

# Gradient Descent; Word Vectors

LING 575K Deep Learning for NLP

Shane Steinert-Threlkeld

March 31 2021

# Announcements

- Office hours:
  - Shane: Wed 3-5PM [e.g. later today]
  - Agatha:
    - Tuesday 4-5PM
    - Wednesday 11:15AM-12:15PM
    - [NB: this week, Thursday at 12PM]
    - See Canvas announcement for Zoom password

# JD Says “Hi”



# Today's Plan

- Terminology / Notation
- Gradient Descent
- Word Vectors, intro
- Homework 1

# Basic Terminology / Notation

# Supervised Learning

# Supervised Learning

- Given: a dataset  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ 
  - $x_i \in X$ : input for i-th example
  - $y_i \in Y$ : output for i-th example

# Supervised Learning

- Given: a dataset  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ 
  - $x_i \in X$ : input for i-th example
  - $y_i \in Y$ : output for i-th example
- For example:
  - Sentiment analysis:
    - Input: bag of words representation of “This movie was great.”
    - Output: 4 [on a scale 1-5]
  - Language modeling:
    - Input: “This movie was”
    - Output: “great”

# Supervised Learning

# Supervised Learning

- Given: a dataset  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ 
  - $x_i \in X$ : input for i-th example
  - $y_i \in Y$ : output for i-th example

# Supervised Learning

- Given: a dataset  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ 
  - $x_i \in X$ : input for i-th example
  - $y_i \in Y$ : output for i-th example
- Goal: *learn* a function  $f: X \rightarrow Y$  which:
  - “Does well” on the given data  $\mathcal{D}$
  - Generalizes well to unseen data

# Parameterized Functions

- A learning algorithm searches for a function  $f$  amongst a space of possible functions
- Parameters define a family of functions
  - $\theta$ : general symbol for parameters
  - $\hat{y} = f(x; \theta)$ : input  $x$ , parameters  $\theta$ ; model/function output  $\hat{y}$
- Example: the family of linear functions  $f(x) = mx + b$ 
  - $\theta = \{m, b\}$
- Later: neural network architecture defines the family of functions

# Loss Minimization

- General form of optimization problem
- $\mathcal{L}(\hat{y}, y)$ : loss function (“objective function”);  $\mathcal{L}(\hat{Y}, Y) = \frac{1}{|Y|} \sum_i \ell(\hat{y}(x_i), y_i)$ 
  - How “close” is the model’s output to the true output
  - $\ell(\hat{y}, y)$ : local (per-instance) loss, averaged over training instances
  - More later: depends on the particular task, among other things
- View the loss as a *function of the model’s parameters*

$$\mathcal{L}(\theta) := \mathcal{L}(\hat{Y}, Y) = \mathcal{L}(f(X; \theta), Y)$$

# Loss Minimization

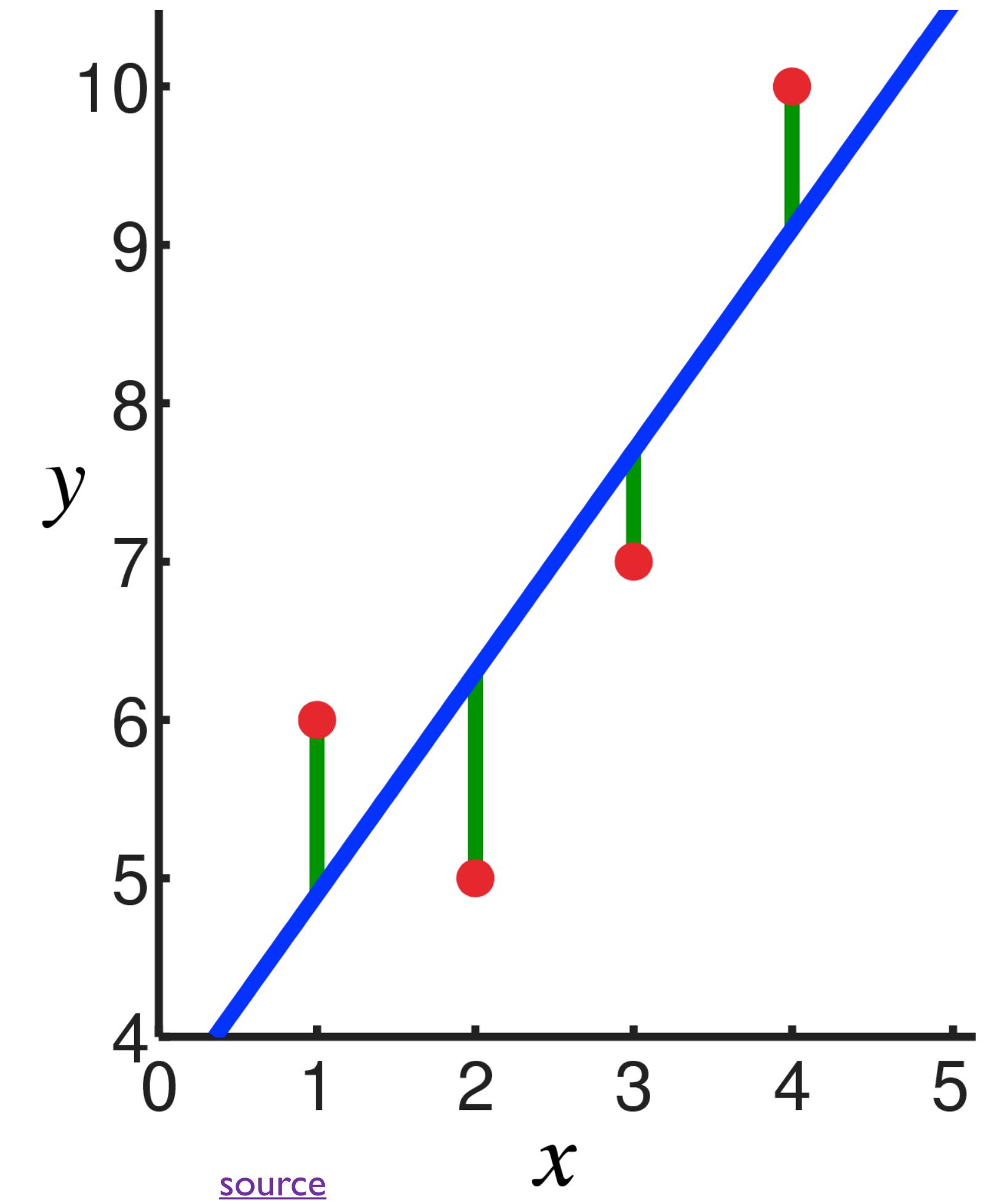
- The optimization problem:

$$\theta^* = \arg \min_{\theta} \mathcal{L}(\theta)$$

- Example: (least-squares) linear regression

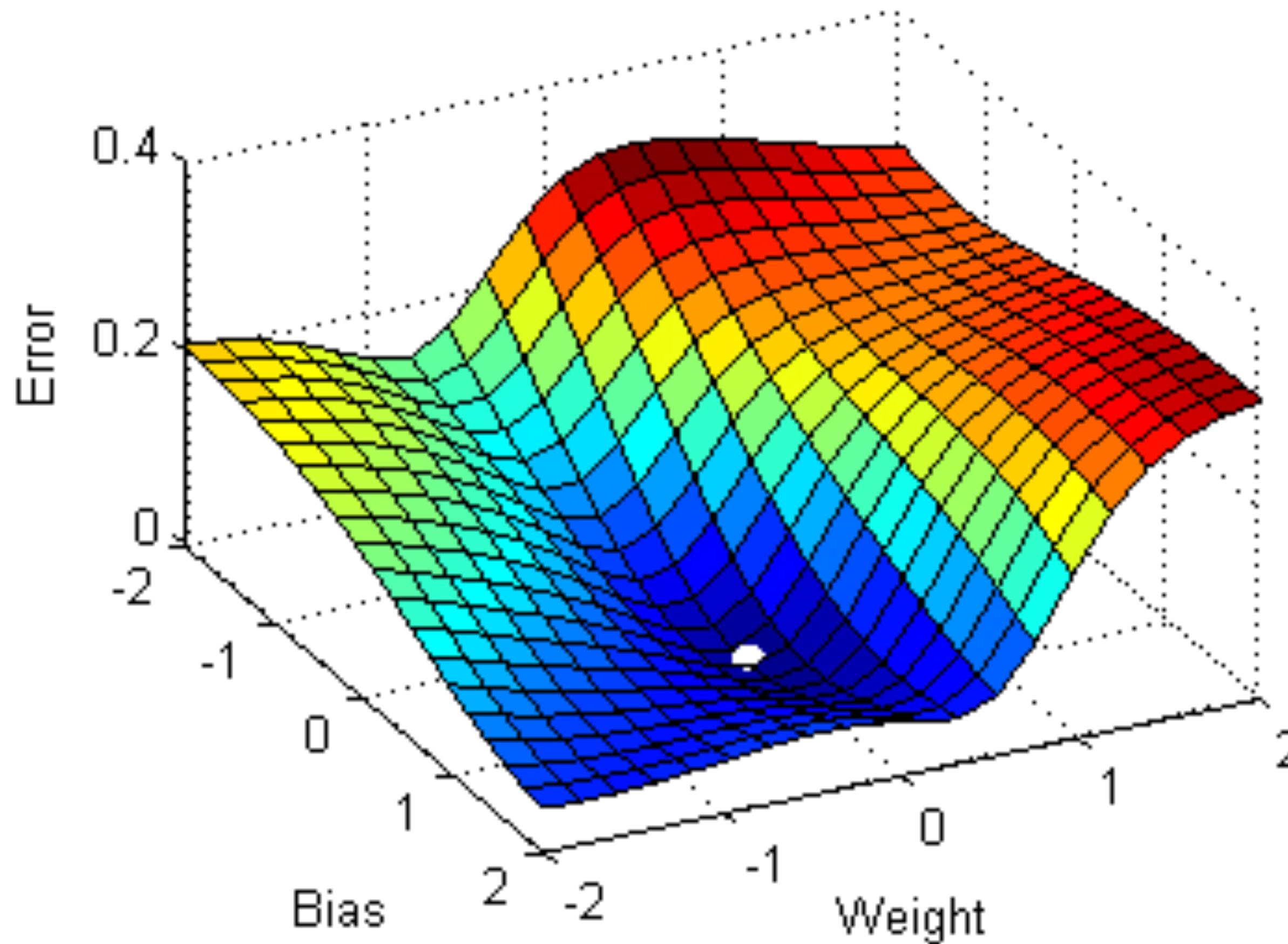
- $\ell(\hat{y}, y) = (\hat{y} - y)^2$

$$m^*, b^* = \arg \min_{m,b} \sum_i ((mx_i + b) - y_i)^2$$

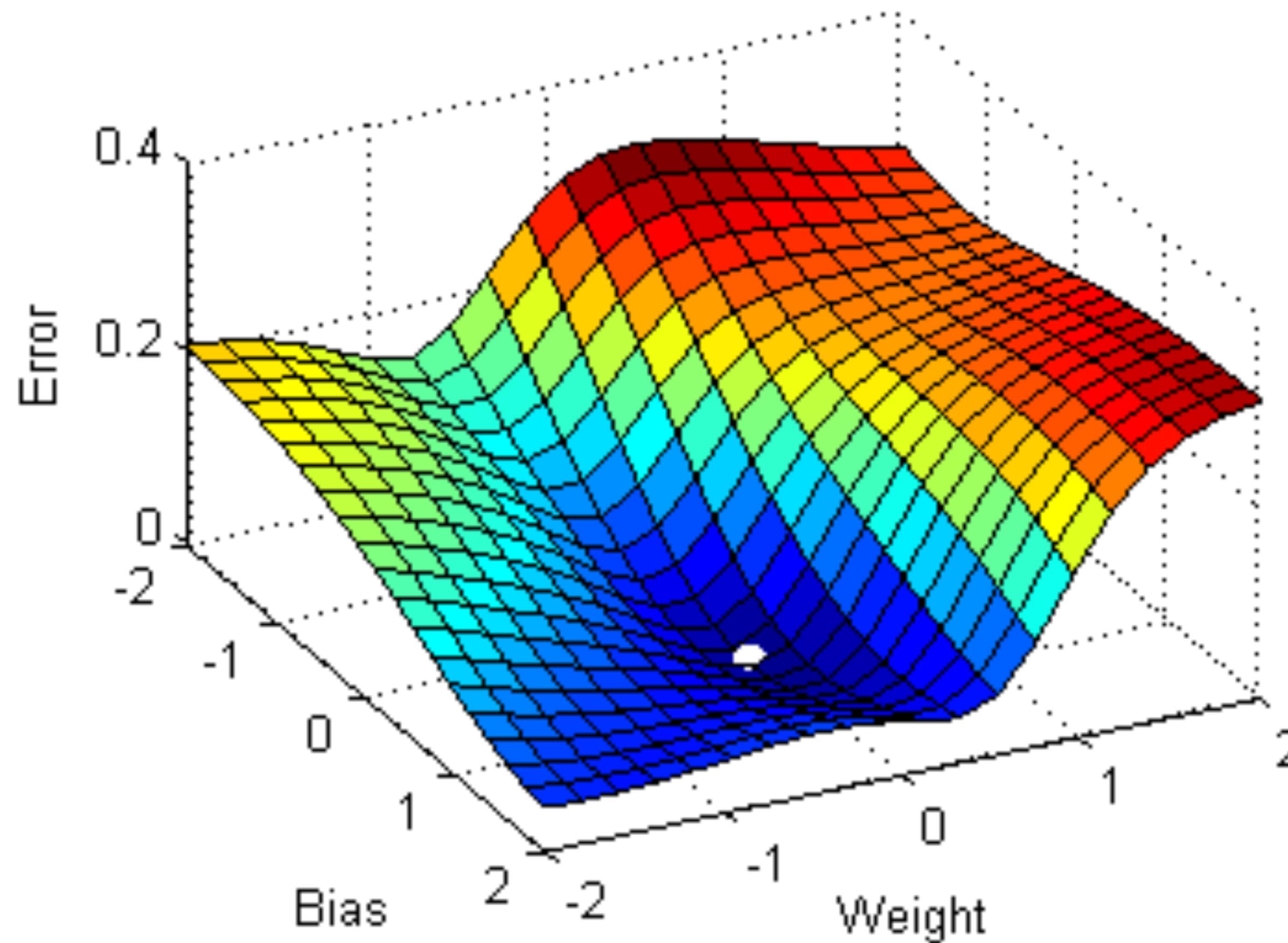


# Learning: (Stochastic) Gradient Descent

# Gradient Descent: Basic Idea



# Gradient Descent: Basic Idea



# Gradient Descent: Basic Idea

- The *gradient* of the loss w/r/t parameters tells which direction in parameter space to “walk” to make the loss smaller (i.e. to improve model outputs)
- Guaranteed to work in linear model case
  - Can get stuck in local minima for non-linear functions, like NNs
  - [More precisely: if loss is a *convex* function of the parameters, gradient descent is guaranteed to find an optimal solution. For non-linear functions, the loss will generally *not* be convex.]

# Derivatives

- The derivative of a function of one real variable measures how much the output changes with respect to a change in the input variable

# Derivatives

- The derivative of a function of one real variable measures how much the output changes with respect to a change in the input variable

$$f(x) = x^2 + 35x + 12$$

$$\frac{df}{dx} = 2x + 35$$

# Derivatives

- The derivative of a function of one real variable measures how much the output changes with respect to a change in the input variable

$$f(x) = x^2 + 35x + 12$$

$$\frac{df}{dx} = 2x + 35$$

$$f(x) = e^x$$

$$\frac{df}{dx} = e^x$$

# Partial Derivatives

- A partial derivative of a function of several variables measures its derivative with respect one of those variables, with the others held constant.

# Partial Derivatives

- A partial derivative of a function of several variables measures its derivative with respect one of those variables, with the others held constant.

$$f(x) = 10x^3y^2 + 5xy^3 + 4x + y$$

# Partial Derivatives

- A partial derivative of a function of several variables measures its derivative with respect one of those variables, with the others held constant.

$$f(x) = 10x^3y^2 + 5xy^3 + 4x + y$$

$$\frac{\partial f}{\partial x} = 30x^2y^2 + 5y^3 + 4$$

# Partial Derivatives

- A partial derivative of a function of several variables measures its derivative with respect one of those variables, with the others held constant.

$$f(x) = 10x^3y^2 + 5xy^3 + 4x + y$$

$$\frac{\partial f}{\partial x} = 30x^2y^2 + 5y^3 + 4$$

$$\frac{\partial f}{\partial y} = 20x^3y + 15xy^2 + 1$$

# Gradient

# Gradient

- The gradient of a function  $f(x_1, x_2, \dots, x_n)$  is a vector function, returning all of the partial derivatives

# Gradient

- The gradient of a function  $f(x_1, x_2, \dots, x_n)$  is a vector function, returning all of the partial derivatives

$$\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right\rangle$$

# Gradient

- The gradient of a function  $f(x_1, x_2, \dots, x_n)$  is a vector function, returning all of the partial derivatives

$$\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right\rangle$$

$$f(x) = 4x^2 + y^2$$

$$\nabla f = \langle 8x, 2y \rangle$$

# Gradient

- The gradient of a function  $f(x_1, x_2, \dots, x_n)$  is a vector function, returning all of the partial derivatives

$$\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right\rangle$$

$$f(x) = 4x^2 + y^2$$

$$\nabla f = \langle 8x, 2y \rangle$$

- The gradient is perpendicular to the *level curve* at a point

# Gradient

- The gradient of a function  $f(x_1, x_2, \dots, x_n)$  is a vector function, returning all of the partial derivatives

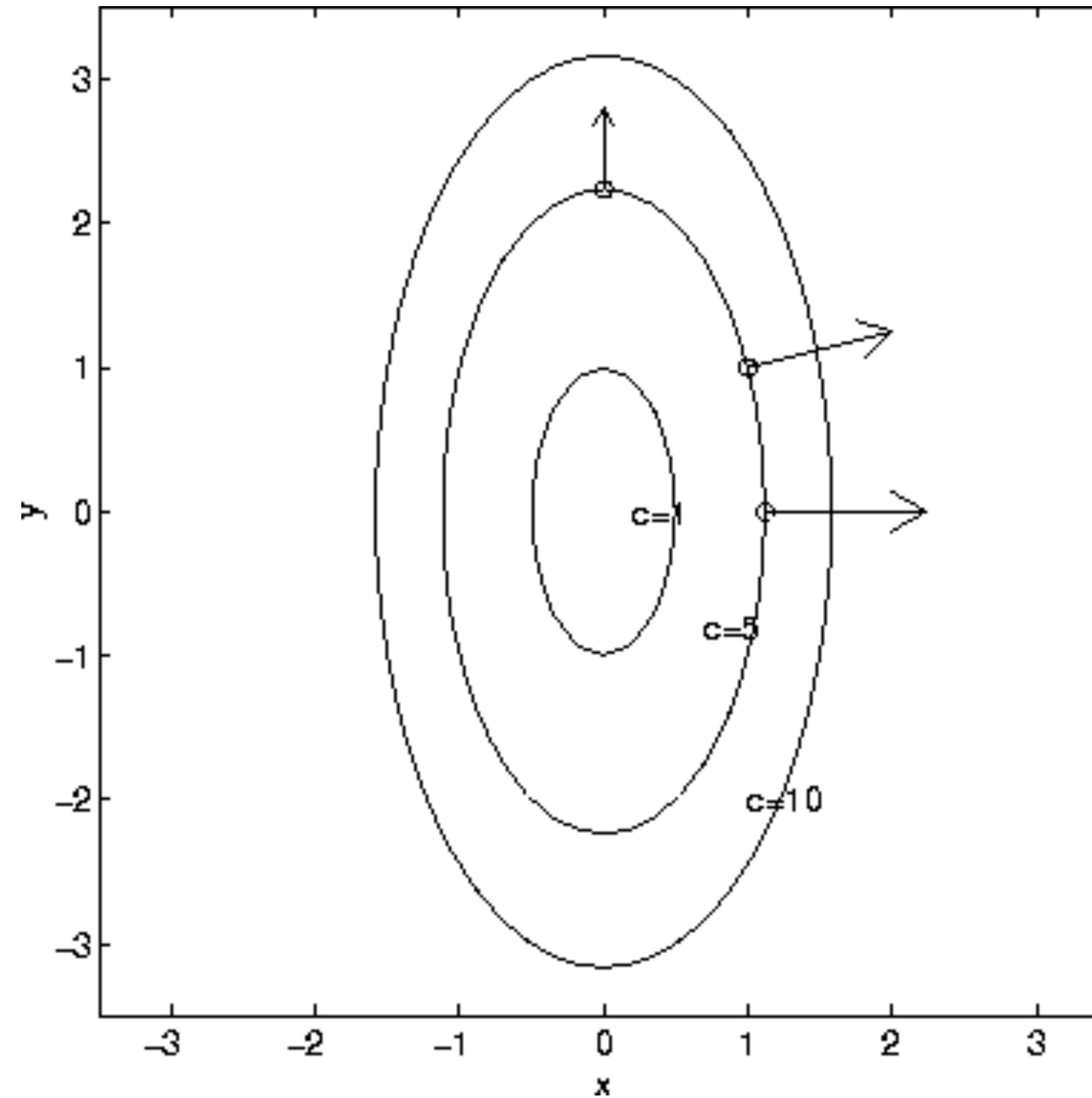
$$\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right\rangle$$

$$f(x) = 4x^2 + y^2$$

$$\nabla f = \langle 8x, 2y \rangle$$

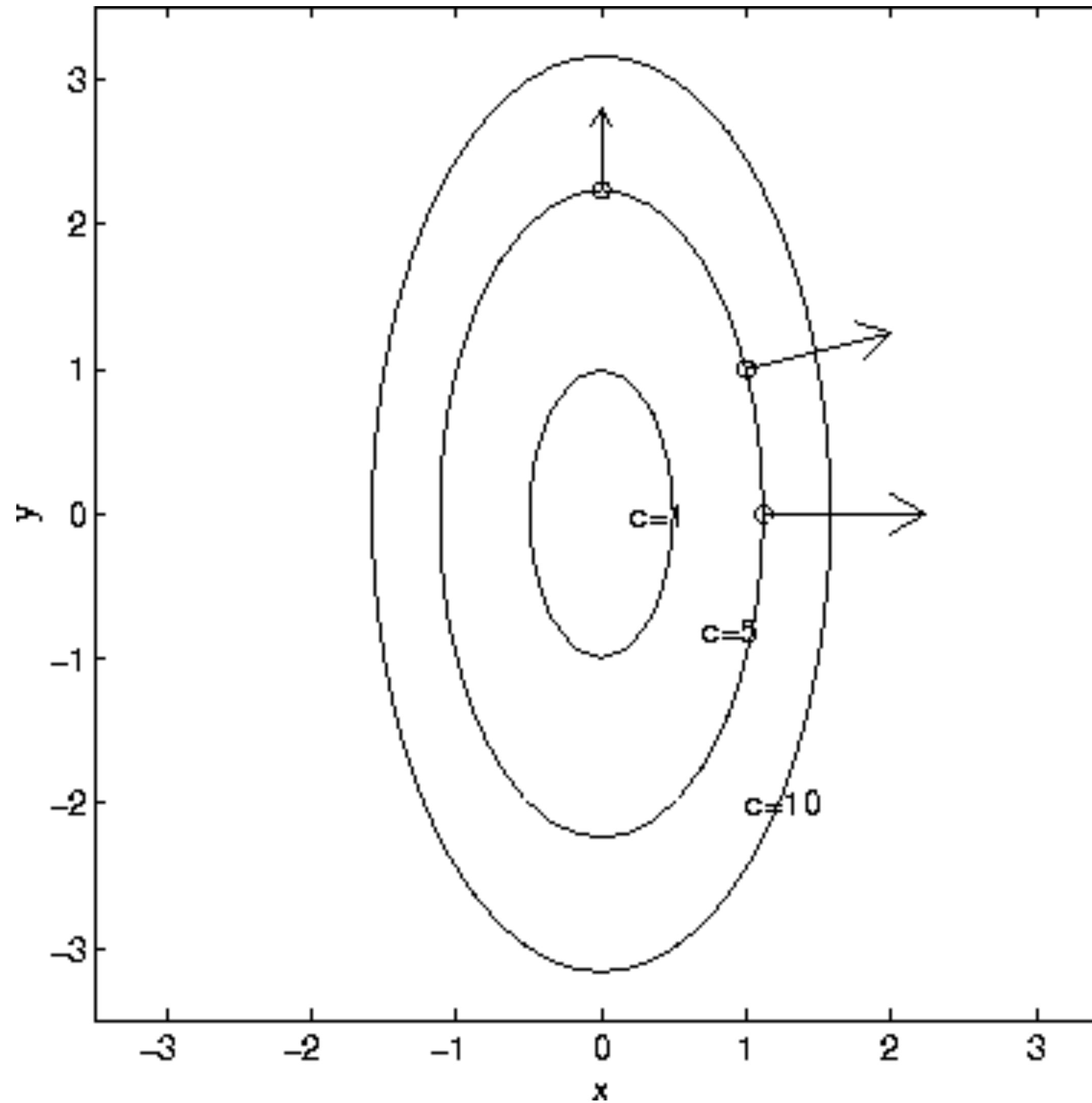
- The gradient is perpendicular to the *level curve* at a point
- The gradient points in the direction of greatest rate of increase of  $f$

# Gradient and Level Curves



Level curves:  $f(x) = c$

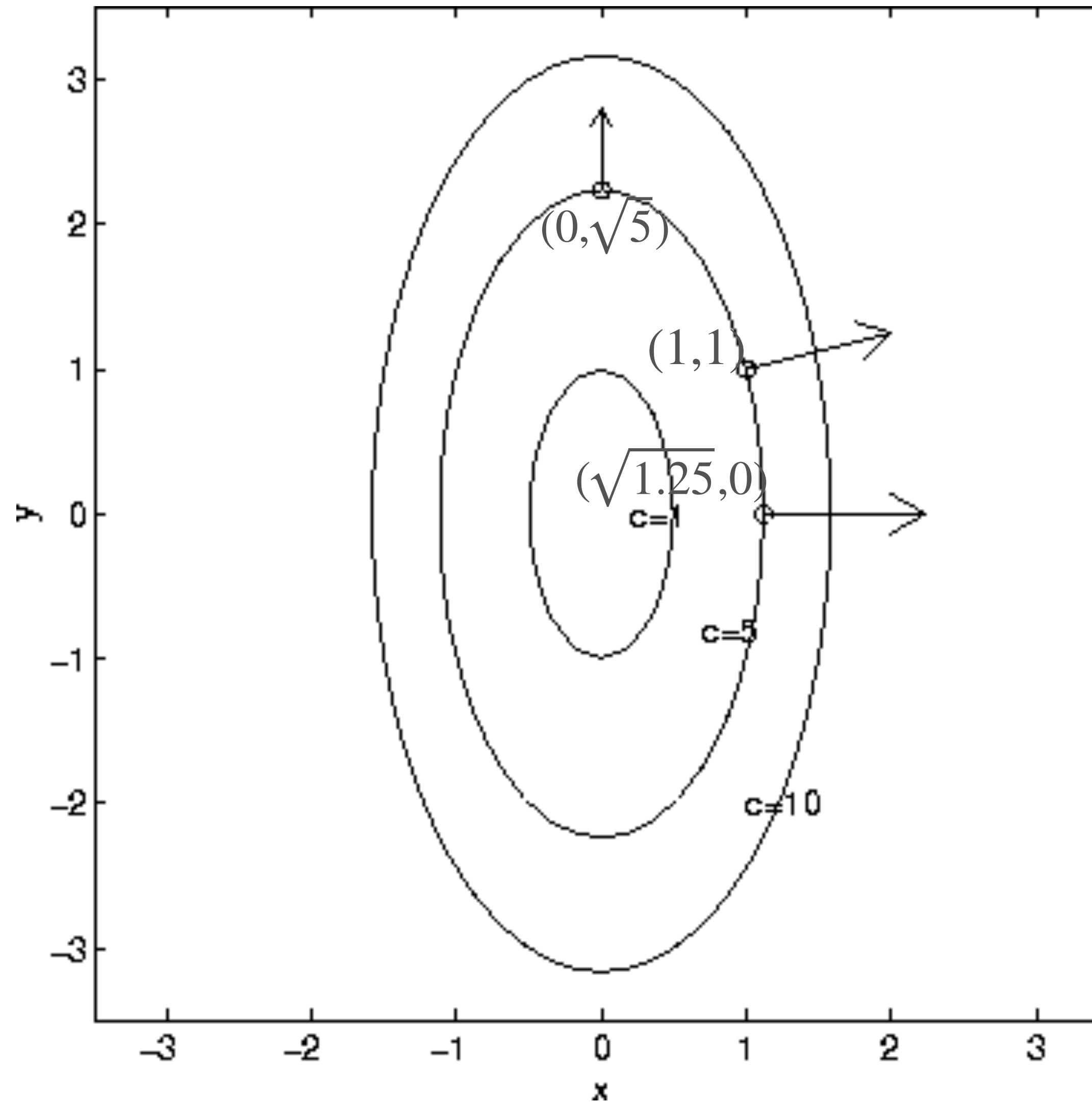
# Gradient and Level Curves



$$f(x) = 4x^2 + y^2$$
$$\nabla f = \langle 8x, 2y \rangle$$

Level curves:  $f(x) = c$

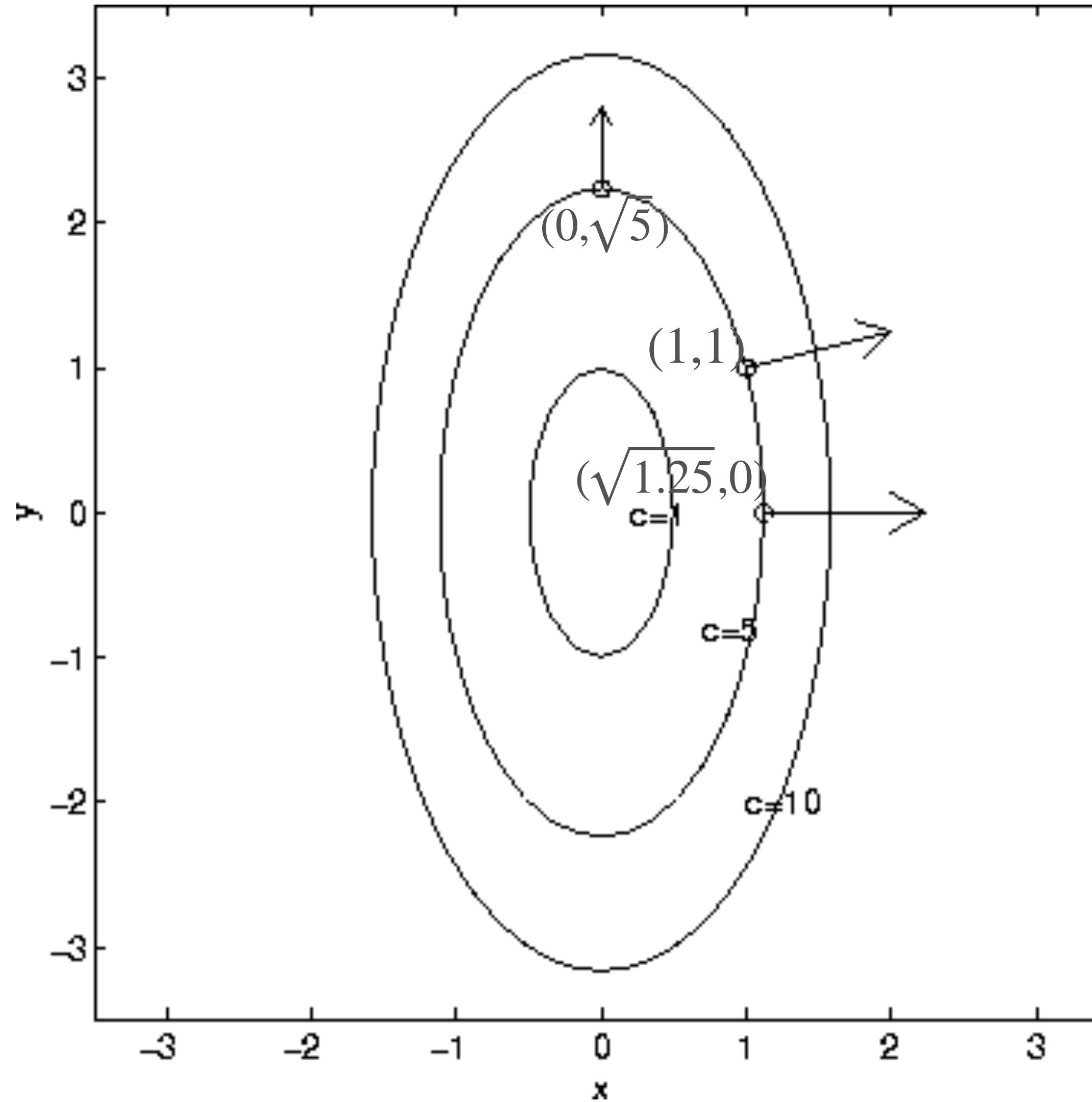
# Gradient and Level Curves



$$f(x) = 4x^2 + y^2$$
$$\nabla f = \langle 8x, 2y \rangle$$

Level curves:  $f(x) = c$

# Gradient and Level Curves

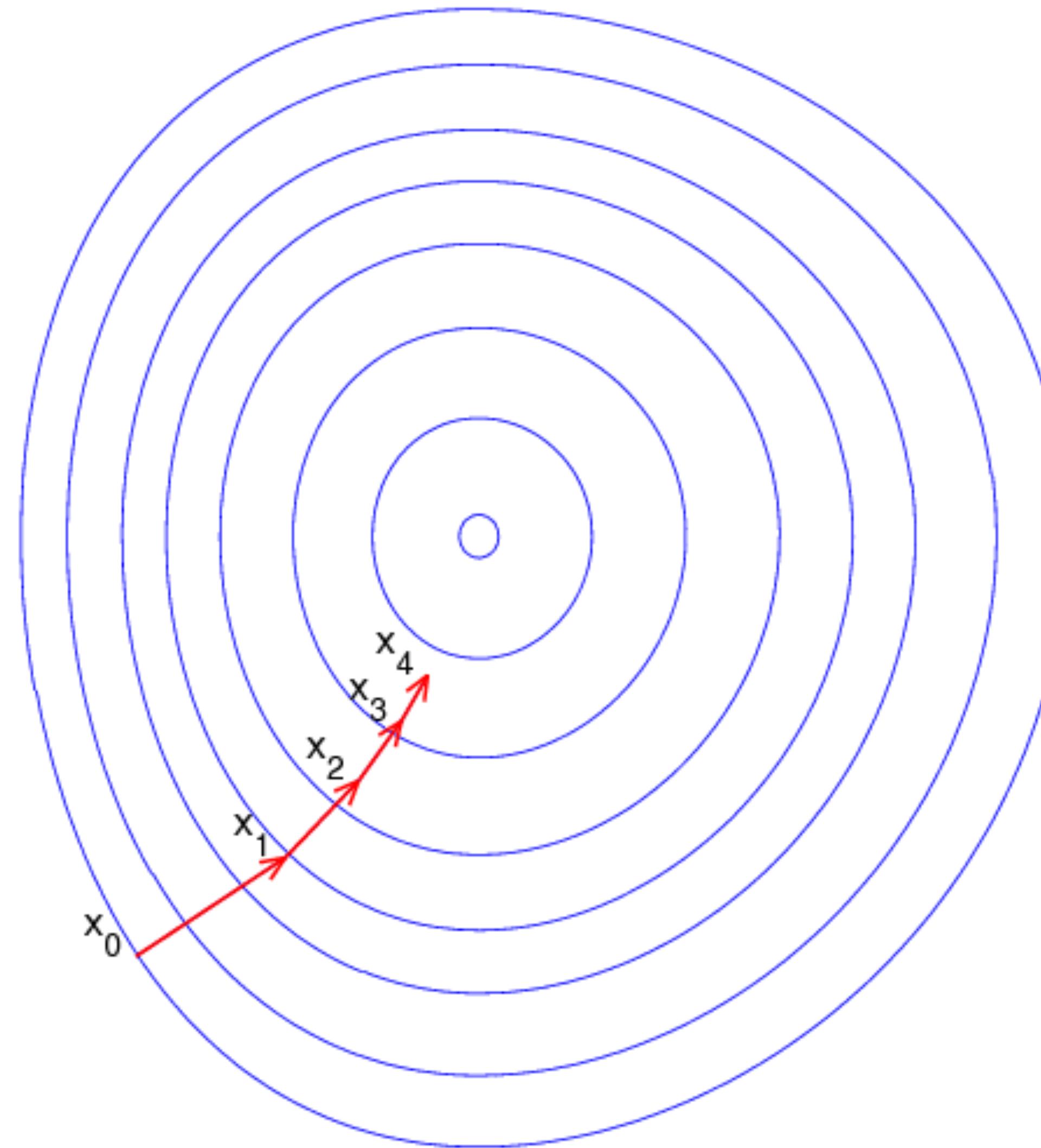


$$f(x) = 4x^2 + y^2$$
$$\nabla f = \langle 8x, 2y \rangle$$

Level curves:  $f(x) = c$

Q: what are the actual gradients  
at those points?

# Gradient Descent and Level Curves



# Gradient Descent Algorithm

- Initialize  $\theta_0$
- Repeat until convergence:

$$\theta_{n+1} = \theta_n - \alpha \nabla \mathcal{L}(\hat{Y}(\theta_n), Y)$$

# Gradient Descent Algorithm

- Initialize  $\theta_0$
- Repeat until convergence:

$$\theta_{n+1} = \theta_n - \alpha \nabla \mathcal{L}(\hat{Y}(\theta_n), Y)$$



Learning rate

# Gradient Descent Algorithm

- Initialize  $\theta_0$
- Repeat until convergence:

$$\theta_{n+1} = \theta_n - \alpha \nabla \mathcal{L}(\hat{Y}(\theta_n), Y)$$

Learning rate

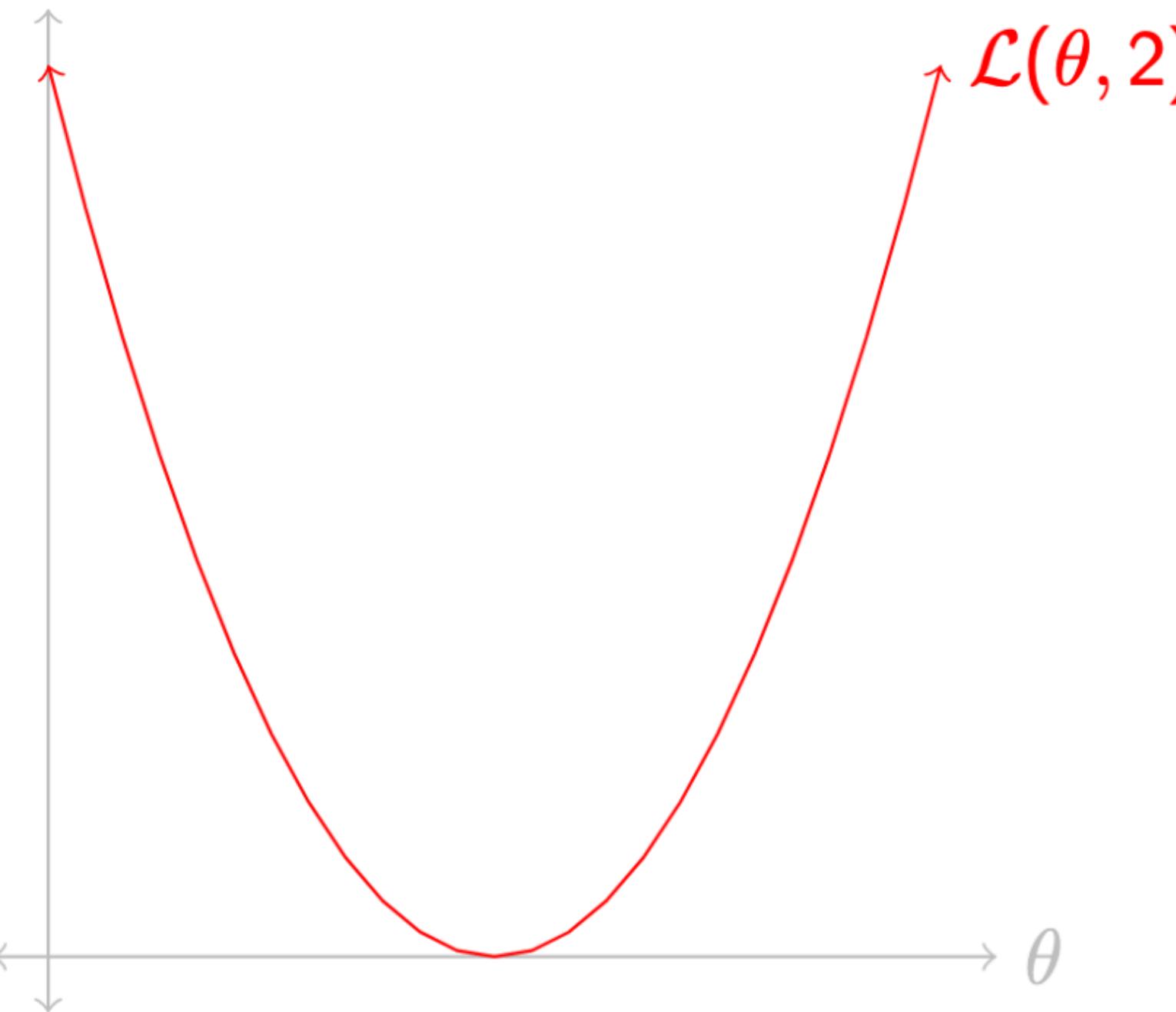
- High learning rate: big steps, may bounce and “overshoot” the target
- Low learning rate: small steps, smoother minimization of loss, but can be slow

# Gradient Descent: Minimal Example

- Task: predict a target/true value  $y = 2$
- “Model”:  $\hat{y}(\theta) = \theta$ 
  - A single parameter: the actual guess
- Loss: Euclidean distance

$$\mathcal{L}(\hat{y}(\theta), y) = (\hat{y} - y)^2 = (\theta - y)^2$$

# Gradient Descent: Minimal Example



$$\frac{\partial}{\partial \theta} \mathcal{L}(\theta, y) = 2(\theta - y)$$

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\partial}{\partial \theta} \mathcal{L}(\theta, y)$$

# Stochastic Gradient Descent

# Stochastic Gradient Descent

- The above is called “batch” gradient descent
  - Updates *once per pass through the dataset*
  - Expensive, and slow; does not scale well

# Stochastic Gradient Descent

- The above is called “batch” gradient descent
  - Updates *once per pass through the dataset*
  - Expensive, and slow; does not scale well
- *Stochastic* gradient descent:
  - Break the data into “mini-batches”: small chunks of the data
  - Compute gradients and update parameters for each batch
  - Mini-batch of size 1 = single example
  - A *noisy estimate* of the true gradient, but works well in practice; more parameter updates

# Stochastic Gradient Descent

- The above is called “batch” gradient descent
  - Updates *once per pass through the dataset*
  - Expensive, and slow; does not scale well
- *Stochastic* gradient descent:
  - Break the data into “mini-batches”: small chunks of the data
  - Compute gradients and update parameters for each batch
  - Mini-batch of size 1 = single example
  - A *noisy estimate* of the true gradient, but works well in practice; more parameter updates
- Epoch: one pass through the whole training data

# Stochastic Gradient Descent

initialize parameters / build model

for each epoch:

```
data = shuffle(data)
batches = make_batches(data)
```

for each batch in batches:

```
outputs = model(batch)
loss = loss_fn(outputs, true_outputs)
compute gradients
update parameters
```

# 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. (From *Lin, 1998a*)

# 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 1 0 0 0 ... 0]
    - 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?

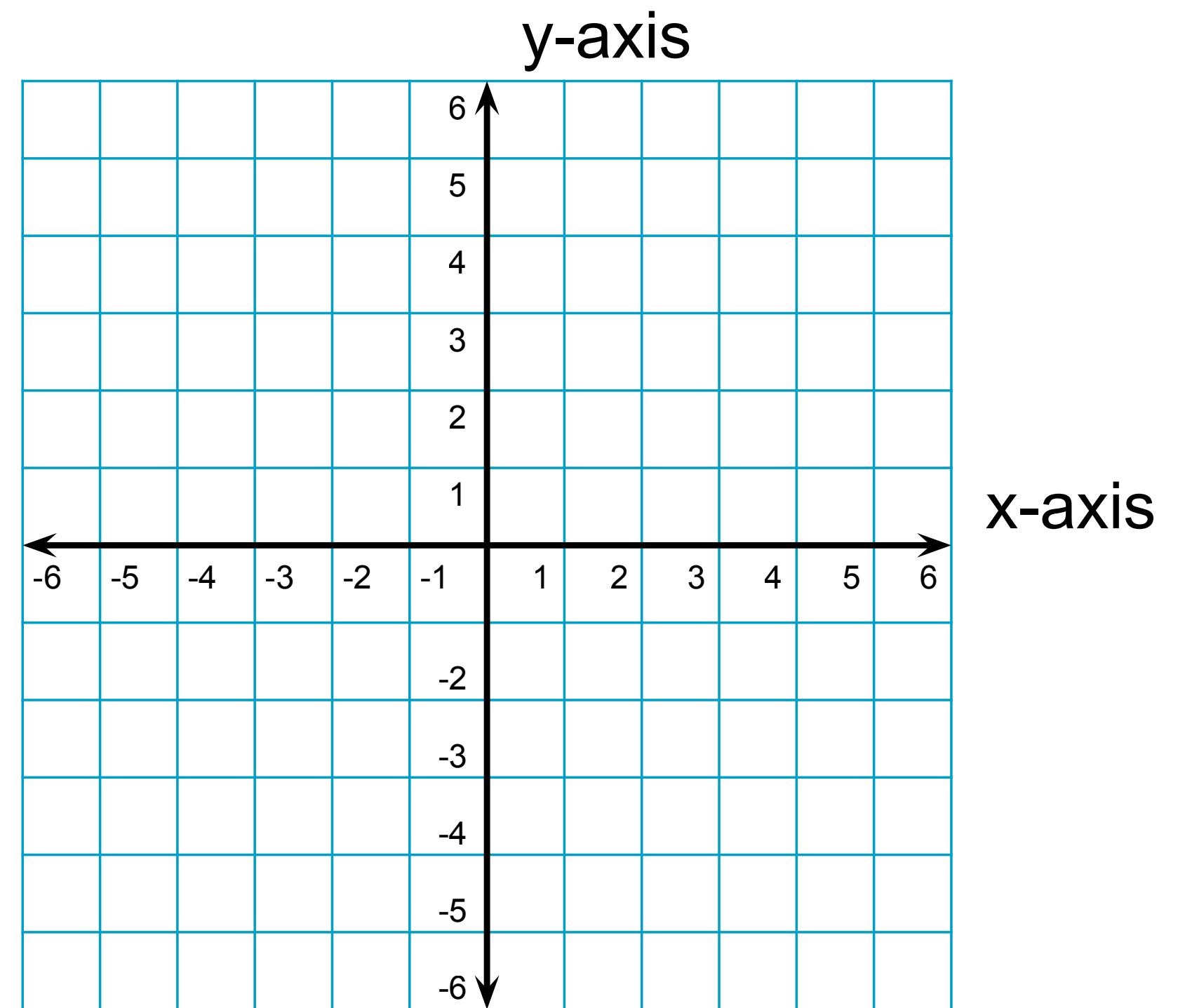
- With embeddings (= vectors):
  - Feature is a word vector
  - 'The previous word was vector [35,22,17, ...]
  - Now in the test set we might see a similar vector [34,21,14, ...]
  - We can generalize to similar but unseen words!

# Vectors: A Refresher

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

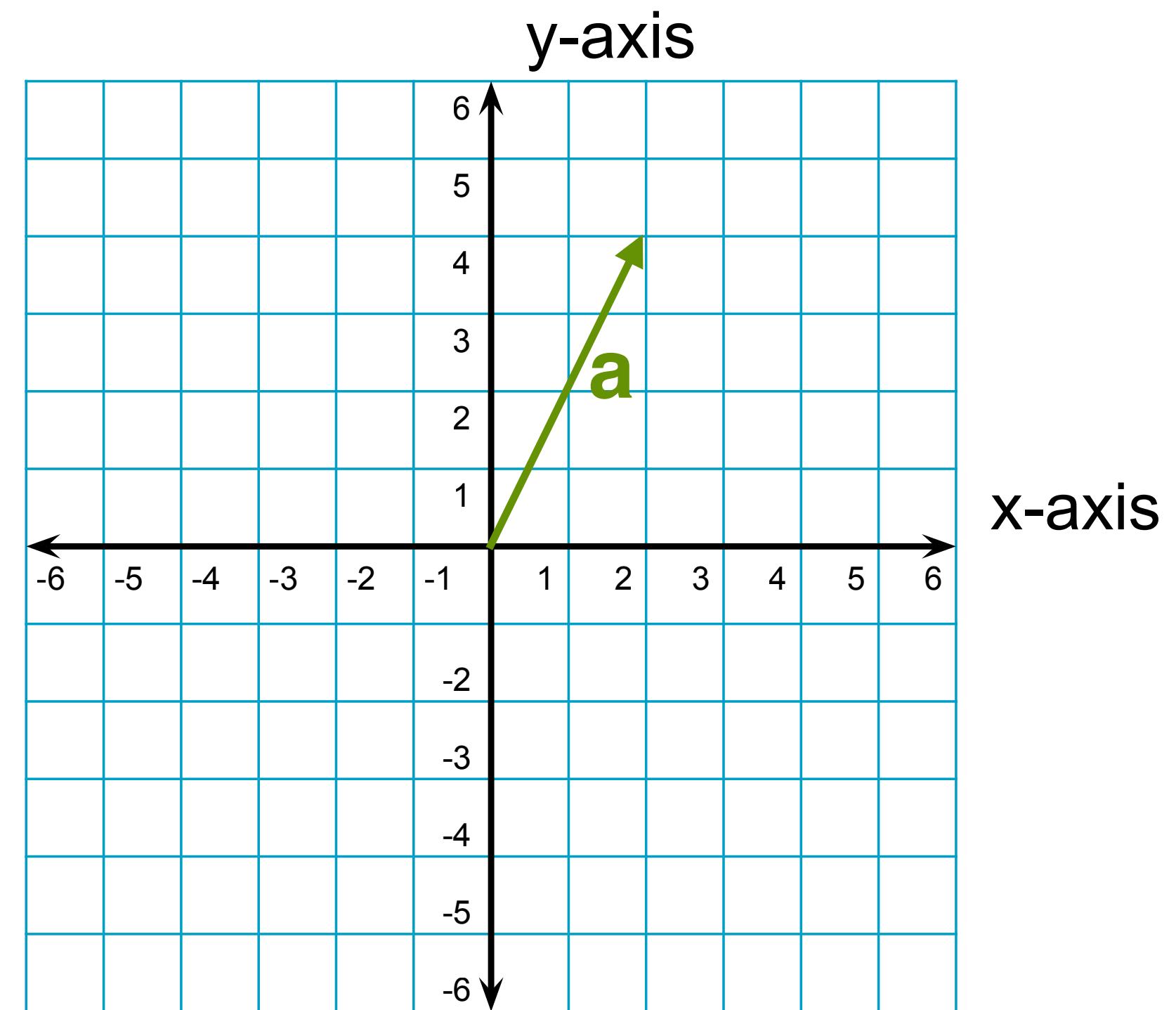
# Vectors: A Refresher

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



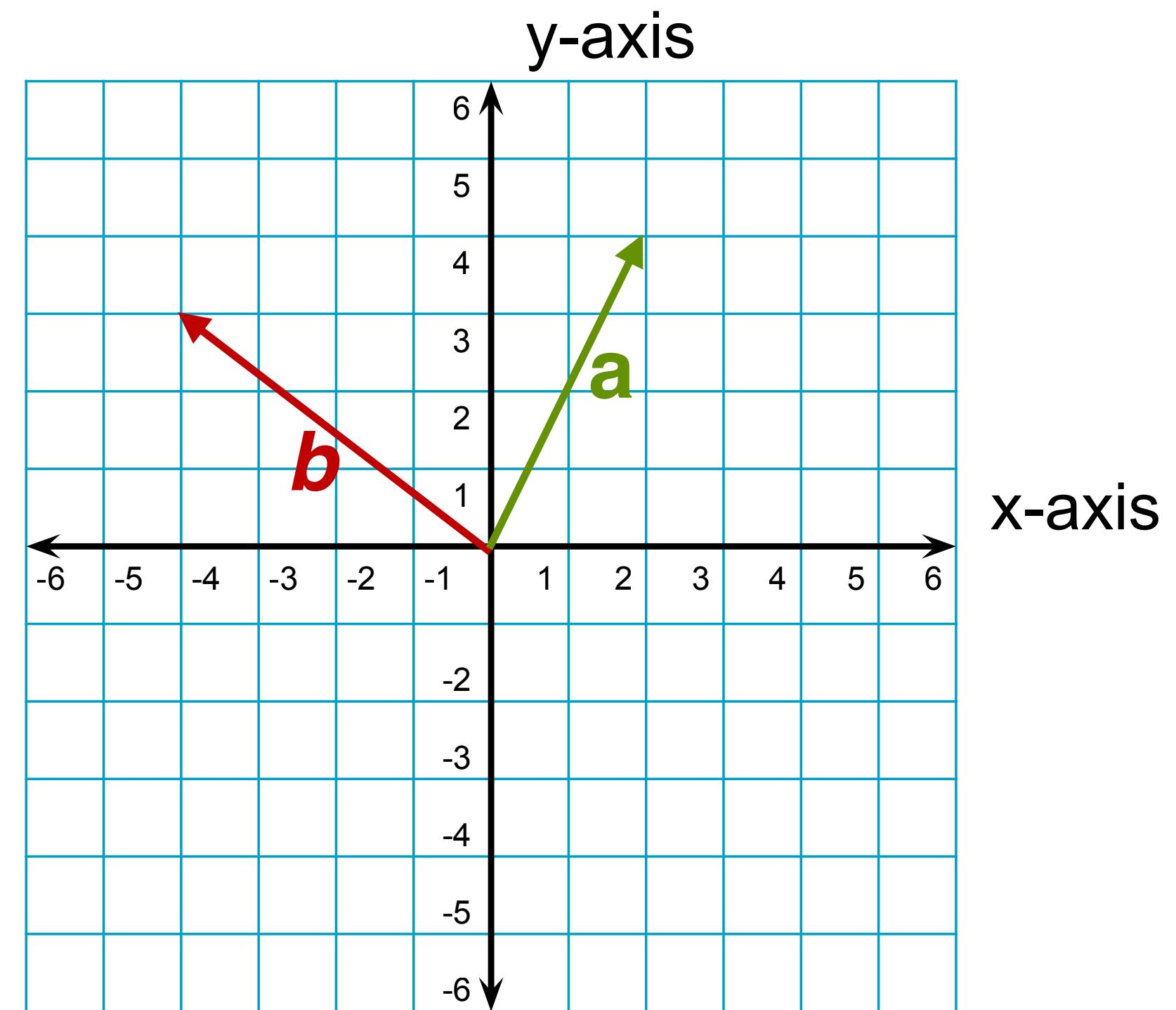
# Vectors: A Refresher

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



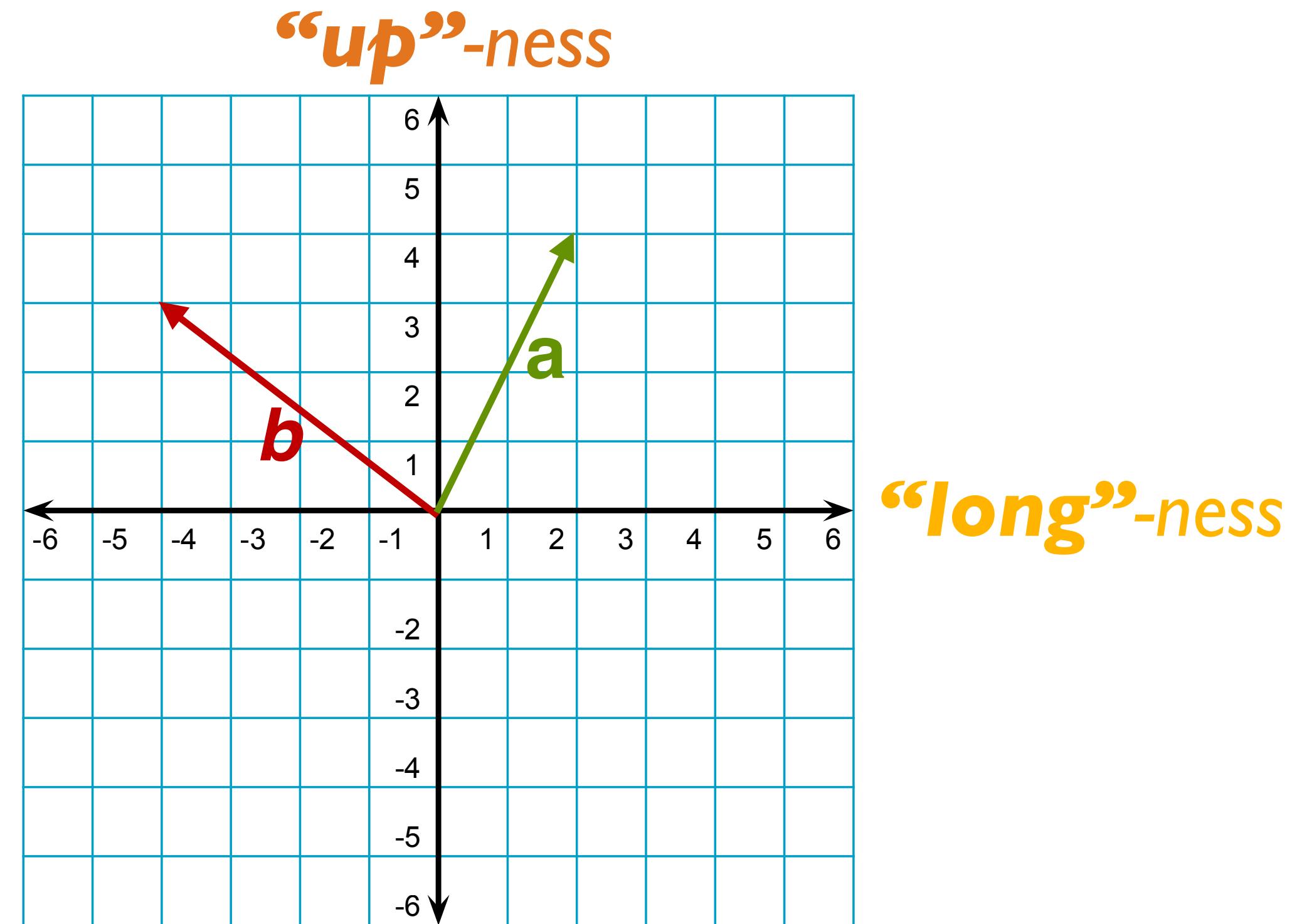
# Vectors: A Refresher

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



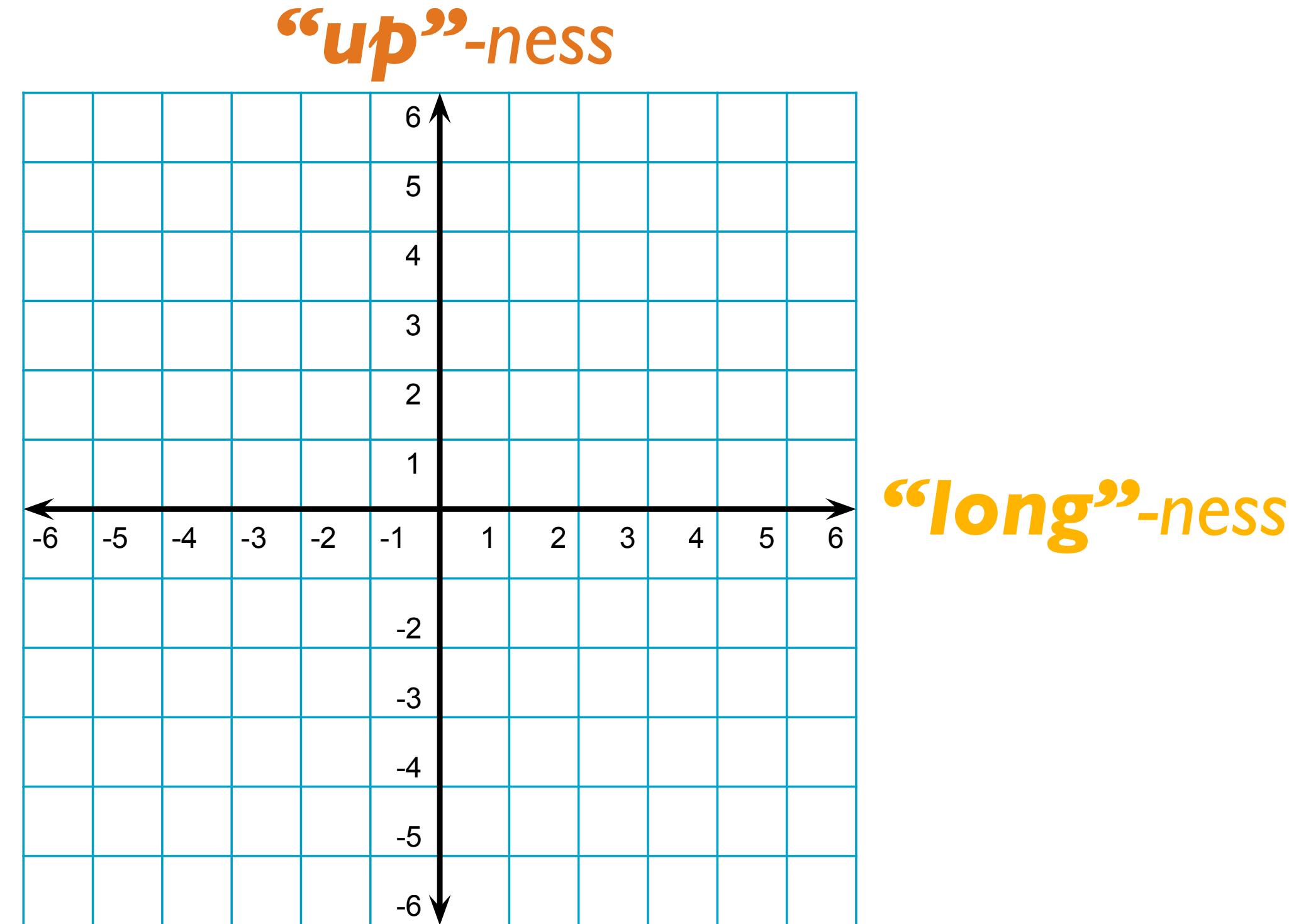
# Vectors: A Refresher

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



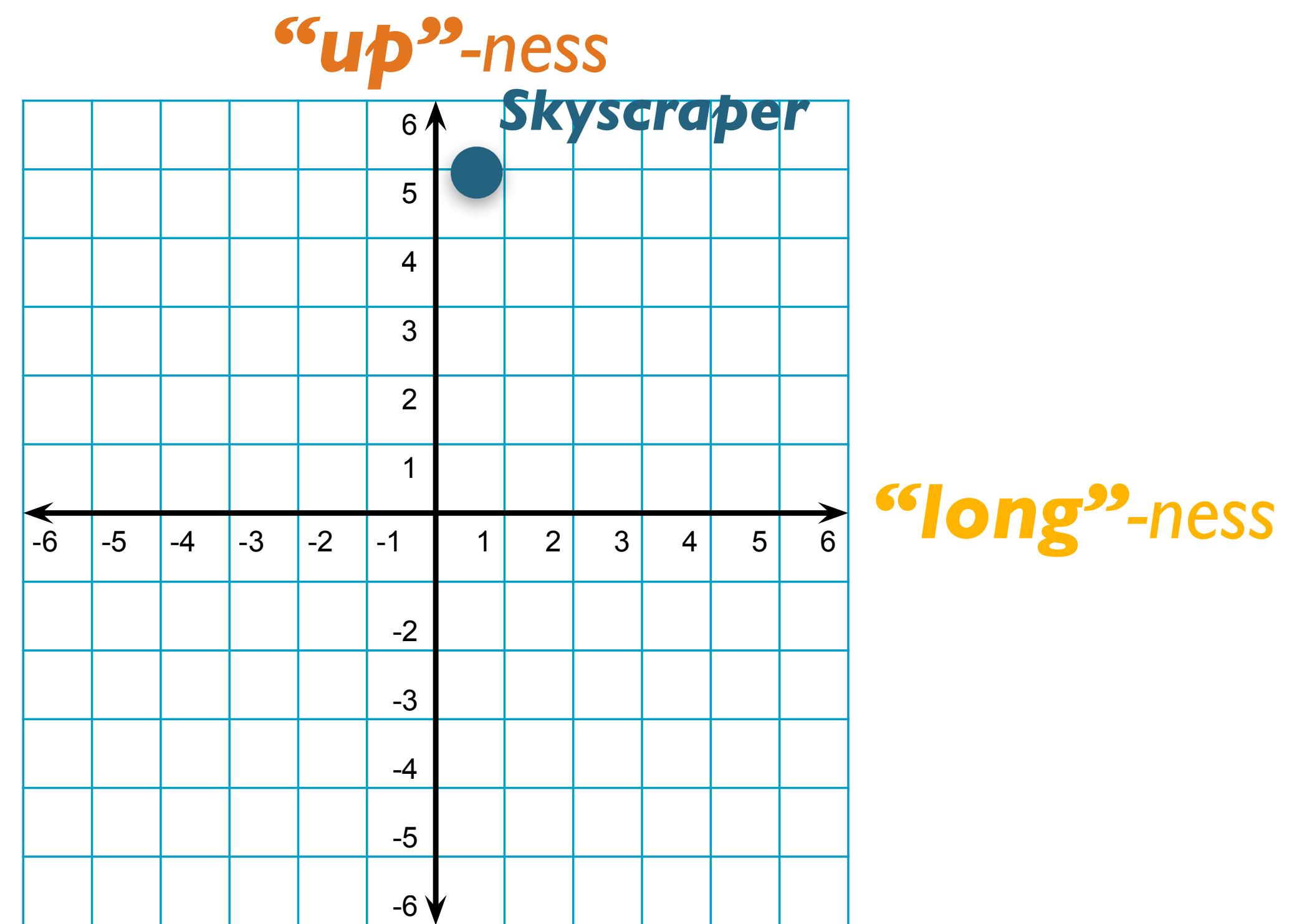
# Vectors: A Refresher

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



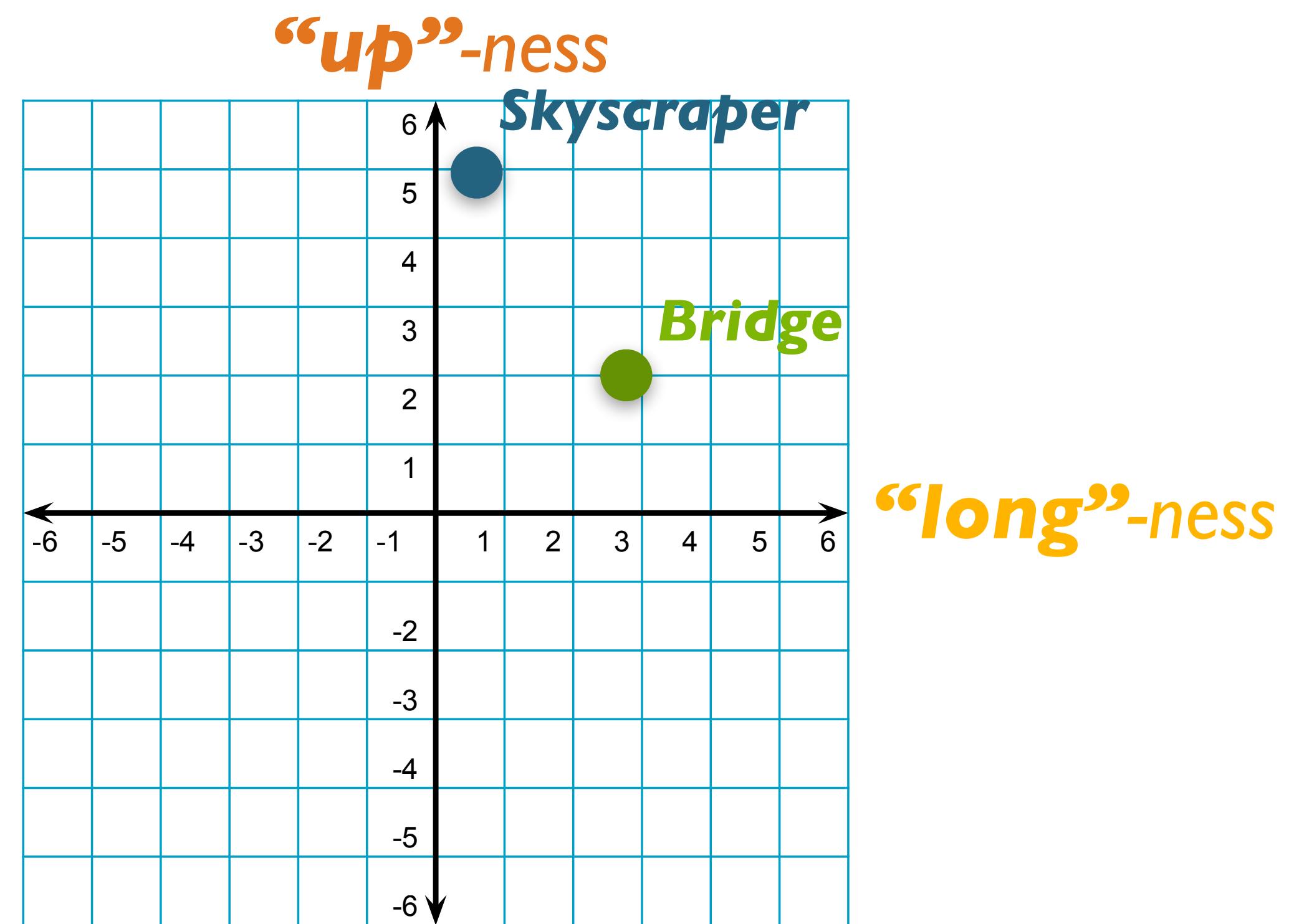
# Vectors: A Refresher

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



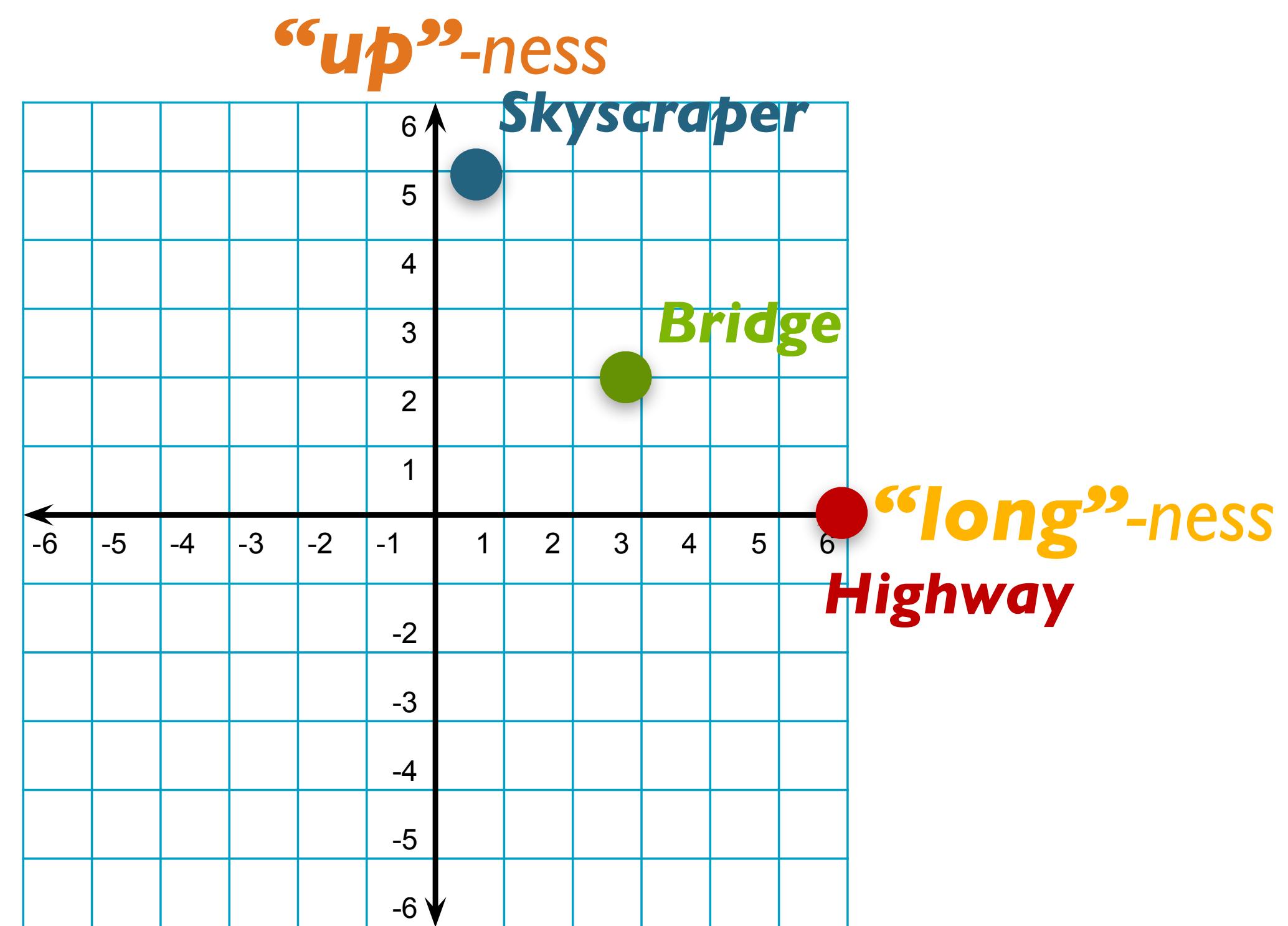
# Vectors: A Refresher

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



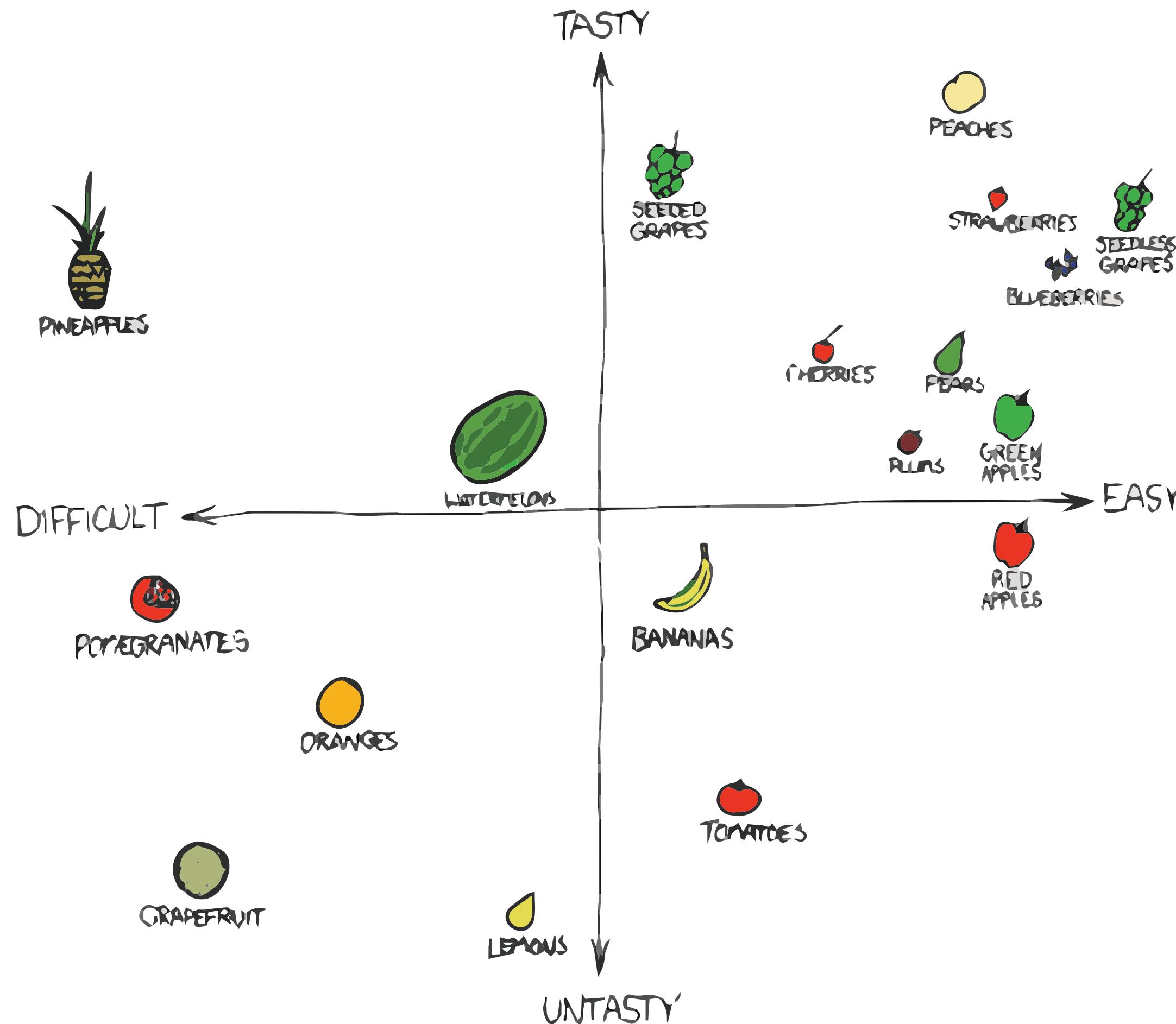
# Vectors: A Refresher

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



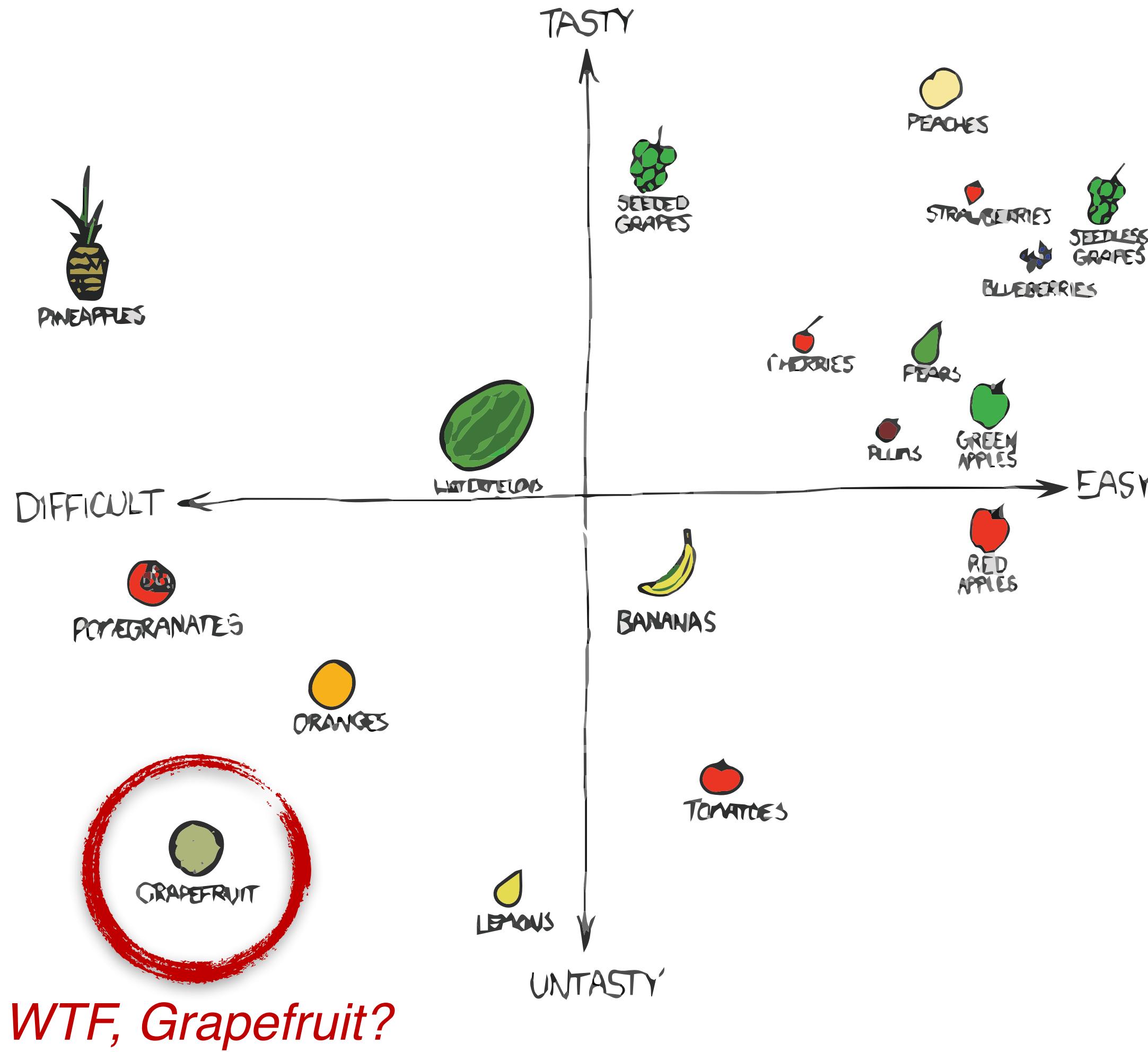
# Vectors: A Refresher

[xkcd.com/388](http://xkcd.com/388)



# Vectors: A Refresher

[xkcd.com/388](http://xkcd.com/388)



# Basic vector operations

- Addition:  $\mathbf{x} + \mathbf{y} = \langle \mathbf{x}_0 + \mathbf{y}_0, \dots, \mathbf{x}_n + \mathbf{y}_n \rangle$
- Subtraction:  $\mathbf{x} - \mathbf{y} = \langle \mathbf{x}_0 - \mathbf{y}_0, \dots, \mathbf{x}_n - \mathbf{y}_n \rangle$
- Scalar multiplication:  $k\mathbf{x} = \langle k\mathbf{x}_0, \dots, k\mathbf{x}_n \rangle$
- Length:  $\|\mathbf{x}\| = \sqrt{\sum_i \mathbf{x}_i^2}$

# Vector Distances: Manhattan & Euclidean

- **Manhattan Distance**

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

- (Distance as cumulative horizontal + vertical moves)

- **Euclidean Distance**

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

- Too sensitive to extreme values

# Vector Distances: Manhattan & Euclidean

- **Manhattan Distance**

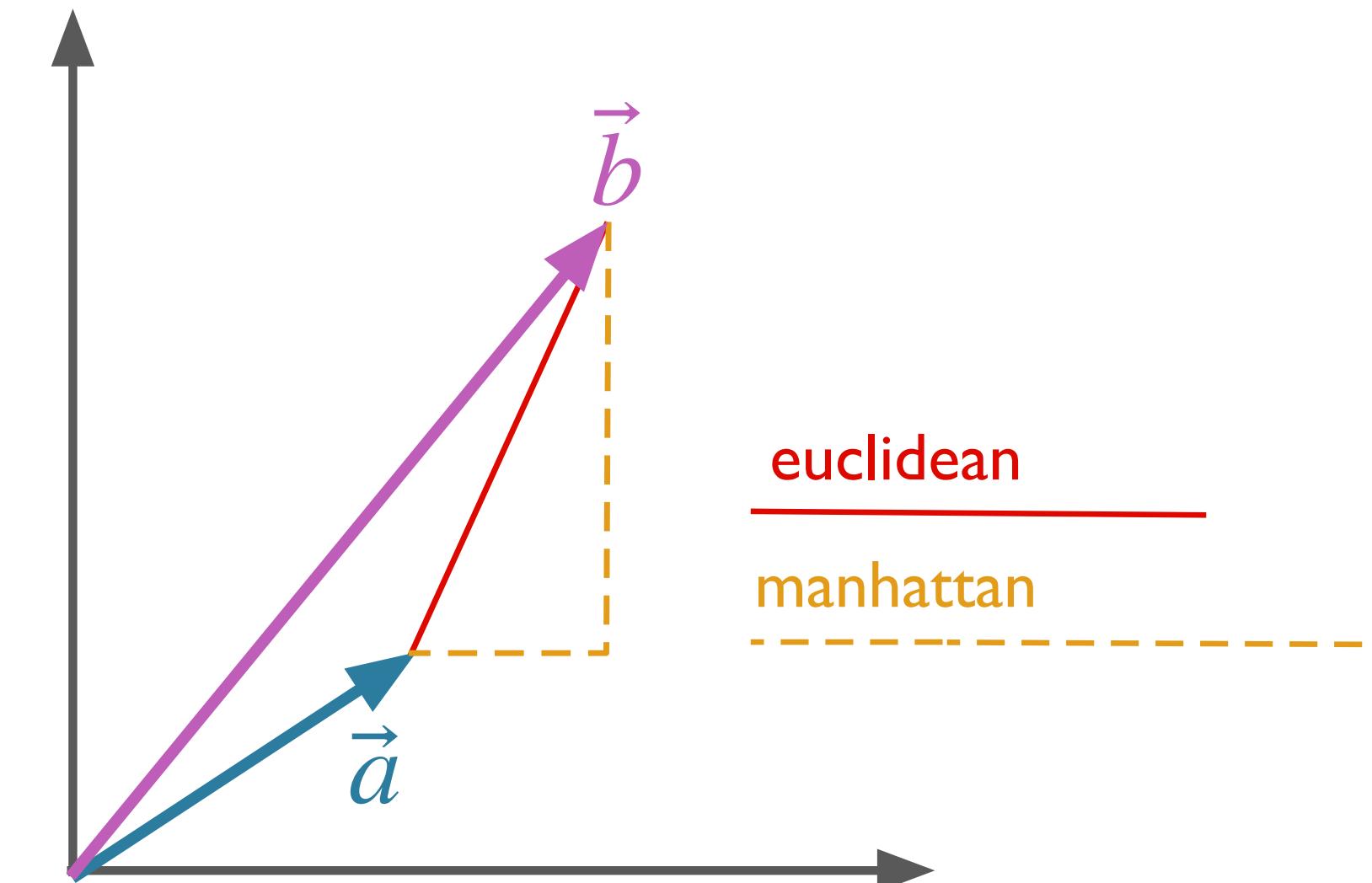
$$d_{\text{manhattan}}(x, y) = \sum |x_i - y_i|$$

- (Distance as cumulative horizontal + vertical moves)

- **Euclidean Distance**

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

- Too sensitive to extreme values



# Vector Similarity: Dot Product

- Produces real number scalar from product of vectors' components
- Biased toward *longer* (larger magnitude) vectors
  - In our case, vectors with fewer zero counts

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

# Vector Similarity: Cosine

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

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

# Bag of Words Vectors

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

# Bag of Words Vectors

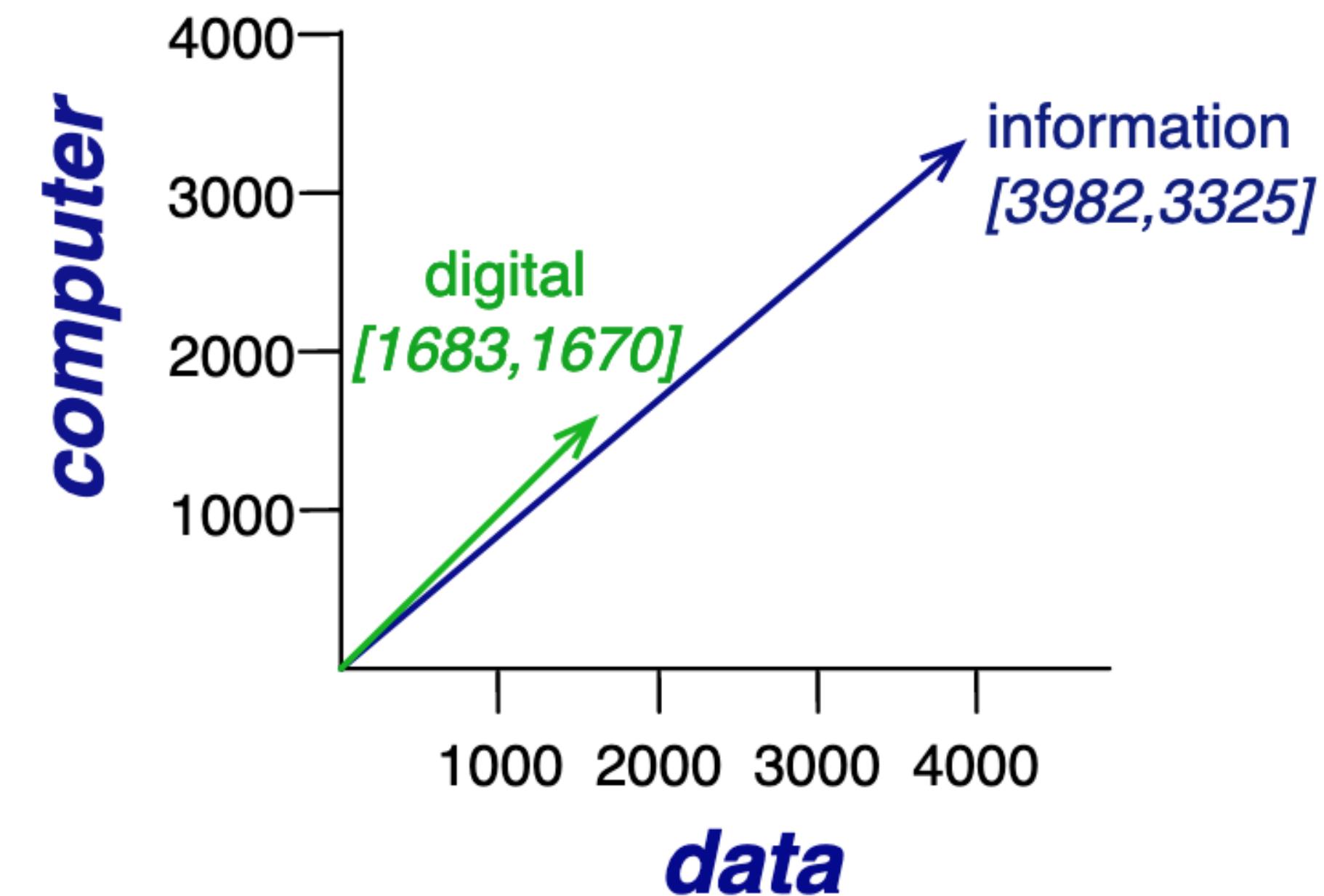
- Represent ‘company’ of word such that similar words will have similar representations
  - ‘Company’ = context
- Word represented by context feature vector
  - Many alternatives for vector

# Bag of Words Vectors

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

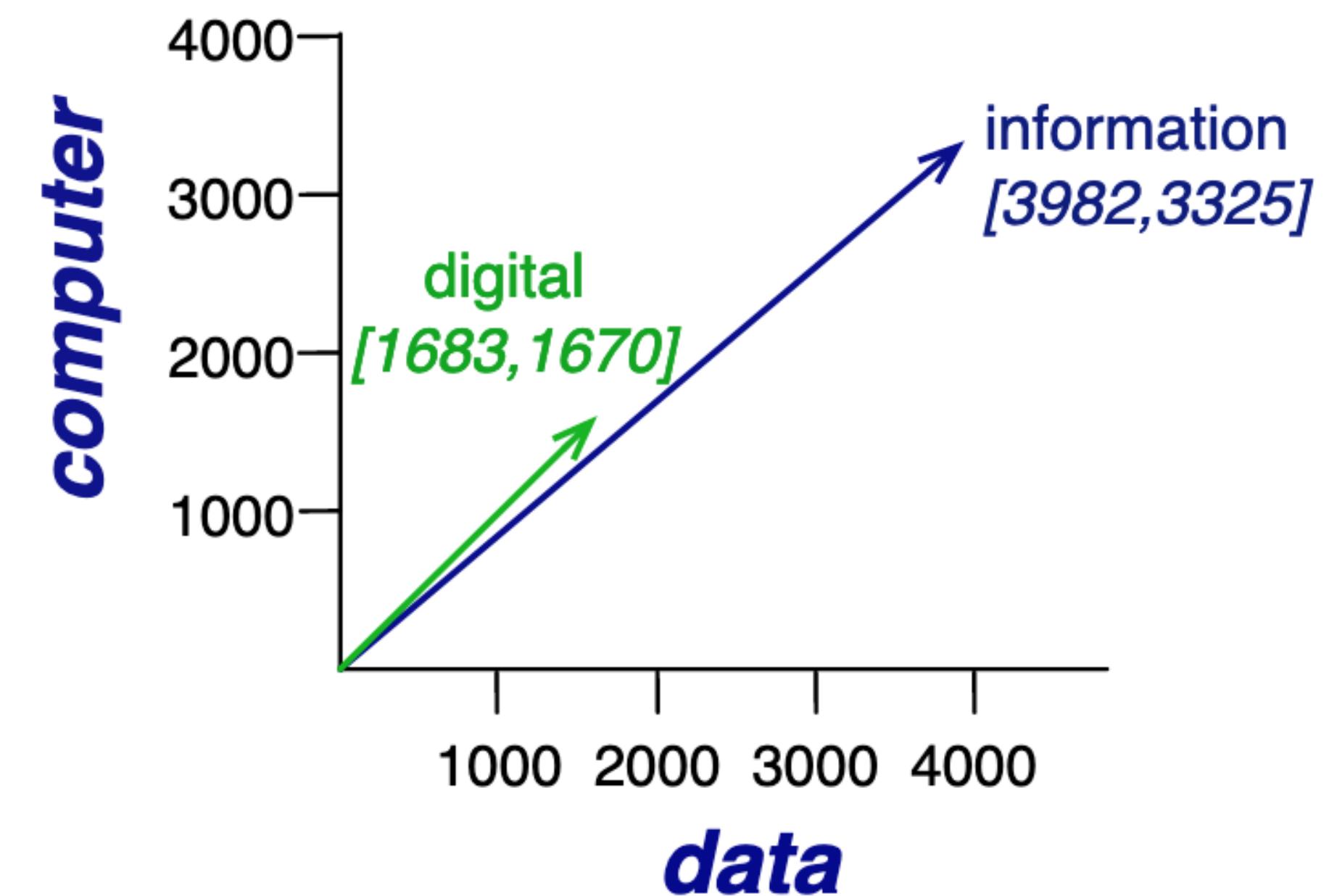
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



# Bag of Words Vectors

- Usually re-weighted, with 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 learn 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

- Continuous Bag of Words (CBOW):

- $P(\text{word} \mid \text{context})$
- Input:  $(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \dots)$
- Output:  $p(w_t)$

# Embeddings: Skip-Gram vs. Continuous Bag of Words

- Continuous Bag of Words (CBOW):

- $P(\text{word} \mid \text{context})$
- Input:  $(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \dots)$
- Output:  $p(w_t)$

- Skip-gram:

- $P(\text{context} \mid \text{word})$
- Input:  $w_t$
- Output:  $p(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \dots)$

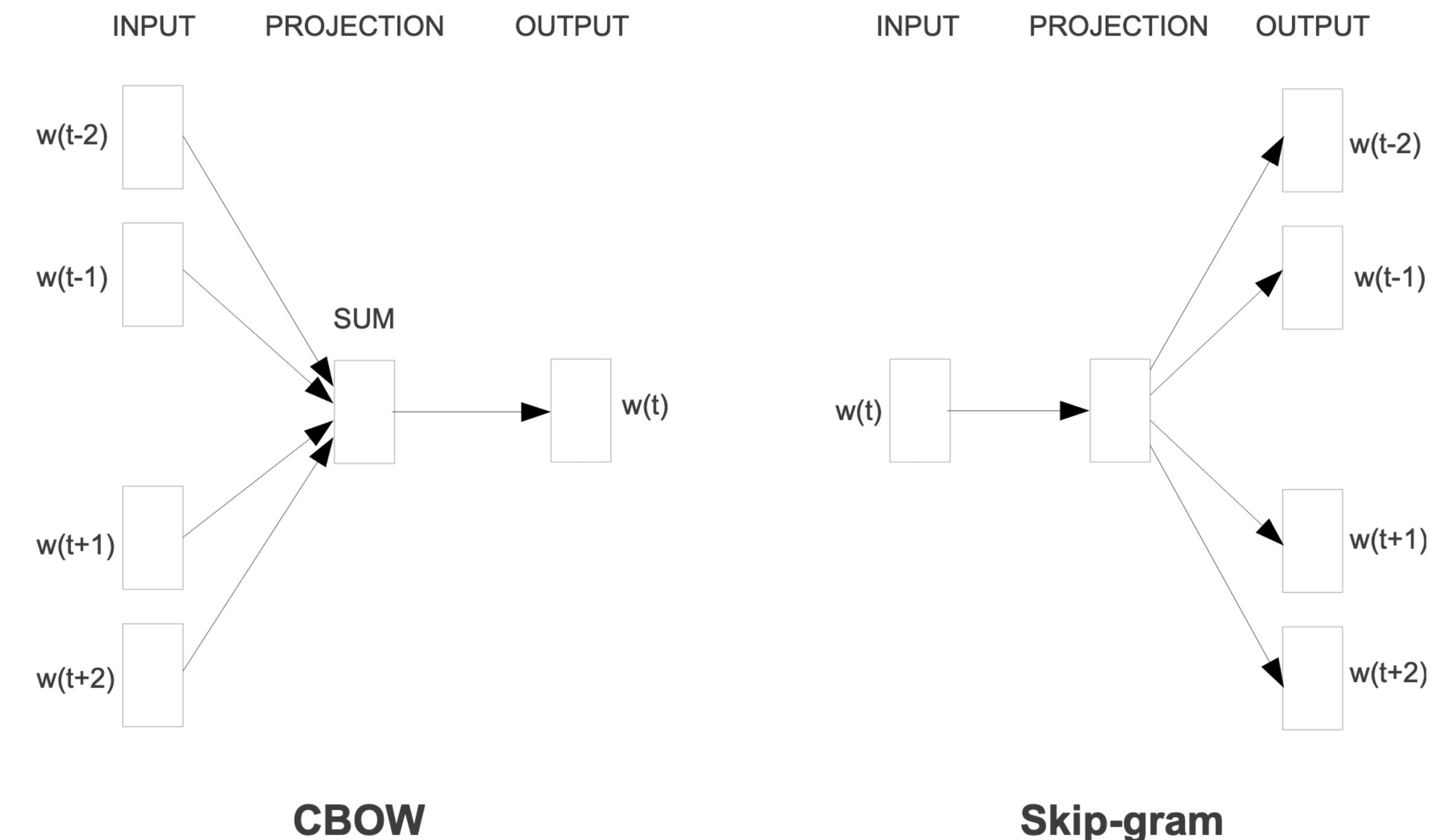
# Embeddings: Skip-Gram vs. Continuous Bag of Words

- Continuous Bag of Words (CBOW):

- $P(\text{word} \mid \text{context})$
- Input:  $(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \dots)$

- Skip-gram:**

- $P(\text{context} \mid \text{word})$
- Input:  $w_t$
- Output:  $p(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \dots)$



Mikolov et al 2013a (the OG word2vec paper)

# Skip-Gram Model

- Learns two embeddings
  - $W$ : word, matrix of shape [vocab\_size, embedding\_dimension]
  - $C$ : context embedding, matrix of same shape

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

# Skip-Gram Model

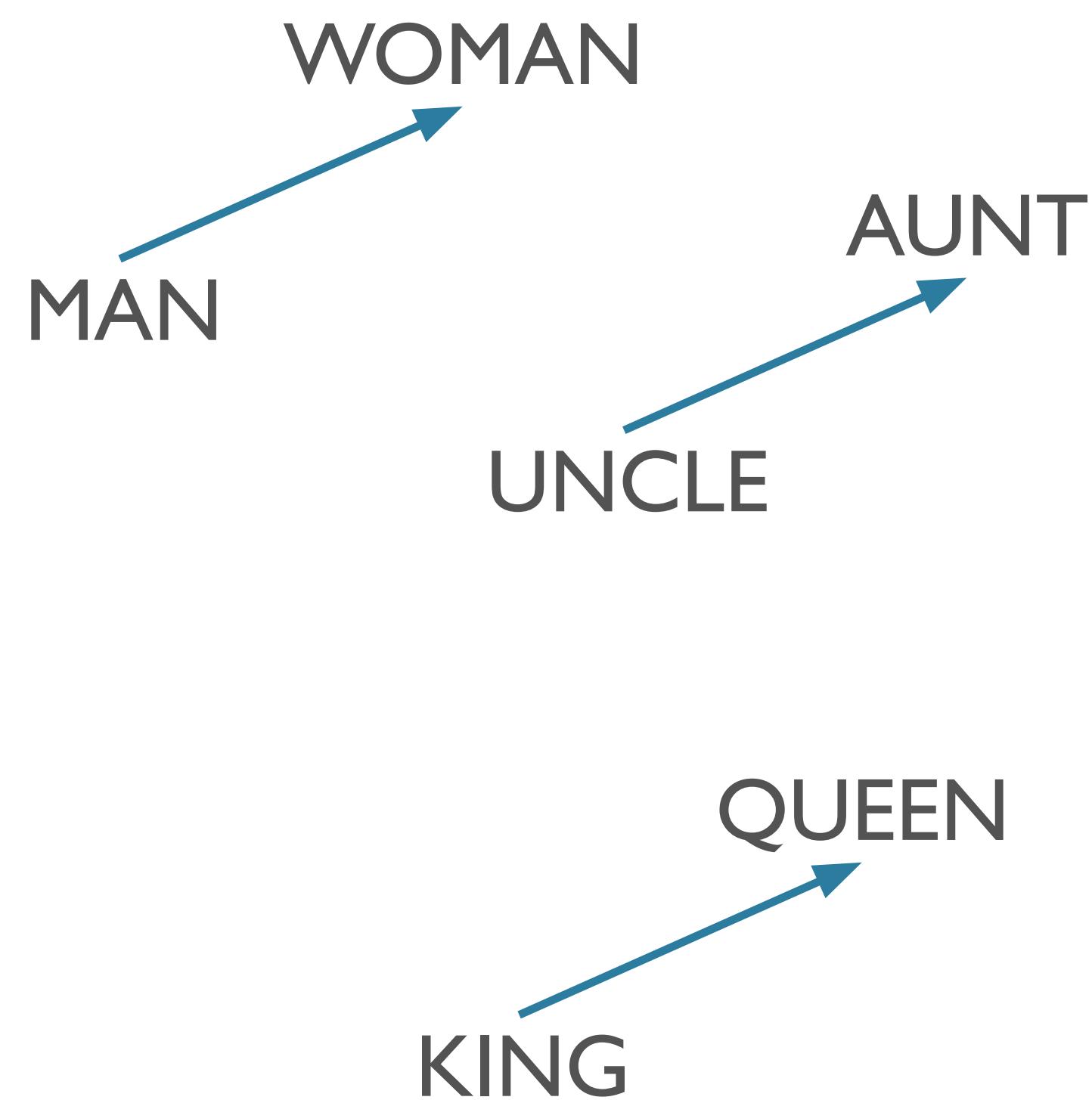
- Learns two embeddings
  - $W$ : word, matrix of shape [vocab\_size, embedding\_dimension]
  - $C$ : context embedding, matrix of same shape
- Prediction task:
  - Given a word, predict each neighbor word in window
  - Compute  $p(w_k|w_j)$  as proportional to  $c_k \cdot w_j$ 
    - For each context position
  - Convert to probability via softmax

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

# Training The Model

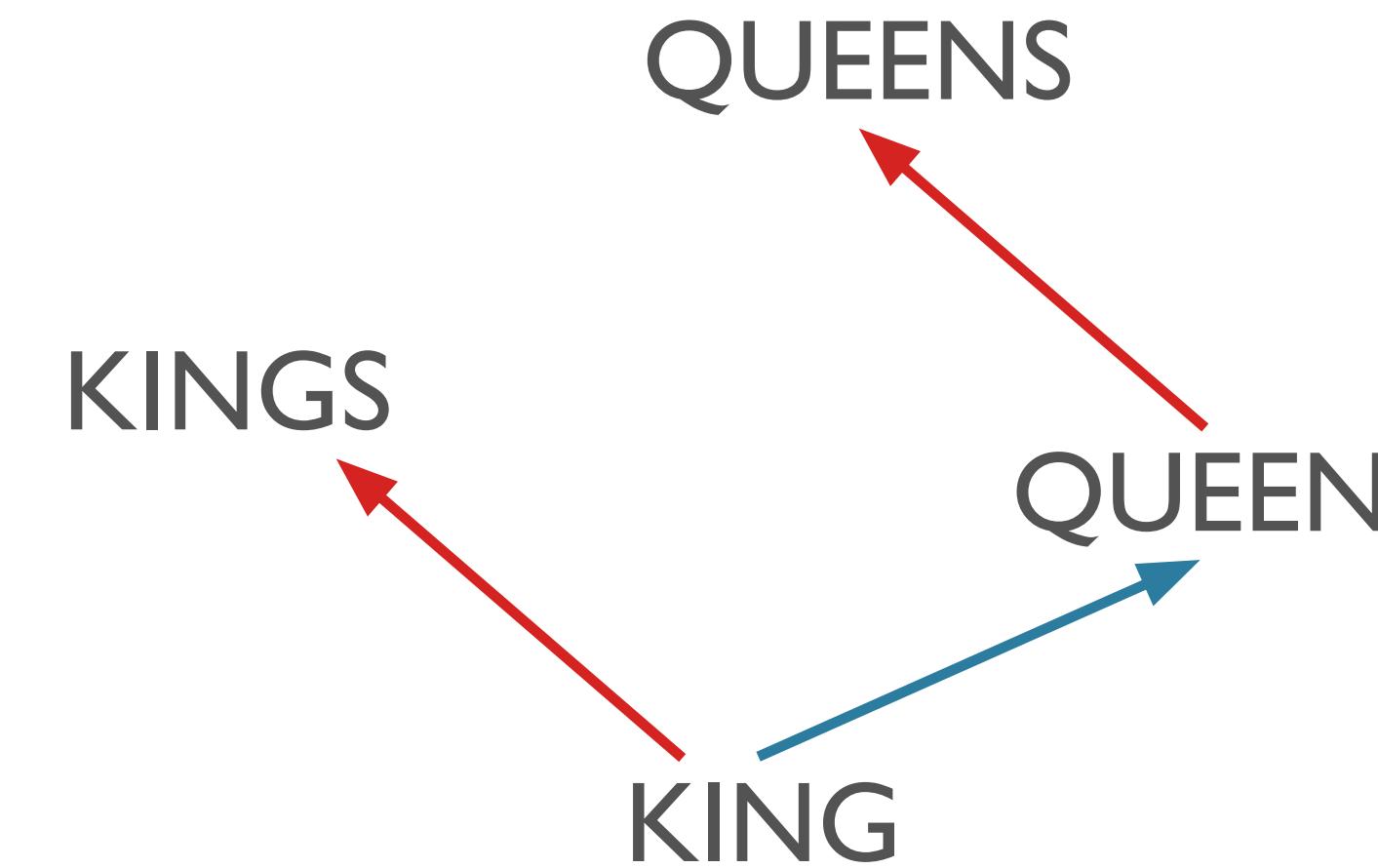
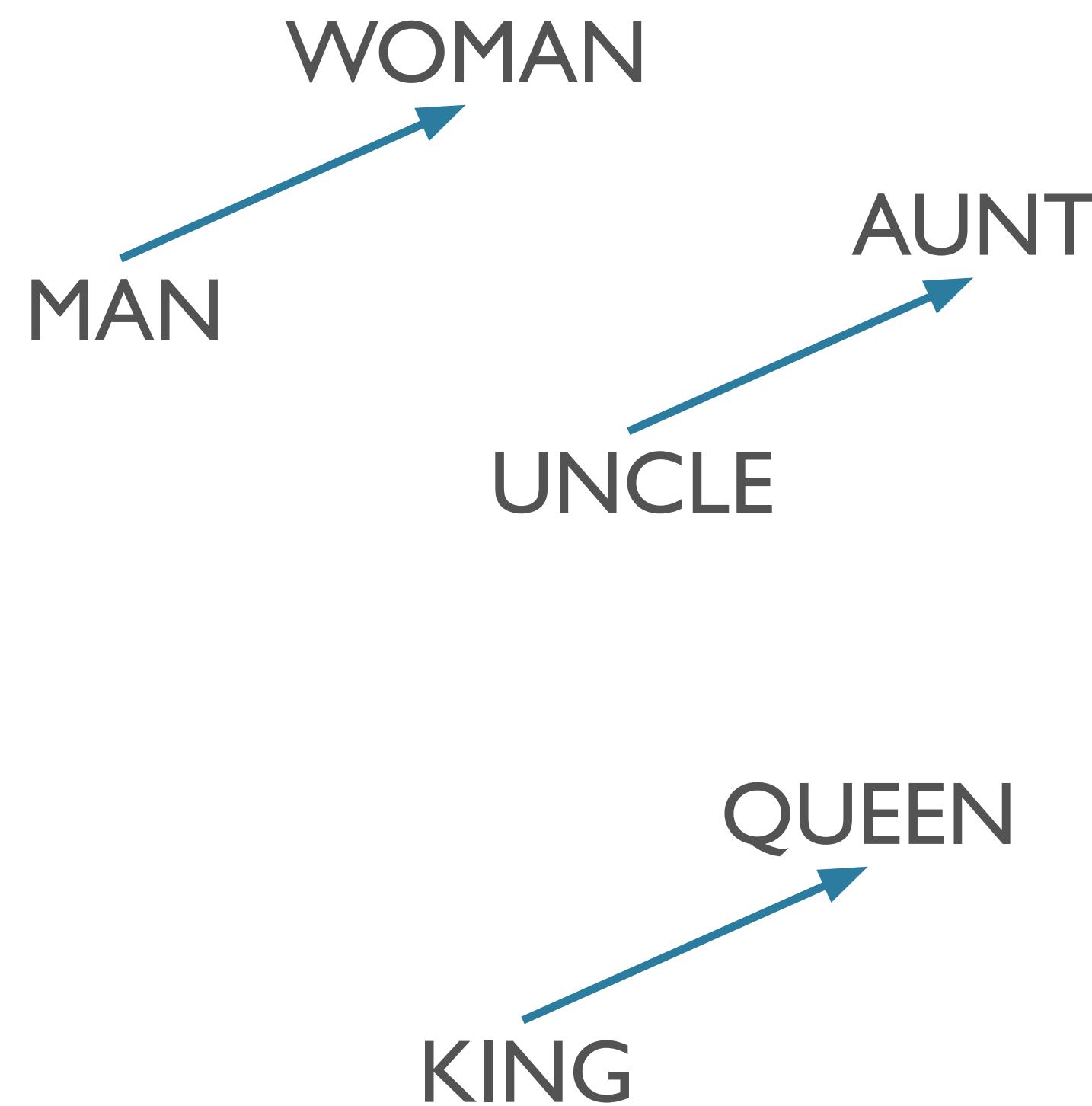
- Issue:
  - Denominator computation is very expensive
- Strategy:
  - Approximate by *negative sampling* (efficient approximation to Noise Contrastive Estimation):
    - + example: true context word
    - – example:  $k$  other words, randomly sampled
- Details: next time!

# Relationships via Offsets



Mikolov et al 2013b

# Relationships via Offsets



Mikolov et al 2013b

# One More Example

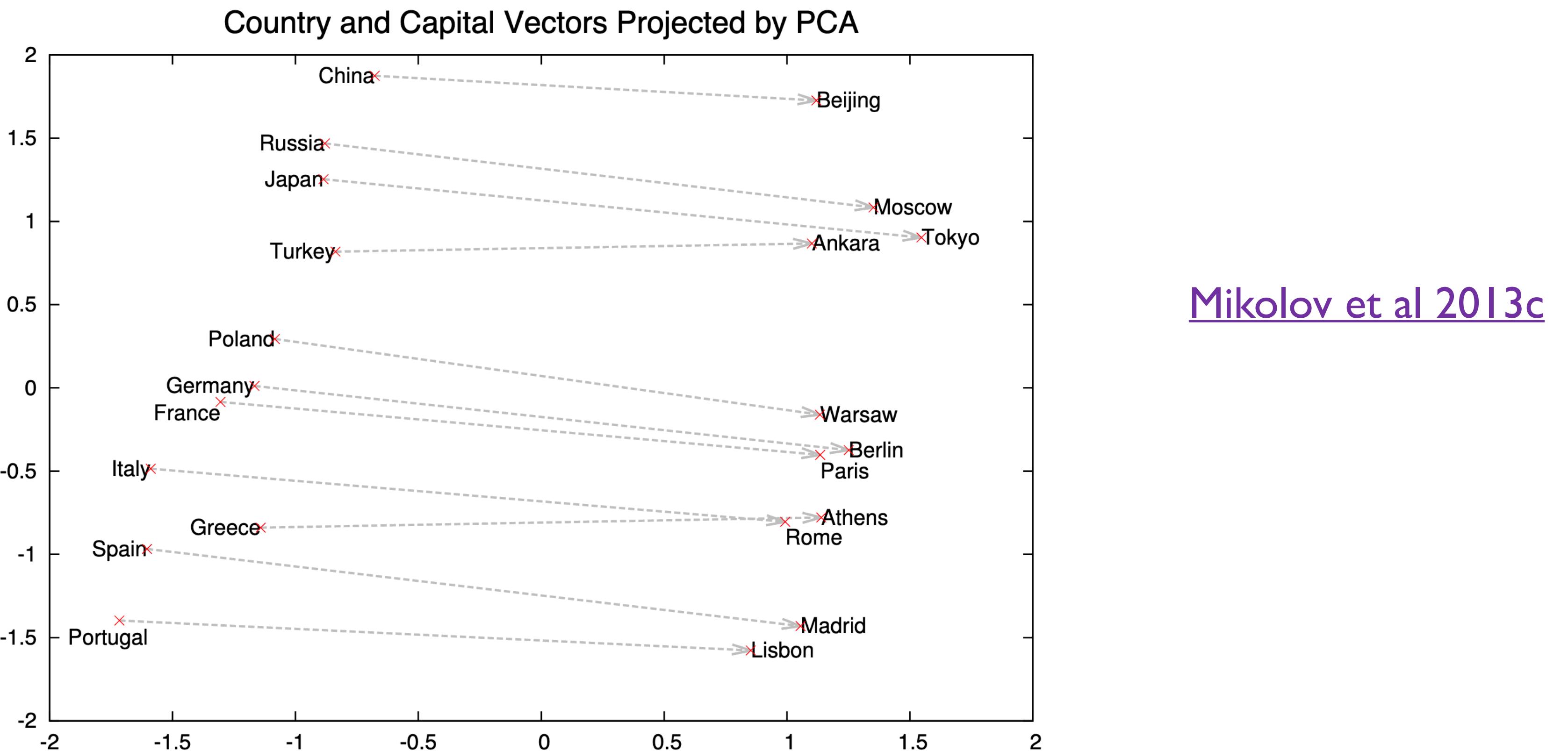
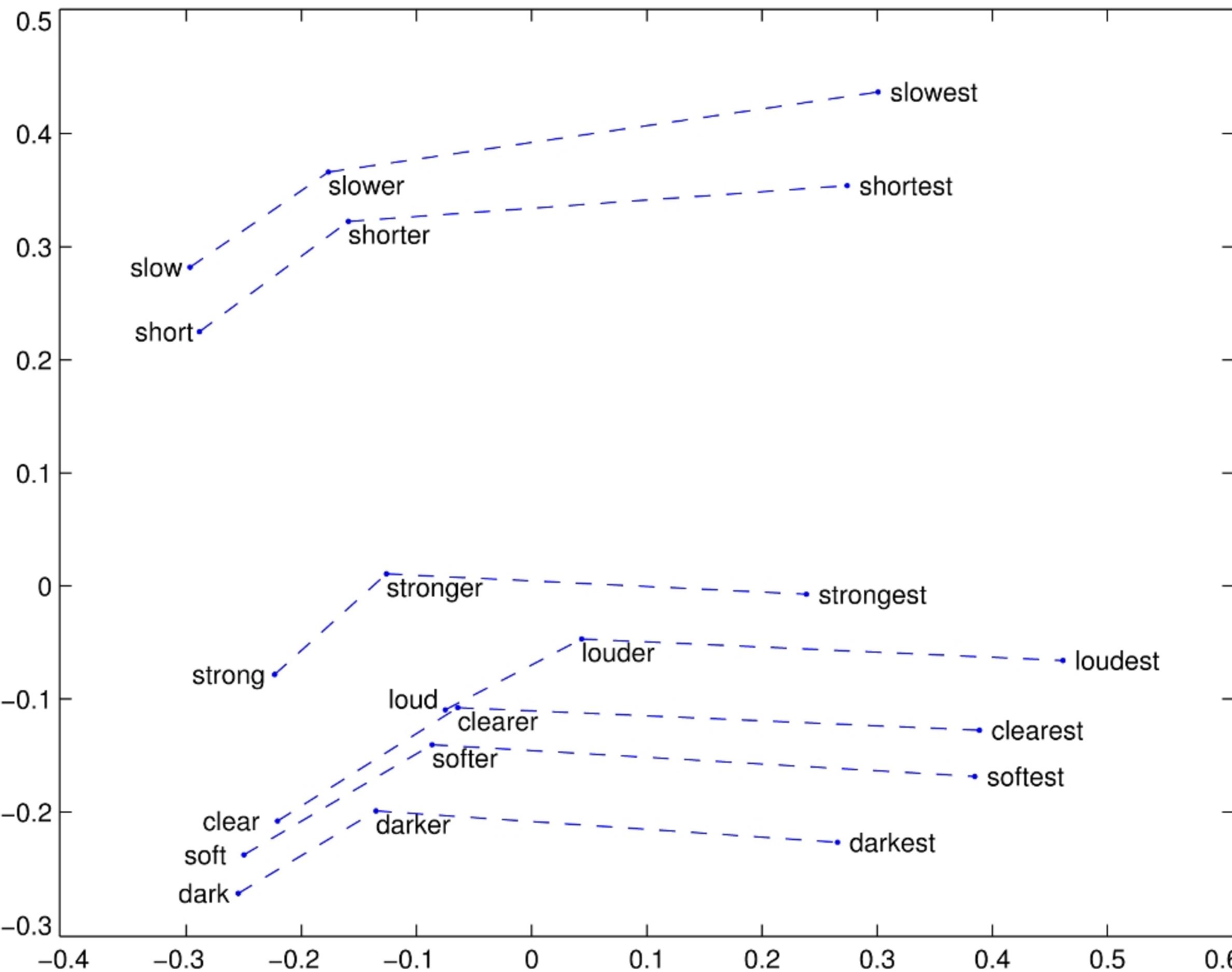


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised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

# 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.

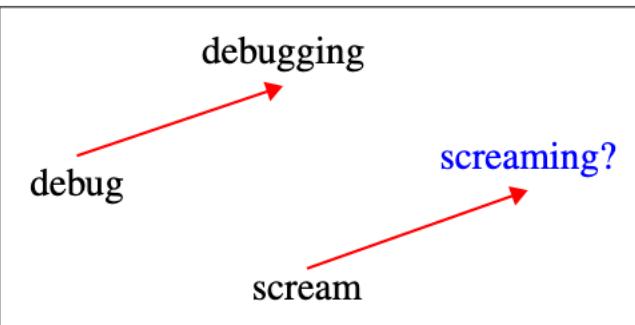


Figure 1: Using the vector offset method to solve the analogy task (Mikolov et al., 2013c).

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

$$a : a^* :: b : \underline{\quad} \quad (1)$$

Linzen 2016, a.o.

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

Tolga Bolukbasi<sup>1</sup>, Kai-Wei Chang<sup>2</sup>, James Zou<sup>2</sup>, Venkatesh Saligrama<sup>1,2</sup>, Adam Kalai<sup>2</sup>

<sup>1</sup>Boston University, 8 Saint Mary's Street, Boston, MA

<sup>2</sup>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

# Homework 1

[posting this afternoon]

# Learning Goals

# Learning Goals

- Get basic infrastructure [Anaconda, environment] set up for this course

# Learning Goals

- Get basic infrastructure [Anaconda, environment] set up for this course
- Build the very first building block for our NLP models: a Vocabulary

# Learning Goals

- Get basic infrastructure [Anaconda, environment] set up for this course
- Build the very first building block for our NLP models: a Vocabulary
- Reflect on dataset documentation, using data that we will use throughout the course

# 1. Installing Anaconda

- Anaconda lets you manage local environments for python and other tools
  - Avoid version conflicts across multiple projects
  - Get exactly the versions of packages you need
  - Helps reproducibility as well
- We've provided an environment in `/dropbox/20-21/575k/env`
- Install:
  - `wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64.sh`
  - `sh Anaconda3-2020.11-Linux-x86_64.sh`
  - `run_hw1.sh` shows you how to activate the environment

## 2. Implementing a Vocabulary

- At the base of every NLP system is a **vocabulary** object, containing:
  - Token → index
  - Index → token
  - These provide the interface between strings (tokens), and integer indices that will be used in our models (e.g. for looking up embeddings)
- `/dropbox/20-21/575k/hw1/vocabulary.py`
- **#TODO:** comments tell you where to write your own code
- Write small script to save various vocabularies from the SST dataset [see next slide]

# 3. Data Statement for SST

- For many assignments in this course, we will be using the [Stanford Sentiment Treebank](#)
  - Input: movie reviews
  - Output: discrete ratings (0-4) of the sentiment from very negative to very positive
  - Simple/cleaned version available in `/dropbox/20-21/575k/data/sst/`
- [Data Statements for NLP](#) [Emily M Bender and Batya Friedman]
  - Best practices for documenting dataset creation
    - Can help understand and mitigate biased models by clearly identifying the nature and source of the data [e.g. which populations]
  - For this assignment: answer (to the best of your ability, given the documentation of SST) the relevant questions that should go into a data statement

# Next Time

- Skip-Gram with Negative Sampling
  - How optimization framework applies to this problem
- Introduction of two tasks that we will use throughout the class
  - Language modeling
  - Text classification [sentiment analysis]