  - We can generalize to similar but unseen words!

Vectors as information

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

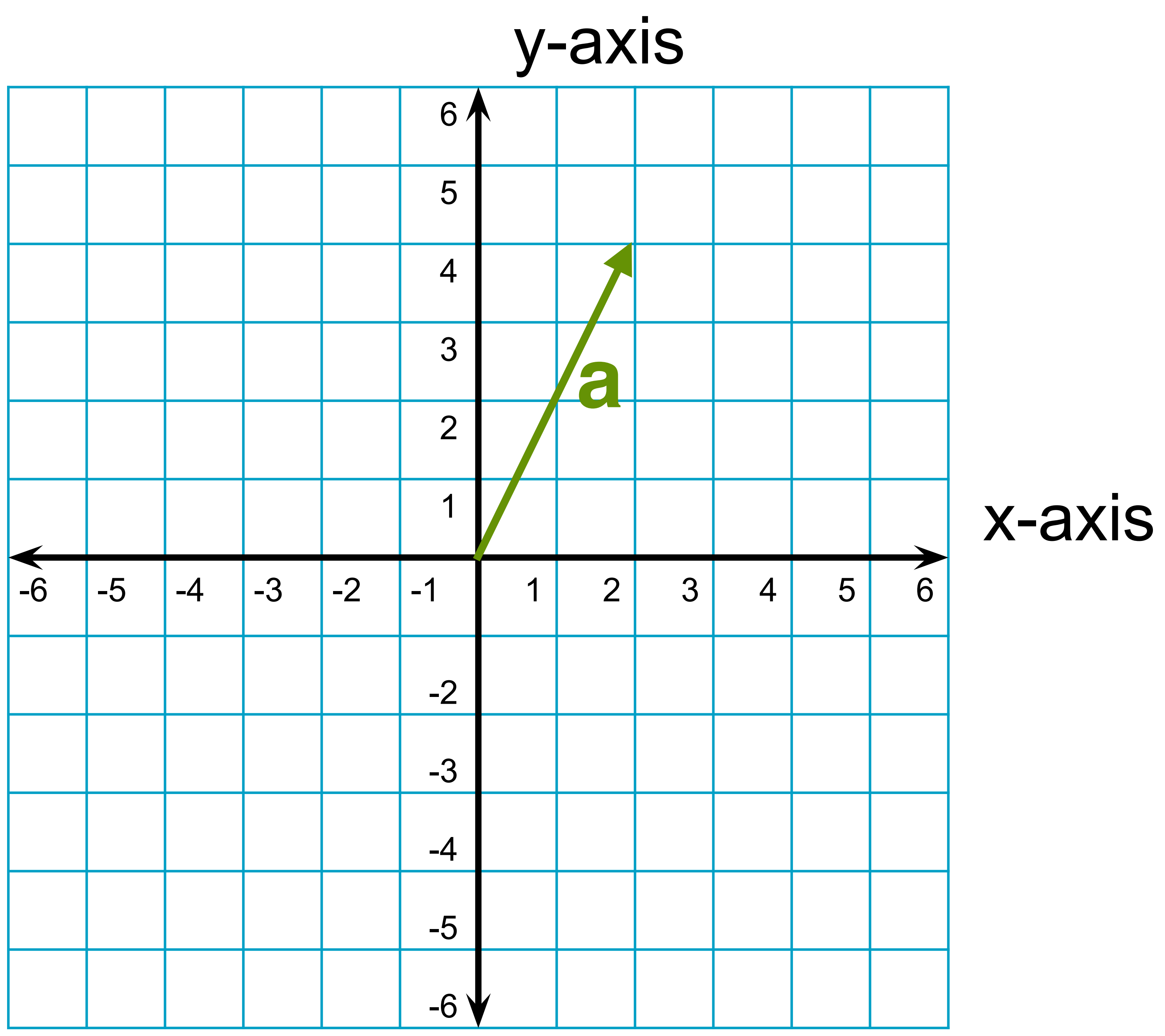
Vectors as information

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



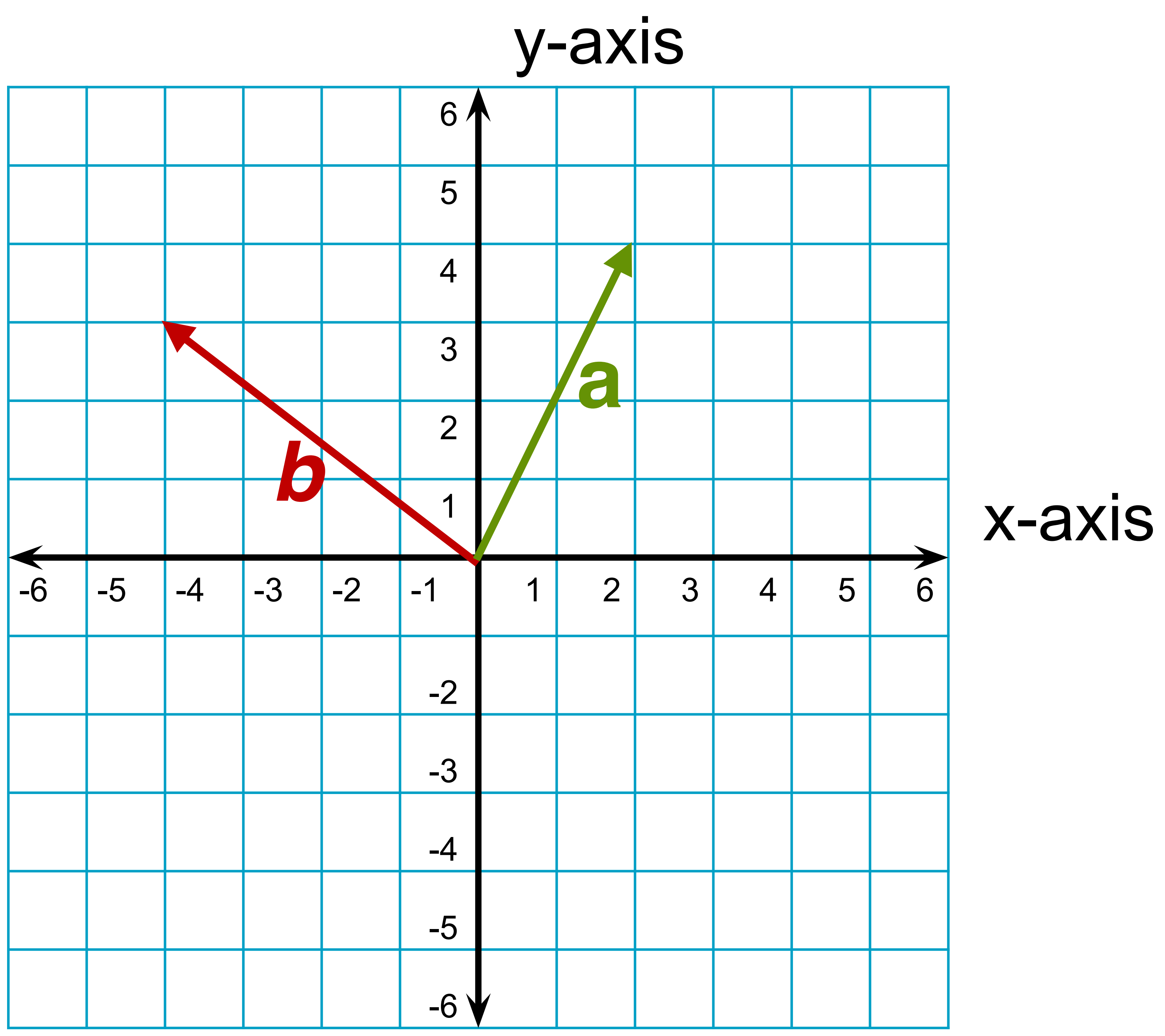
Vectors as information

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



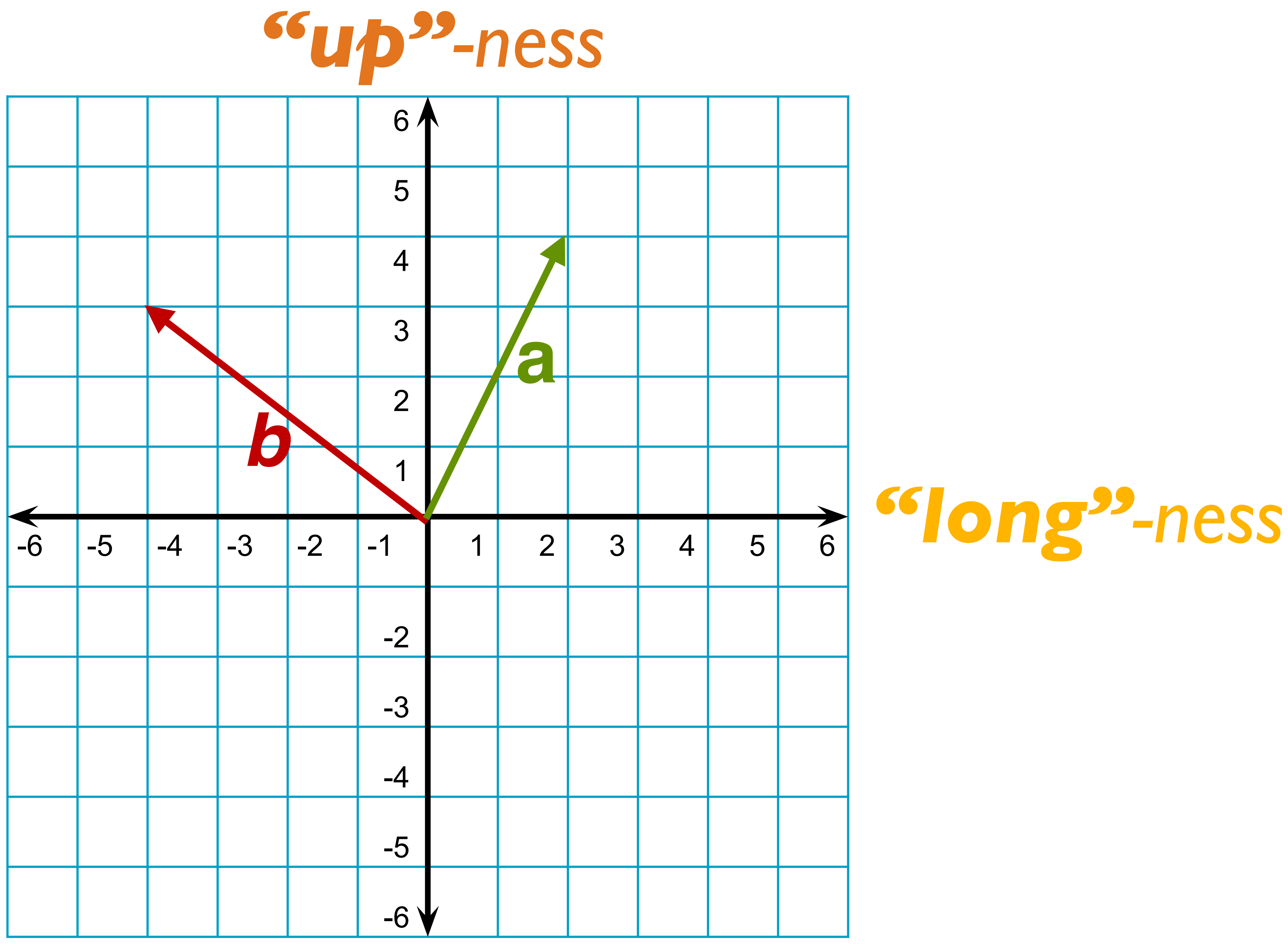
Vectors as information

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



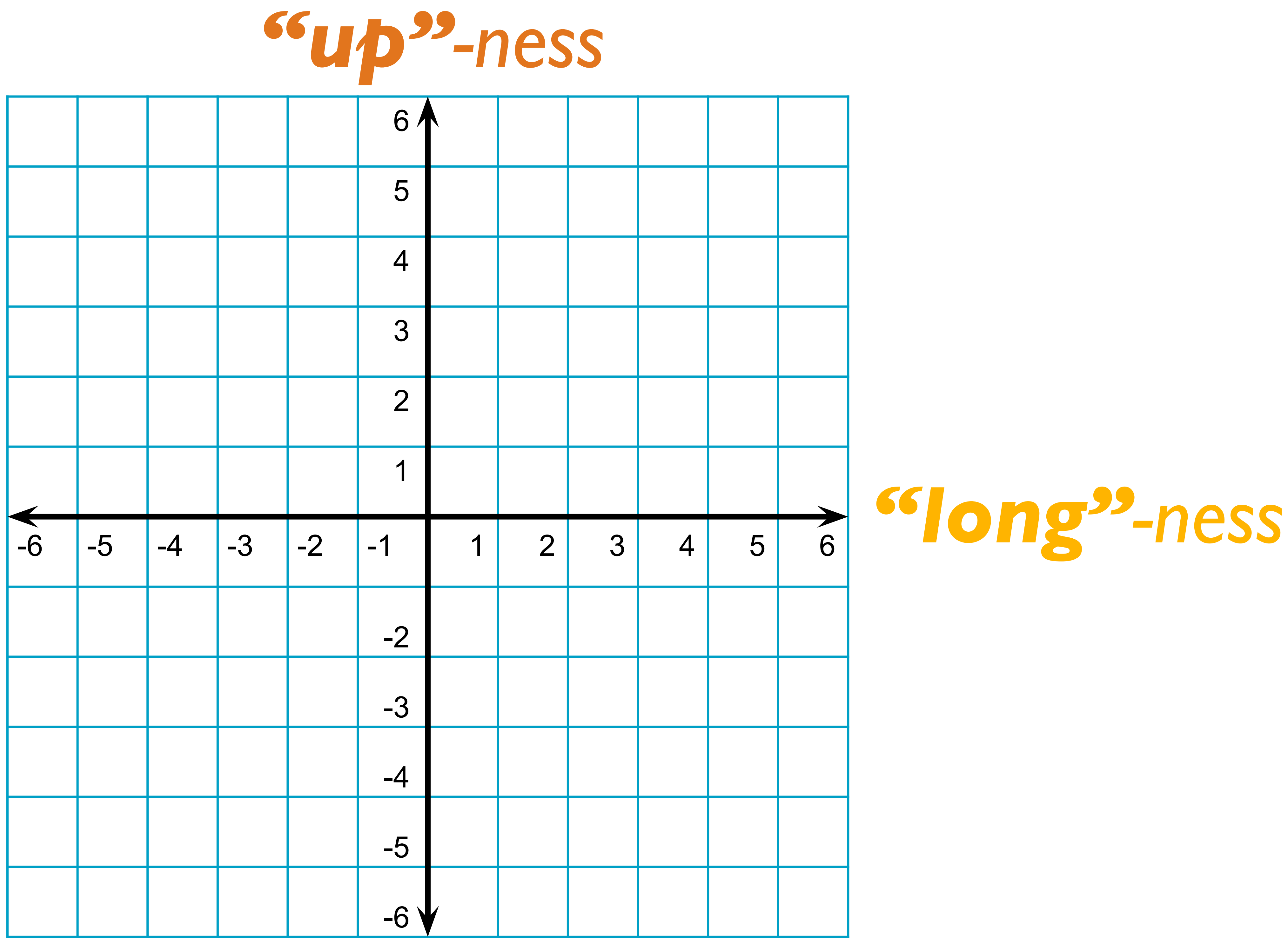
Vectors as information

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



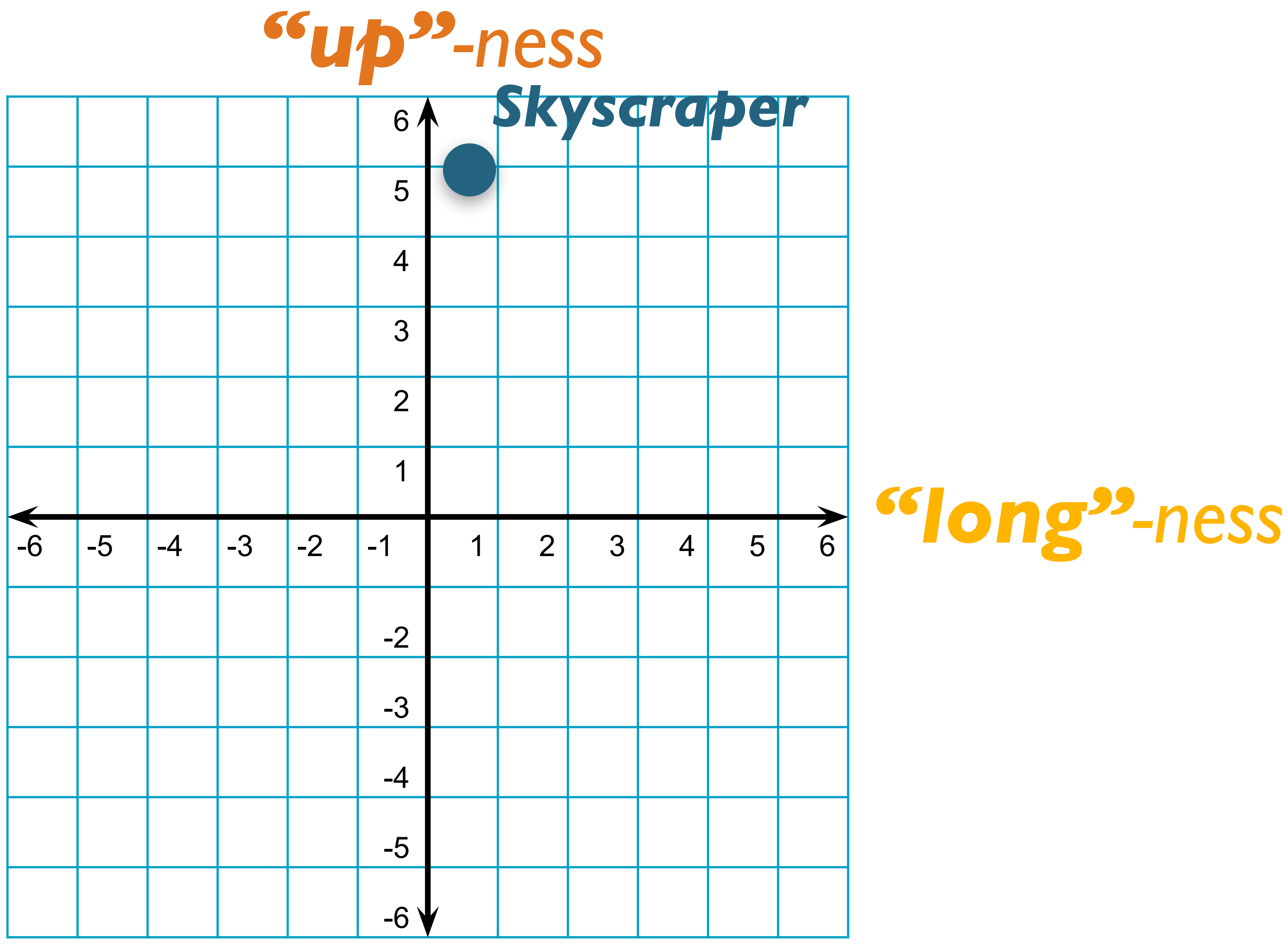
Vectors as information

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



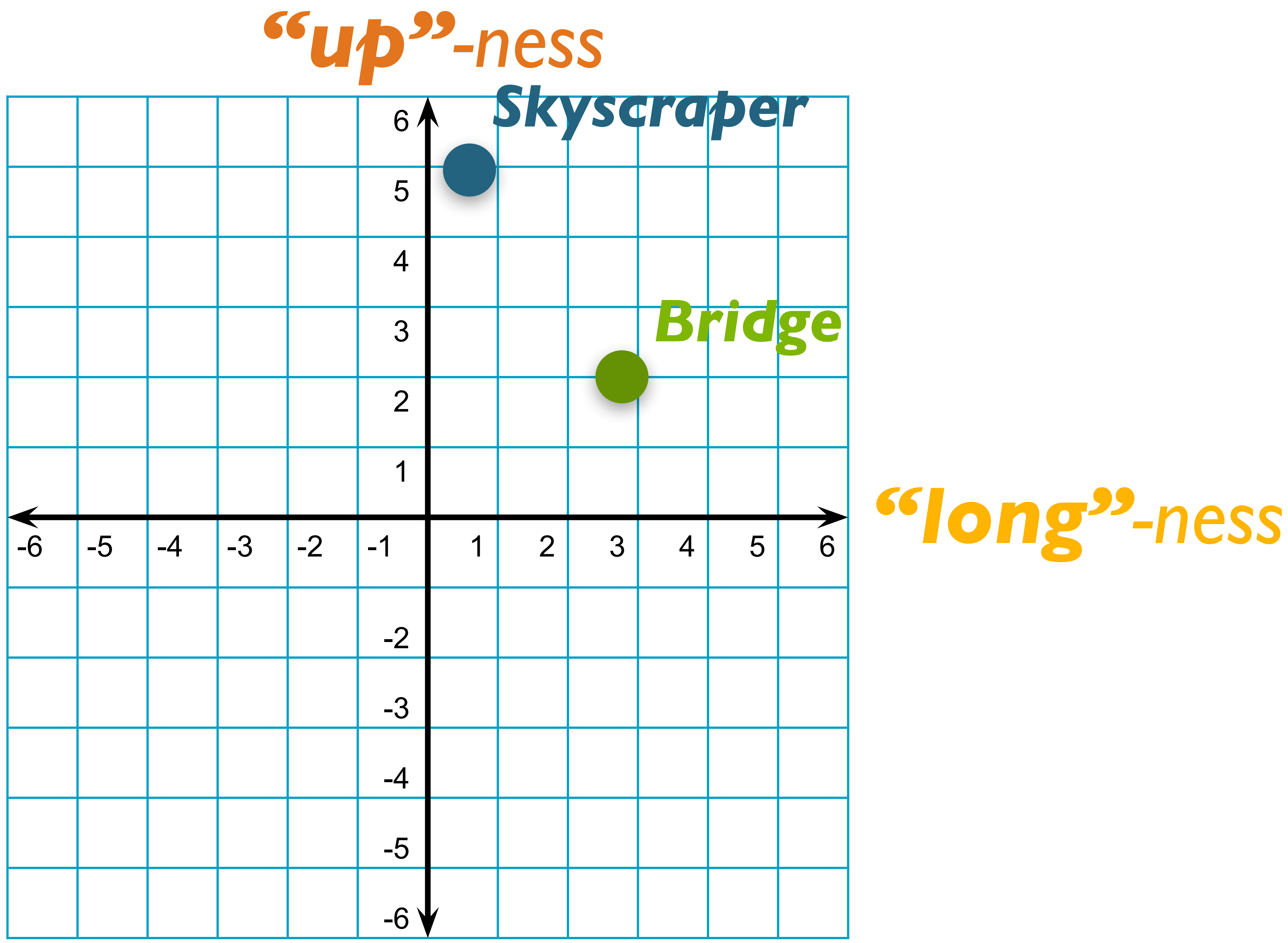
Vectors as information

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



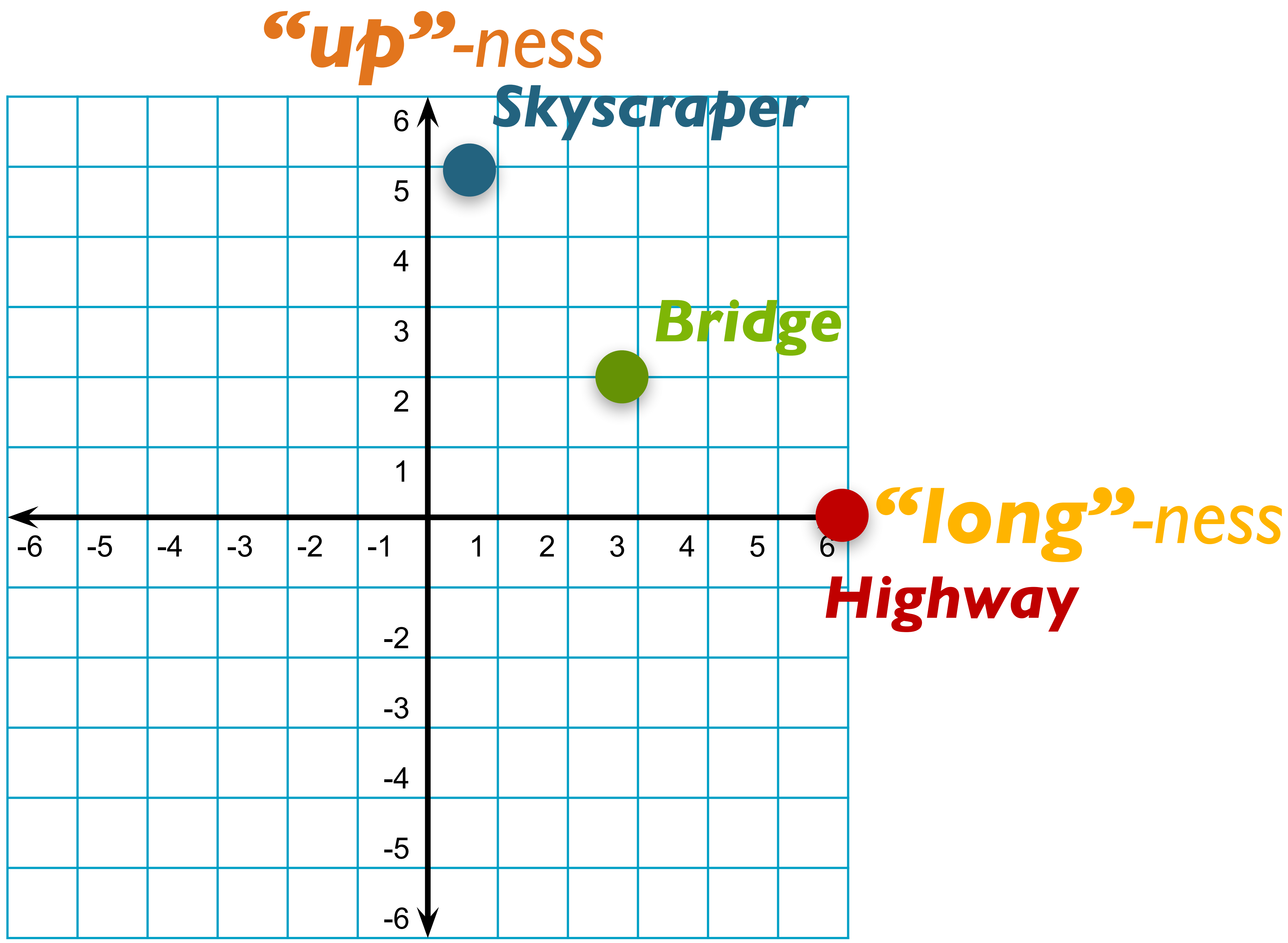
Vectors as information

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



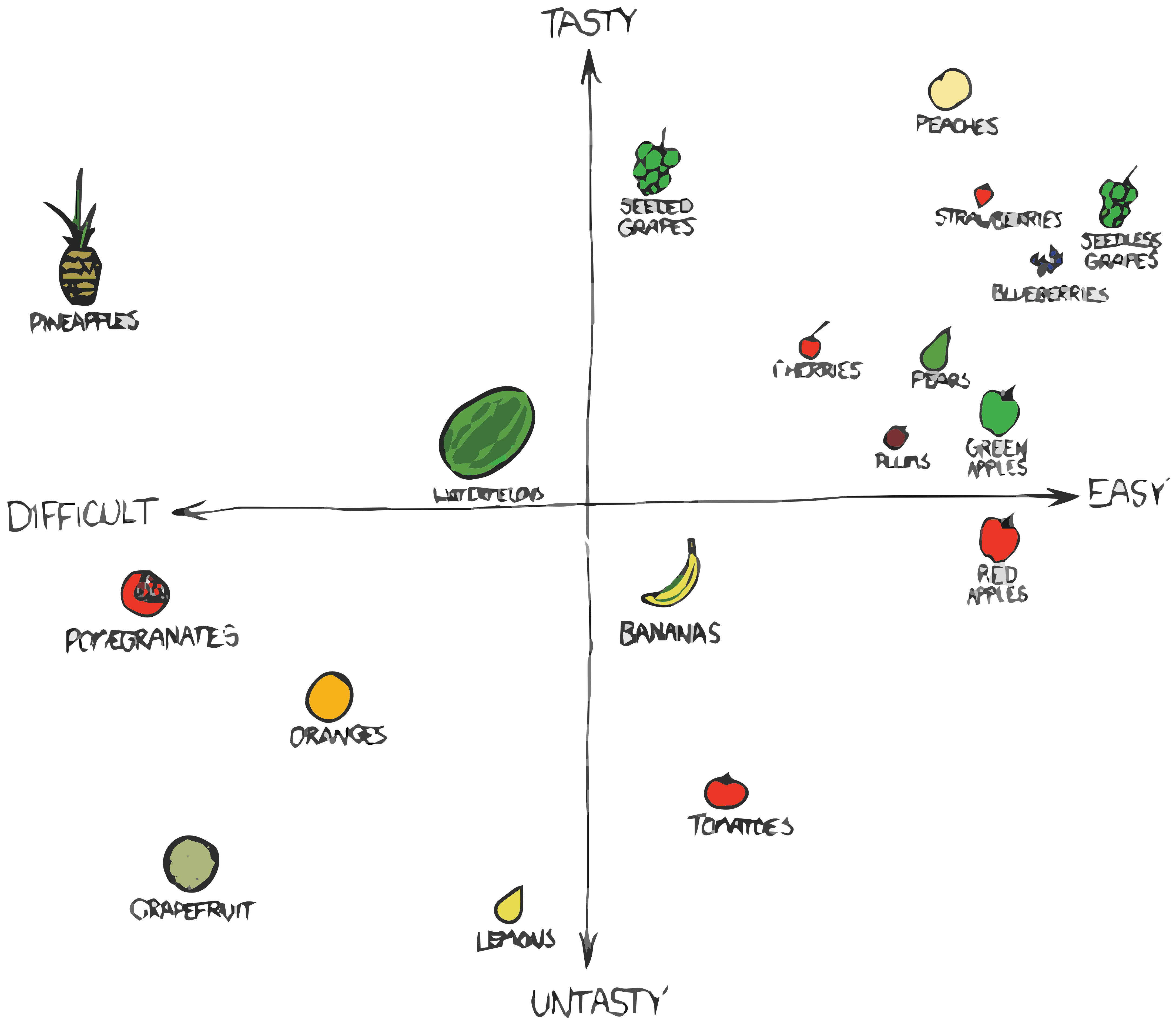
Vectors as information

- 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



Vector Length

- A vector's length is equal to the *square root of the dot product with itself*

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

Vector Distances: Manhattan & Euclidean

- **Manhattan Distance**
  - Distance as cumulative horizontal + vertical moves
- **Euclidean Distance**
  - Our normal notion of distance
- Both are too sensitive to extreme values

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

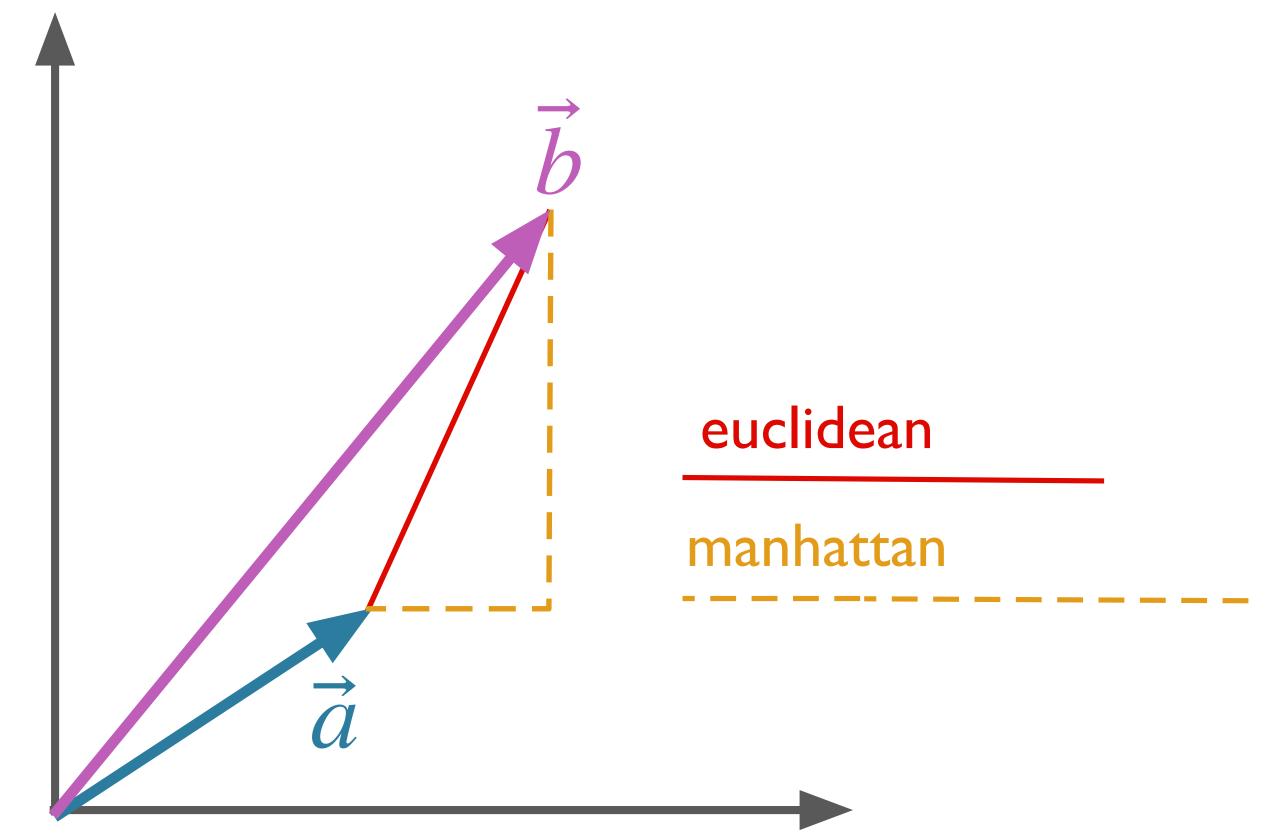
$$d_{\text{euclidian}}(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Vector Distances: Manhattan & Euclidean

- **Manhattan Distance**
  - Distance as cumulative horizontal + vertical moves
- **Euclidean Distance**
  - Our normal notion of distance
- Both are too 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

- Produces real number scalar from product of vectors' components
- Gives **higher similarity** to **longer** vectors

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

Vector Similarity: Cosine

- If you normalize the dot product for vector magnitude…
- …result is same as **cosine of angle** between the vectors

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

Bag of Words Vectors

- Represent ‘company’ of word such that similar words will have similar representations
  - ‘Company’ = context

Bag of Words Vectors

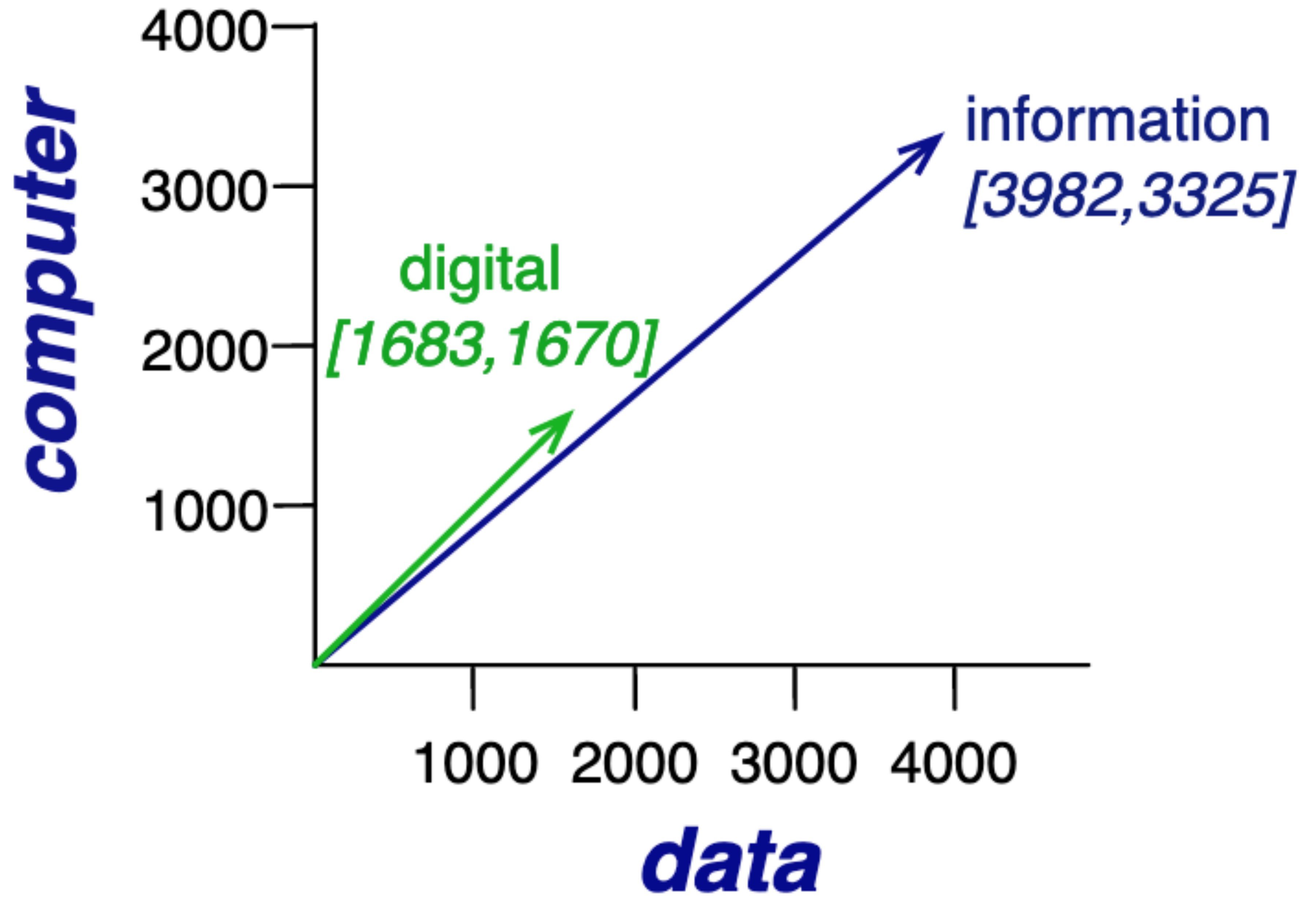
- Represent ‘company’ of word such that similar words will have similar representations
  - ‘Company’ = context
- Word represented by **context feature vector**

Bag of Words Vectors

- Represent ‘company’ of word such that similar words will have similar representations
  - ‘Company’ = context
- Word represented by **context feature vector**
- Initial representation:
  - “**Bag of words**” feature vector
  - Feature vector length N, where N is size of vocabulary
    - f_i+=1 if $word_i$ within window size w of *word*

Bag of Words Vectors

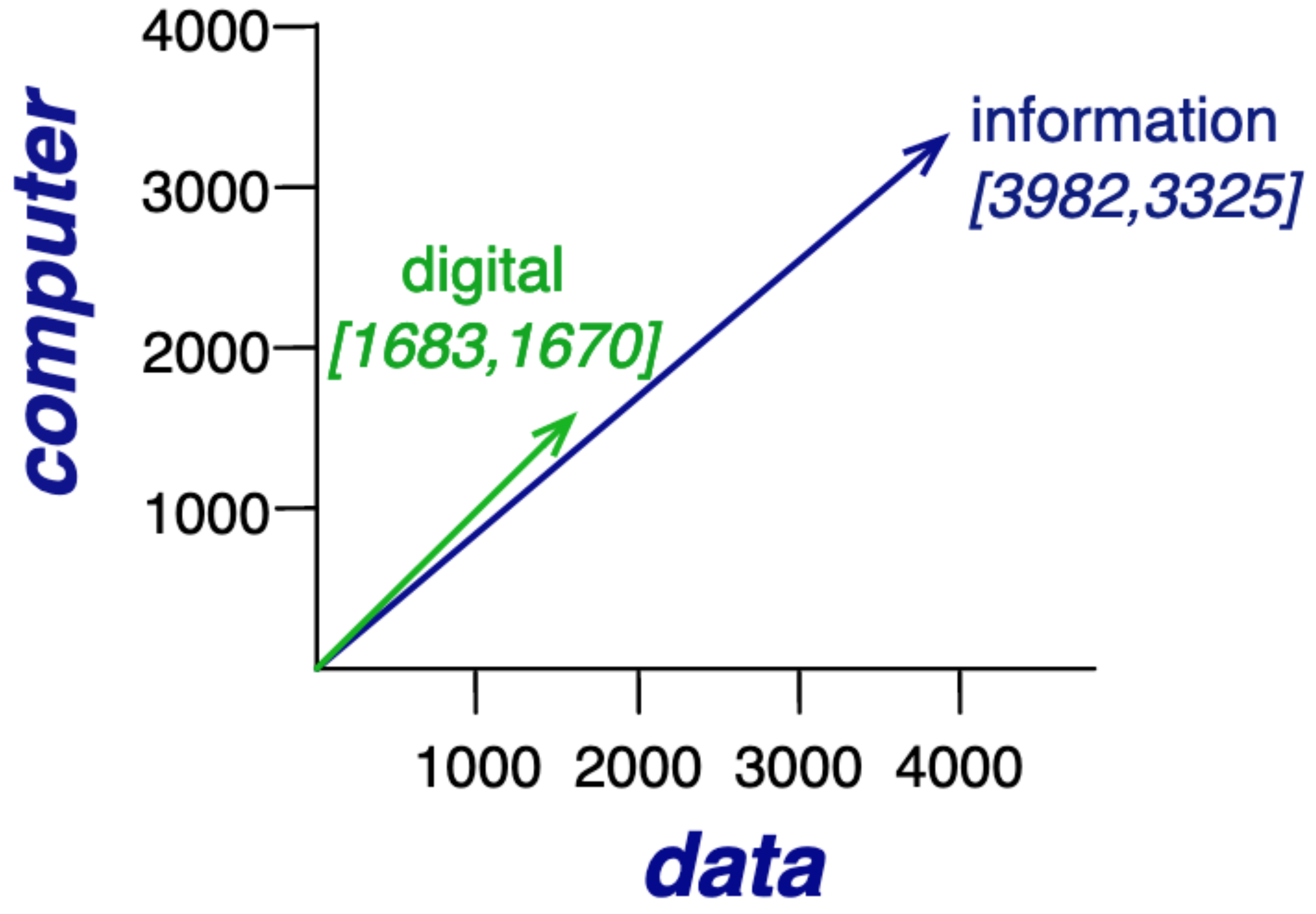
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Bag of Words Vectors

- Usually re-weighted by some algorithm
  - (e.g. tf-idf, ppmi)
- Still sparse
- Very high-dimensional: |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	...



Prediction-Based Models (Word2Vec)

Prediction-based Embeddings

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

Prediction-based Embeddings

- *Skip-gram* and *Continuous Bag of Words* (CBOW) models
- Intuition:
  - Words with **similar meanings** share **similar contexts**
  - Instead of counting:
    - Train models to **predict context words**
  - Models train embeddings that make current word more like nearby words and less like distance words

Embeddings:
Skip-Gram vs. Continuous Bag of Words

Embeddings: Skip-Gram vs. Continuous Bag of Words

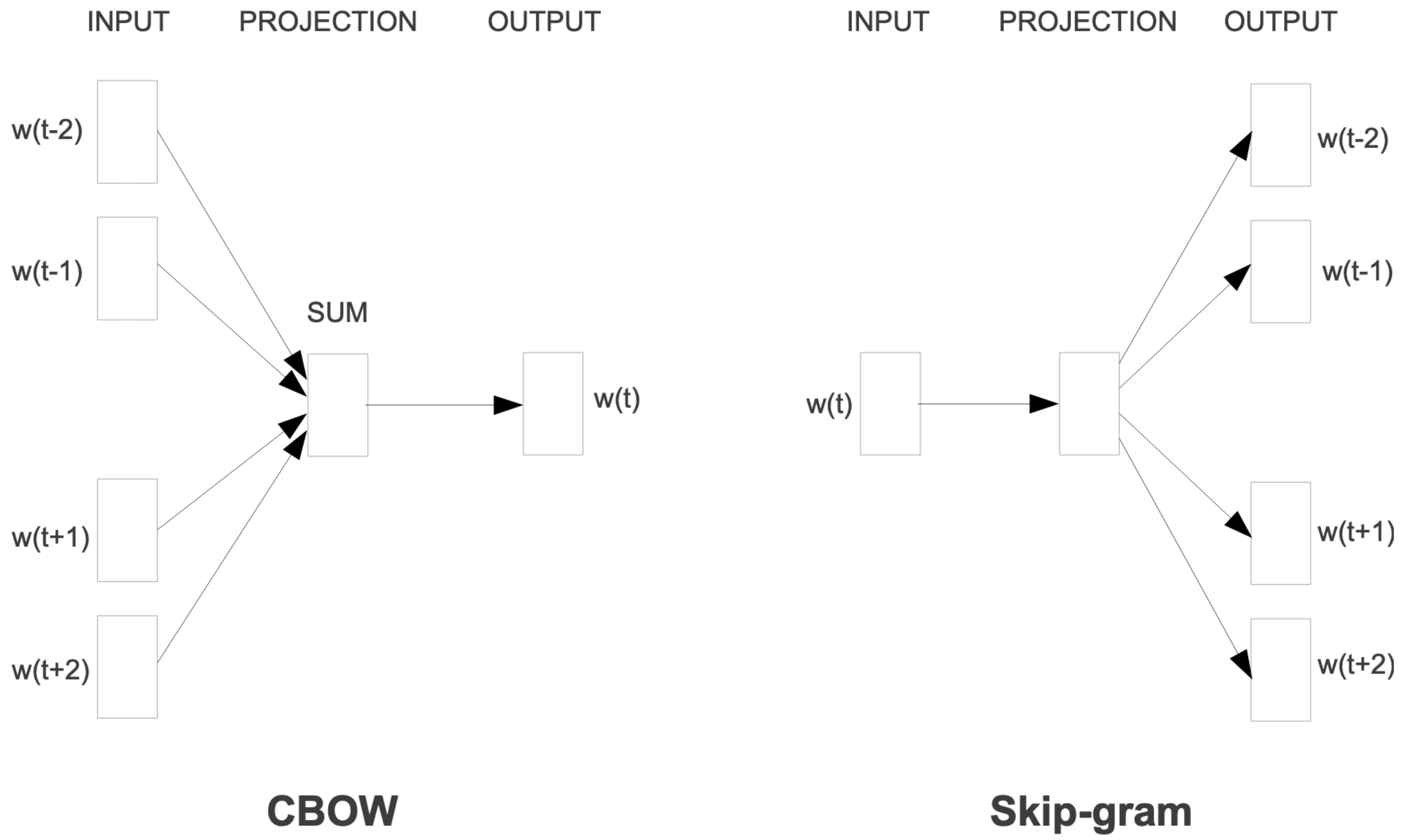
- Continuous Bag of Words (CBOW):
  - $P(\textit{word} \mid \textit{context})$
  - Input: $(w_{t-1}, w_{t-2}, w_{t+1}, wt_{+2} \ldots)$
  - Output: $p(w_t)$

Embeddings: Skip-Gram vs. Continuous Bag of Words

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

Embeddings: Skip-Gram vs. Continuous Bag of Words

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



[Mikolov et al 2013a](#) (the OG `word2vec` paper)

Skip-Gram Model

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

Skip-Gram Model

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

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

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 $p(w_k|w_j)$ as proportional to $\mathbf{c_k} \cdot \mathbf{w_j}$
    - For each context position
  - Convert to probability via softmax

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

Parameters and Hyper-parameters

- The embedding dimension is a *hyper-parameter*
  - Chosen by the modeler / practitioner
  - Not updated during the course of learning / training
  - Other examples we’ve seen so far:
    - Learning rate for SGD
  - 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

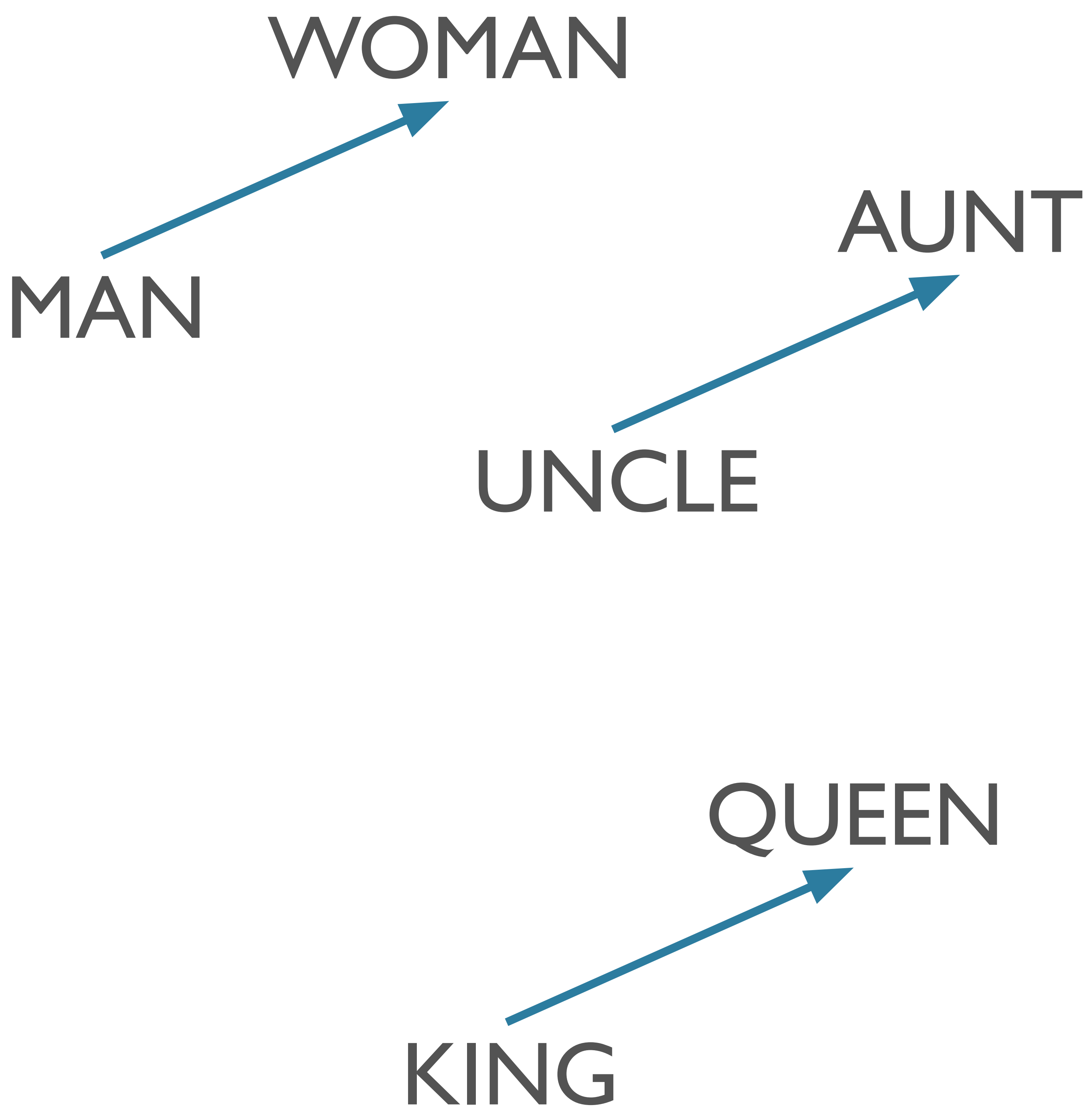
Power of Prediction-based Embeddings

- 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

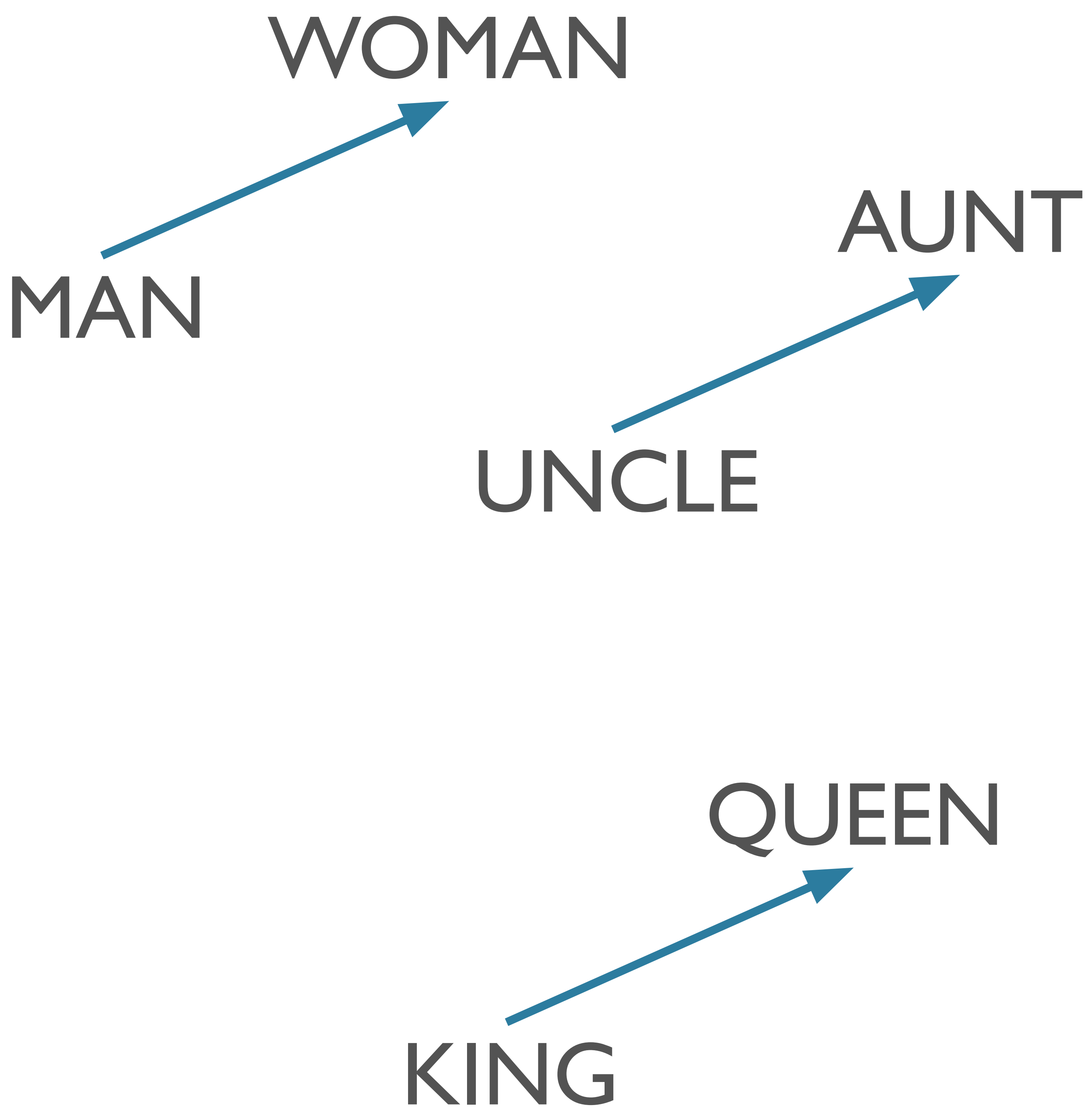
- Count-based embeddings:
  - Very high-dimensional (|V|)
  - Sparse
  - Pro: features are interpretable [“occurred with word W N times in corpus”]
- Prediction-based embeddings:
  - “Low”-dimensional (typically ~256-2048)
  - Dense
  - Con: features are not immediately interpretable
    - i.e. what does “dimension 36 has value -9.63” mean?

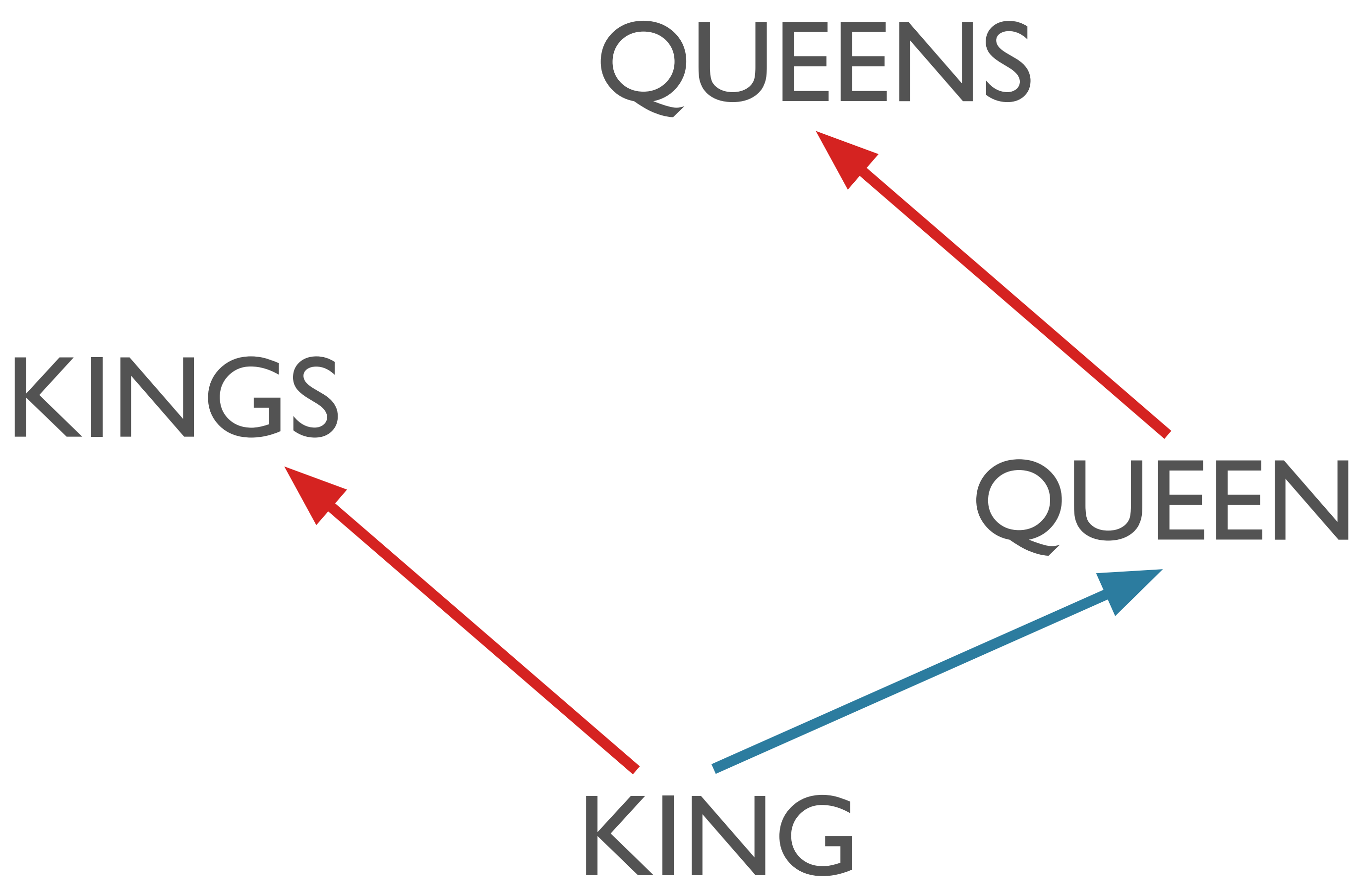
Relationships via Offsets



Mikolov et al 2013b

Relationships via Offsets





[Mikolov et al 2013b]

One More Example

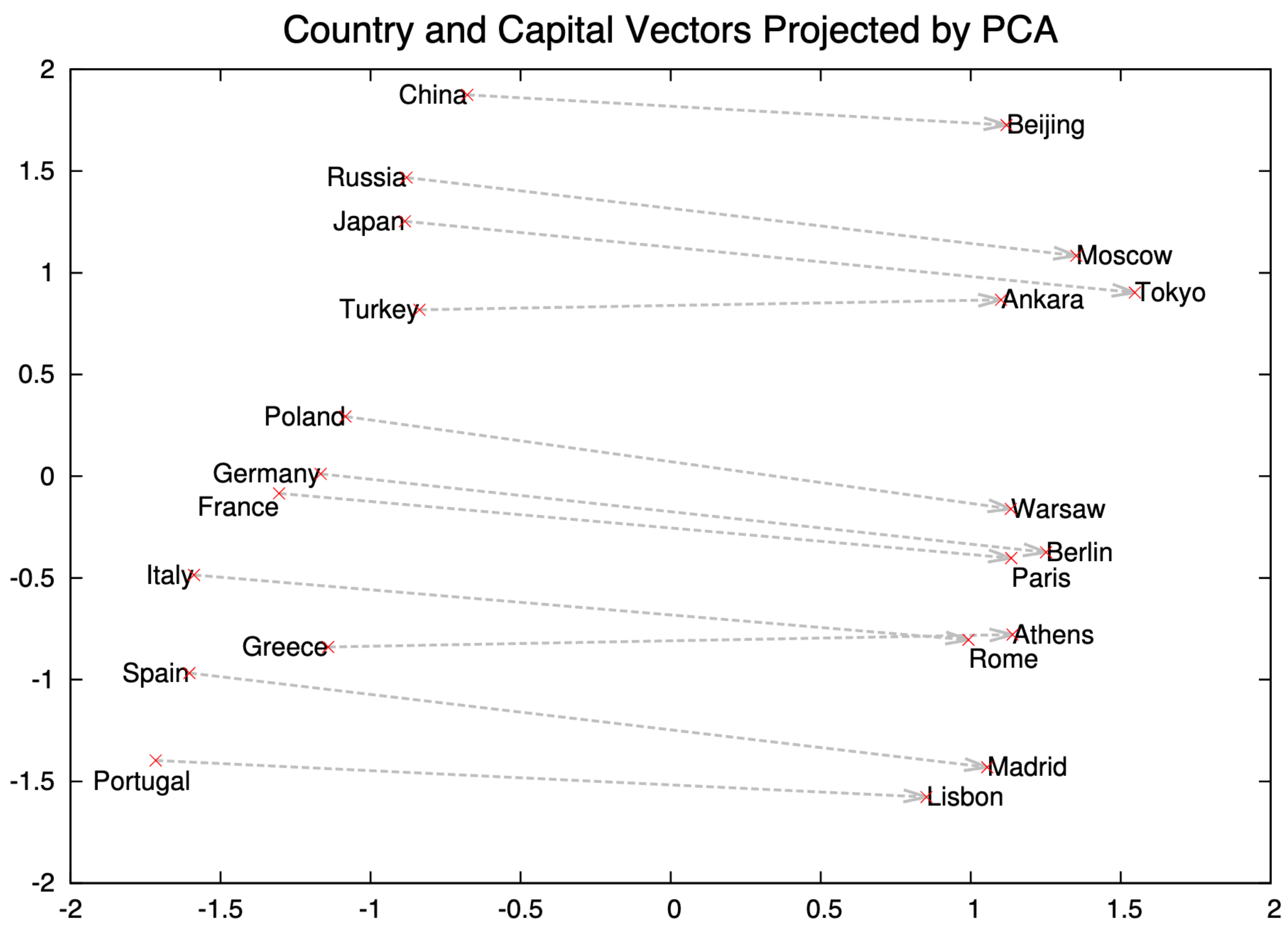
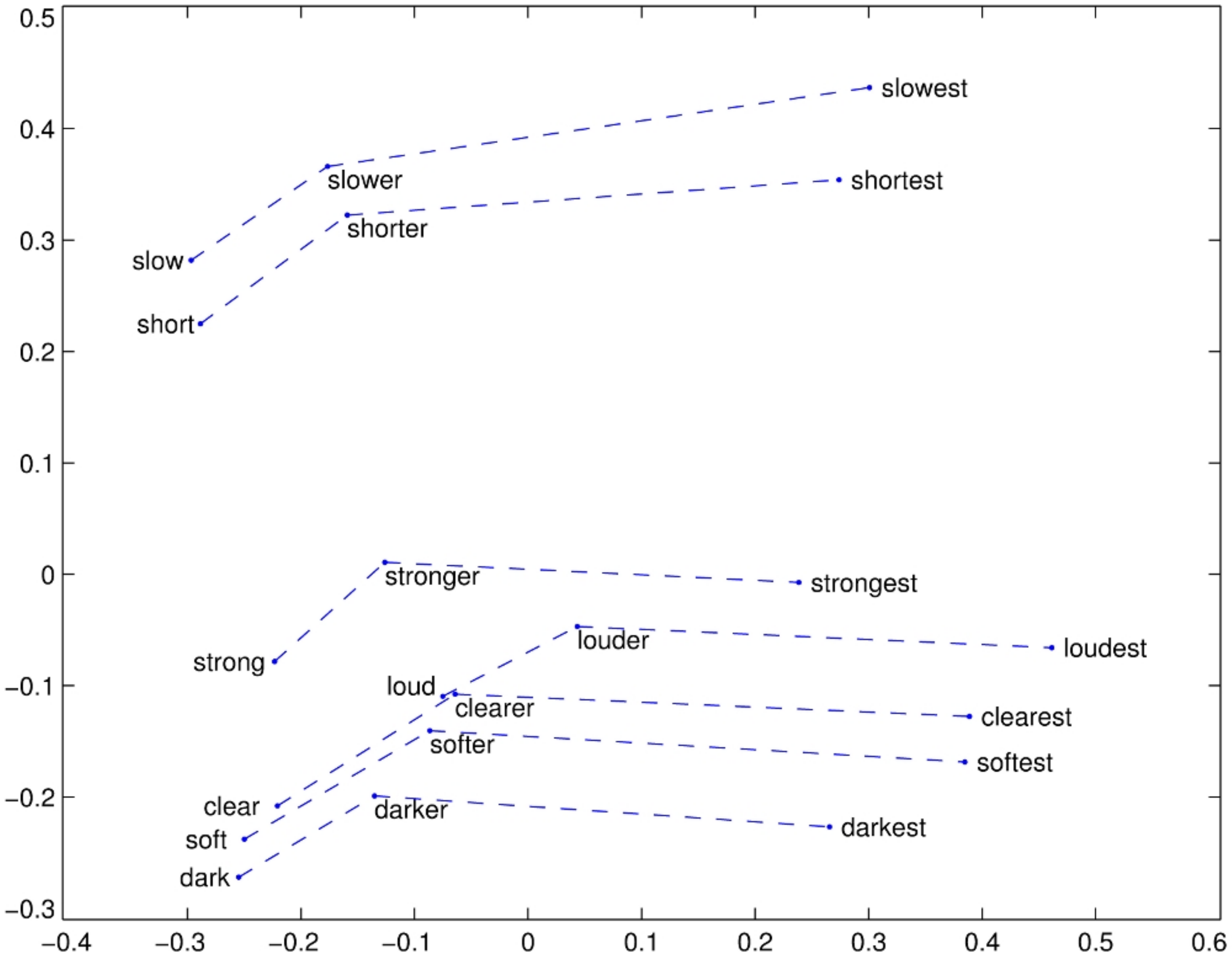


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised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

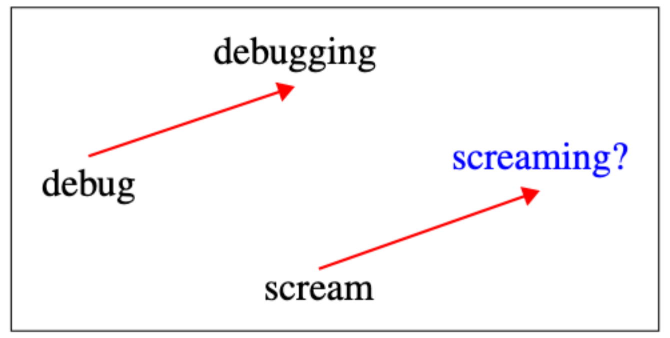


Figure 1: Using the vector offset method to solve the analogy task (Mikolov et al., 2013c).

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
- Strategy:
  - Approximate by *negative sampling* (efficient approximation to Noise Contrastive Estimation)
    - + example: true context word
    - – example: *k* other words, randomly sampled

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

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

Generating Positive Examples

- Iterate through the corpus

Generating Positive Examples

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

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

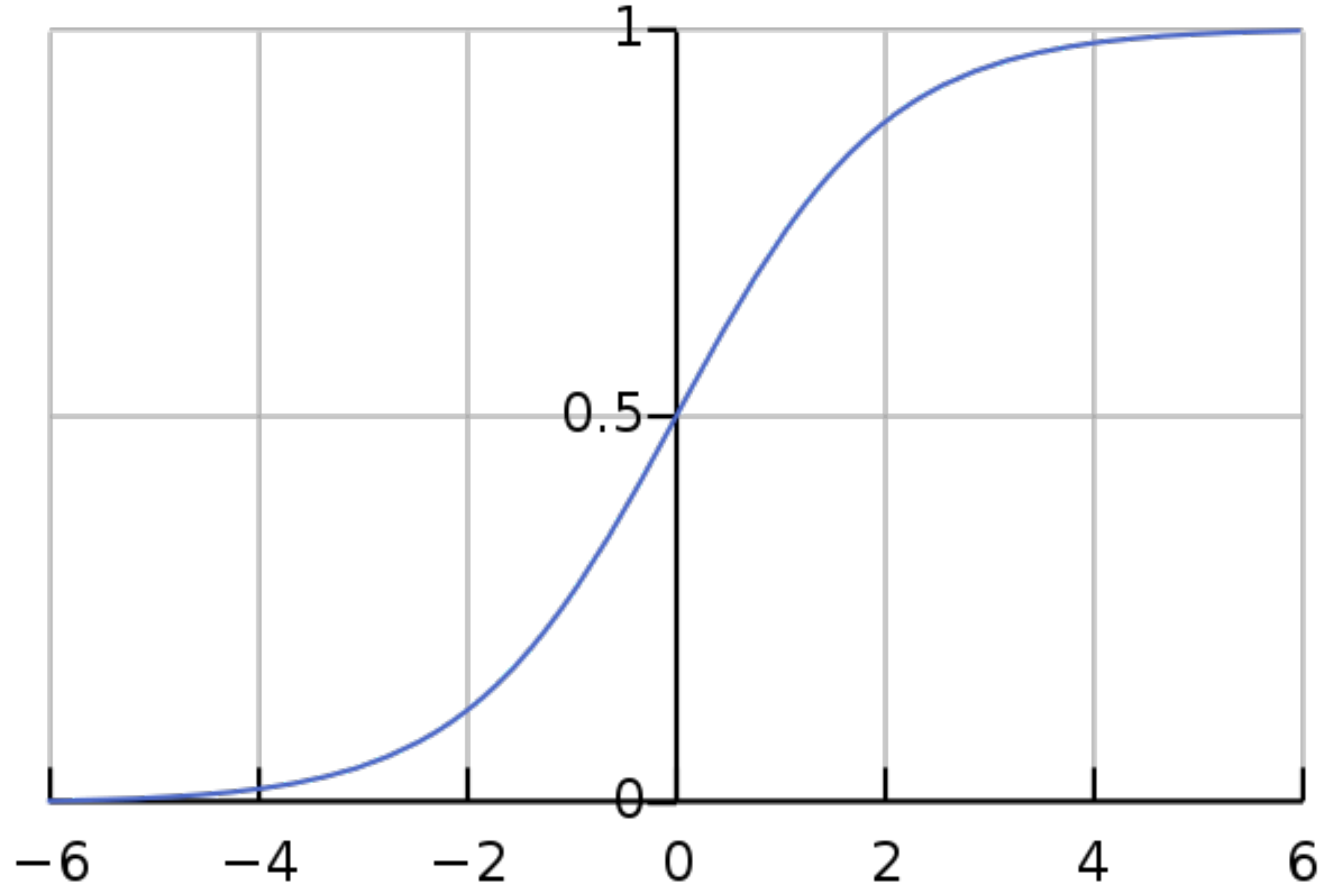
sigmoid

Target word embedding

Context word embedding

Similarity (dot-product)

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



The Model

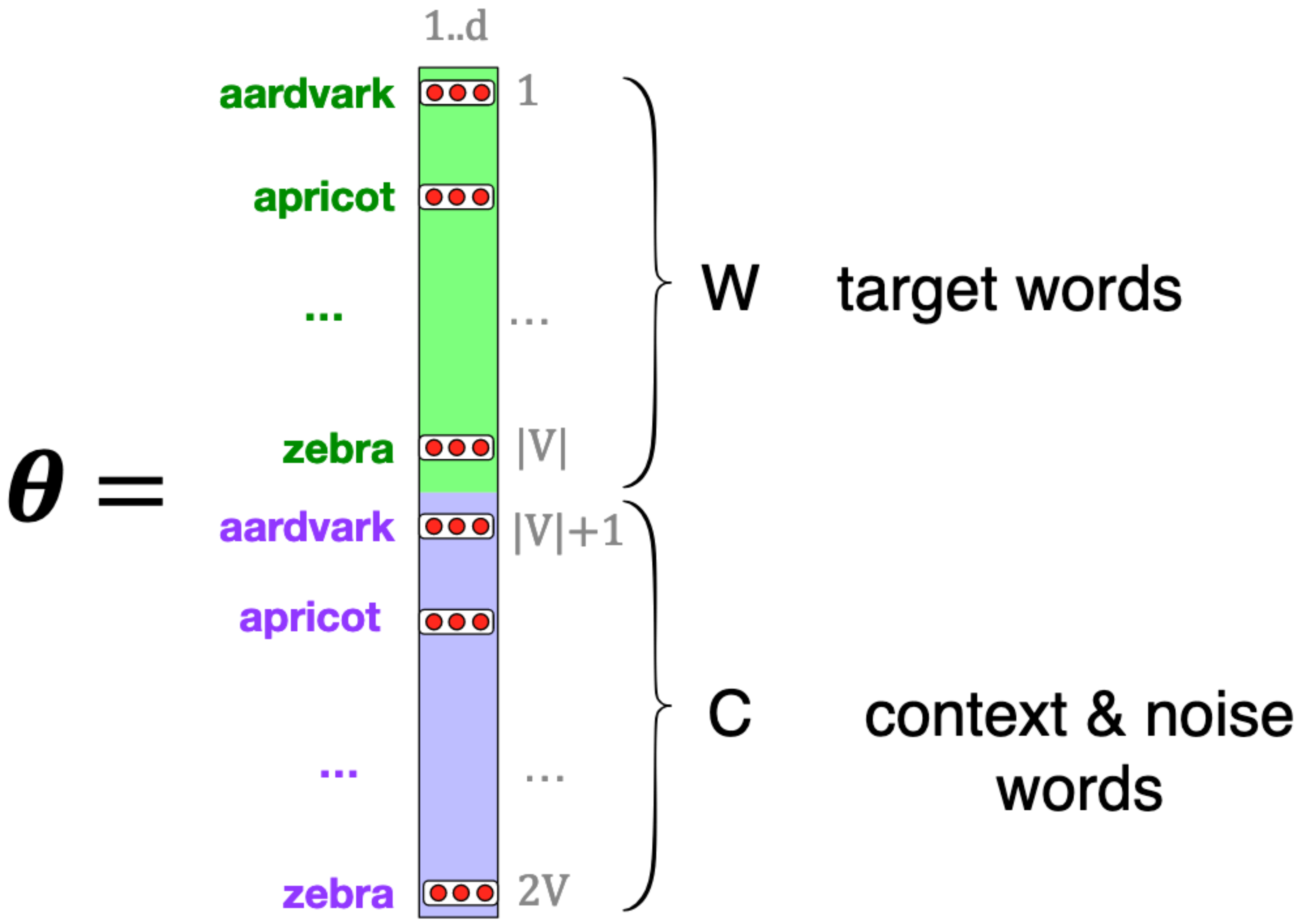
- Target and context words that are **more similar** to each other (have more similar embeddings) have a **higher probability** of being a positive example

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

Learning

- What are the parameters?
- What is the loss?

Learning: Parameters



Learning: Loss

Learning: Loss

- We want our model to:

Learning: Loss

- We want our model to:
  - Assign high $P(1 \mid w, c_+)$ (c+ is a positive context word)

Learning: Loss

- 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)

Learning: Loss

- 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 $P(0 \mid w, c_-)$

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:

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

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

$$L_{CE} = -\log P(1,0,0,\ldots,0 \mid w, c_+, c_{-1}, c_{-2}, \ldots, c_{-k})$$

$$= -\log\left(P(1 \mid w, c_+)\prod_{i=1}^{k} P(0 \mid w, c_{-i})\right)$$

$$= -\log P(1 \mid w, c_+) - \sum_{i=1}^{k} \log P(0 \mid w, c_{-i})$$

Learning: Intuitively