
DAY 1 · LANGUAGE MODELS · PART 2 OF 5

Language Models with Neural Networks

Word embeddings and feed-forward language models

Pavlos Protopapas

Harvard University

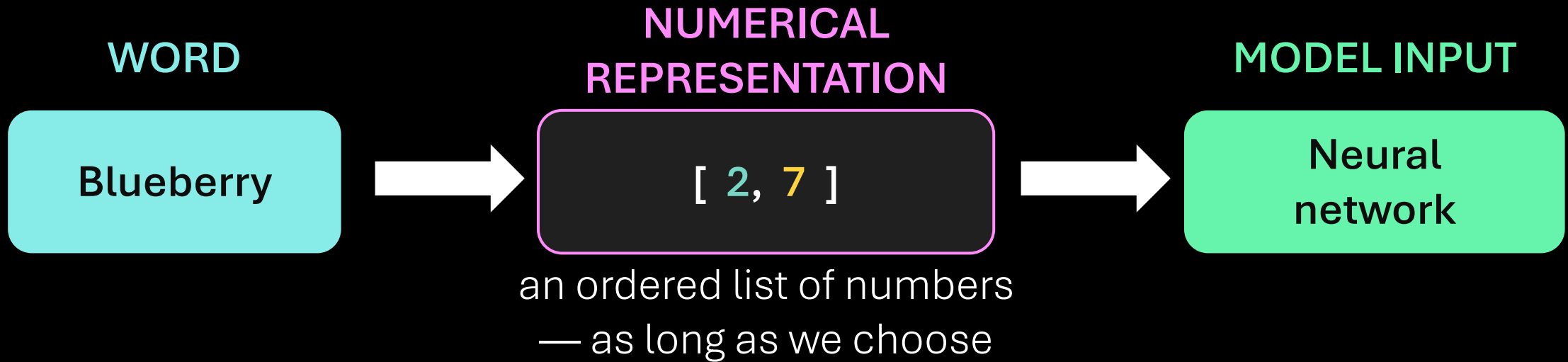


Outline

- › **Why Do We Need Neural Networks?**
- › Word Embeddings: Representing Words as Vectors
- › A Feed-Forward Neural Language Model
- › Training the Network: Loss and Backpropagation
- › Where We Go From Here

Neural Networks Operate on Numbers

Before a model can use a word, we must represent it numerically.



We will begin with deliberately interpretable toy values.

A Fruit Can Be Represented by Two Numbers

Toy vector components: **Size** · **Sweetness**

Fruit	Size	Sweetness
● Blueberry	2	7
● Kiwi	5	5
● Lemon	6	2
● Banana	7	6

What is a vector?

Just an ordered list of numbers.

1 · Size

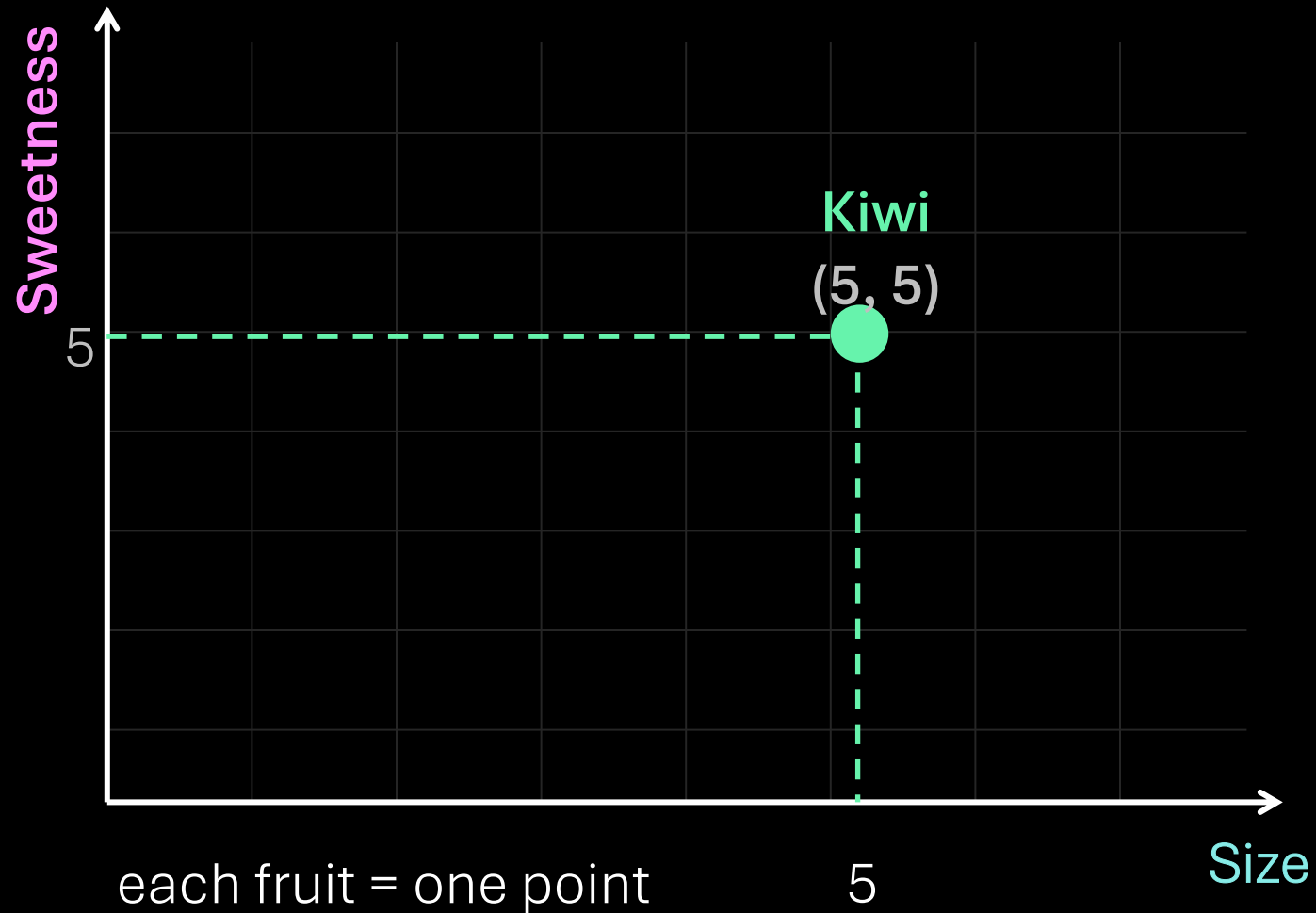
2 · Sweetness

The slots never move, so two fruits can be compared number by number.

each row is one vector **Blueberry** → [**2**, **7**] 3 numbers = 3 dimensions

Illustrative teaching values — deliberately simplified, not nutritional measurements.

A Vector is Just an Address in This Space

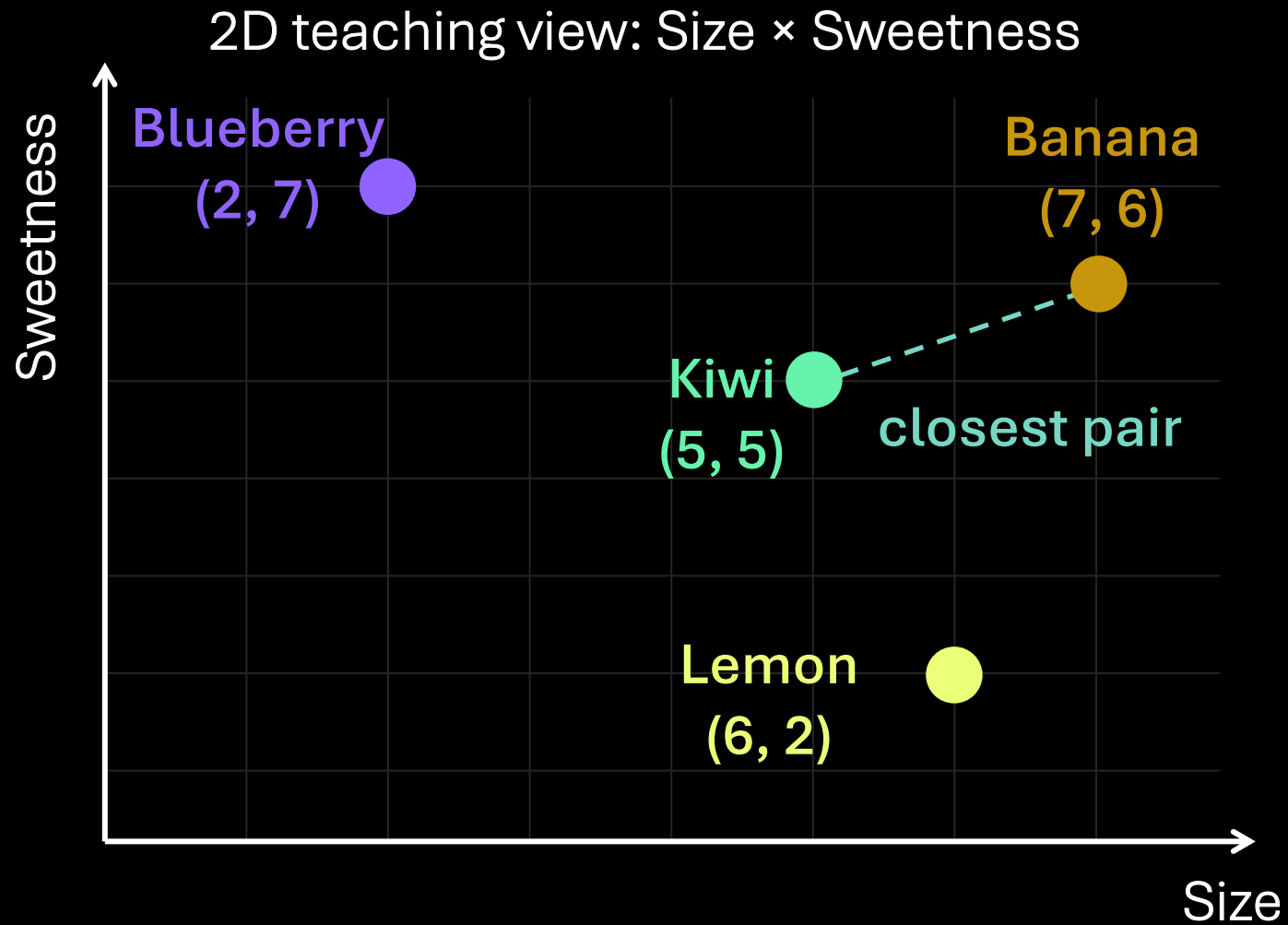


Kiwi = [5, 5]

- **First number** → how far along **Size**
- **Second number** → how far up **Sweetness**
- The order is fixed, so the two numbers always point to the same spot.

More numbers = more dimensions. Same idea — just harder to draw.

Nearby Points Represent More Similar Fruits



Nearby points share more of the selected characteristics.

Distance gives us a simple notion of similarity.

Two components mean we can draw every fruit as a point on this plane.

Illustrative teaching values—not nutritional measurements.

Word Embeddings Keep the Geometry, Not the Labels

Toy fruit vector

chosen and named by us

Size	2
Sweetness	7



The learned dimensions are usually not human-interpretable.

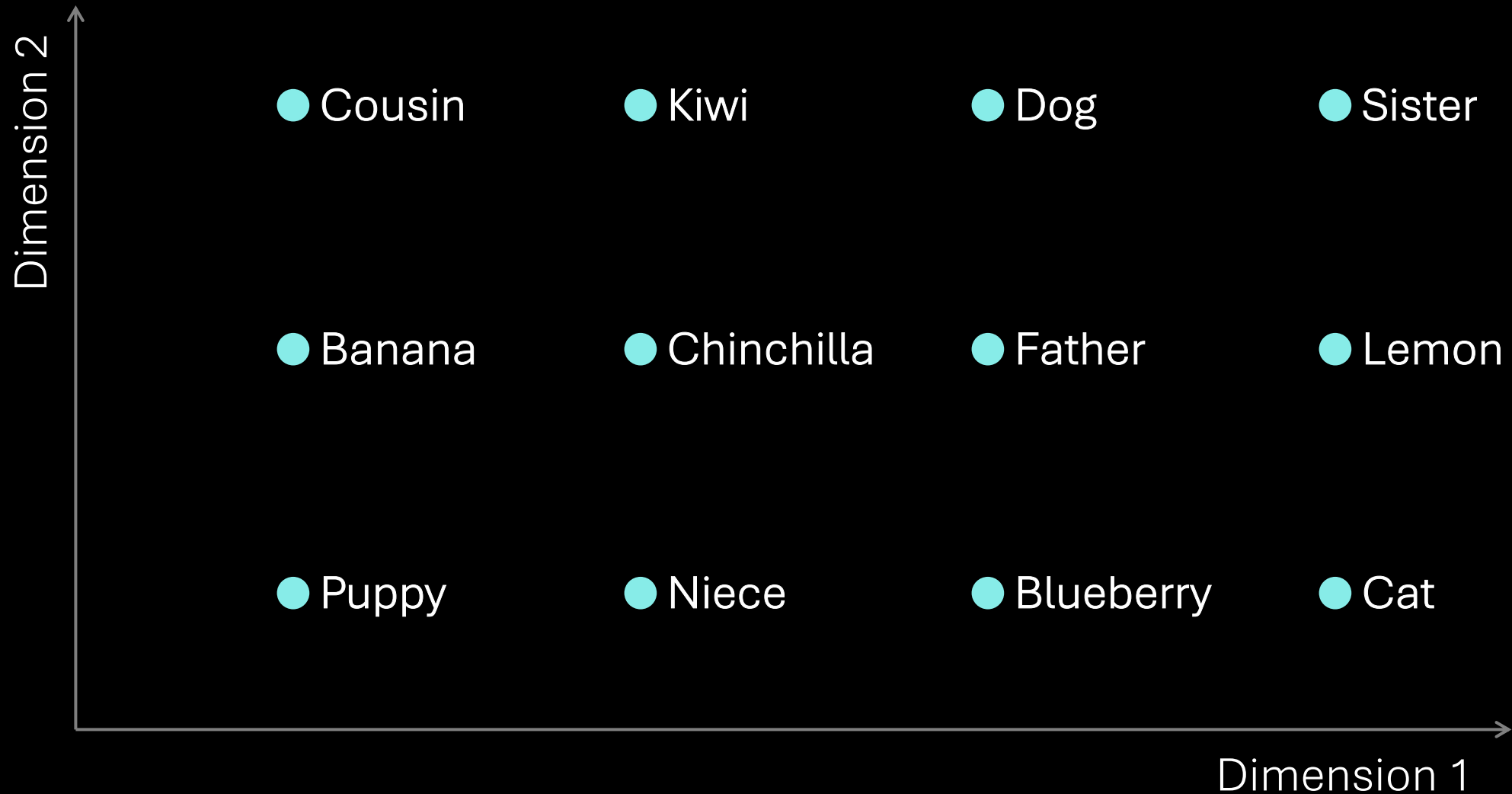
Real word embedding

learned from language data

d_1	0.12
d_2	-0.47
d_3	0.08
$\vdots$	$\vdots$
d_{200}	0.31

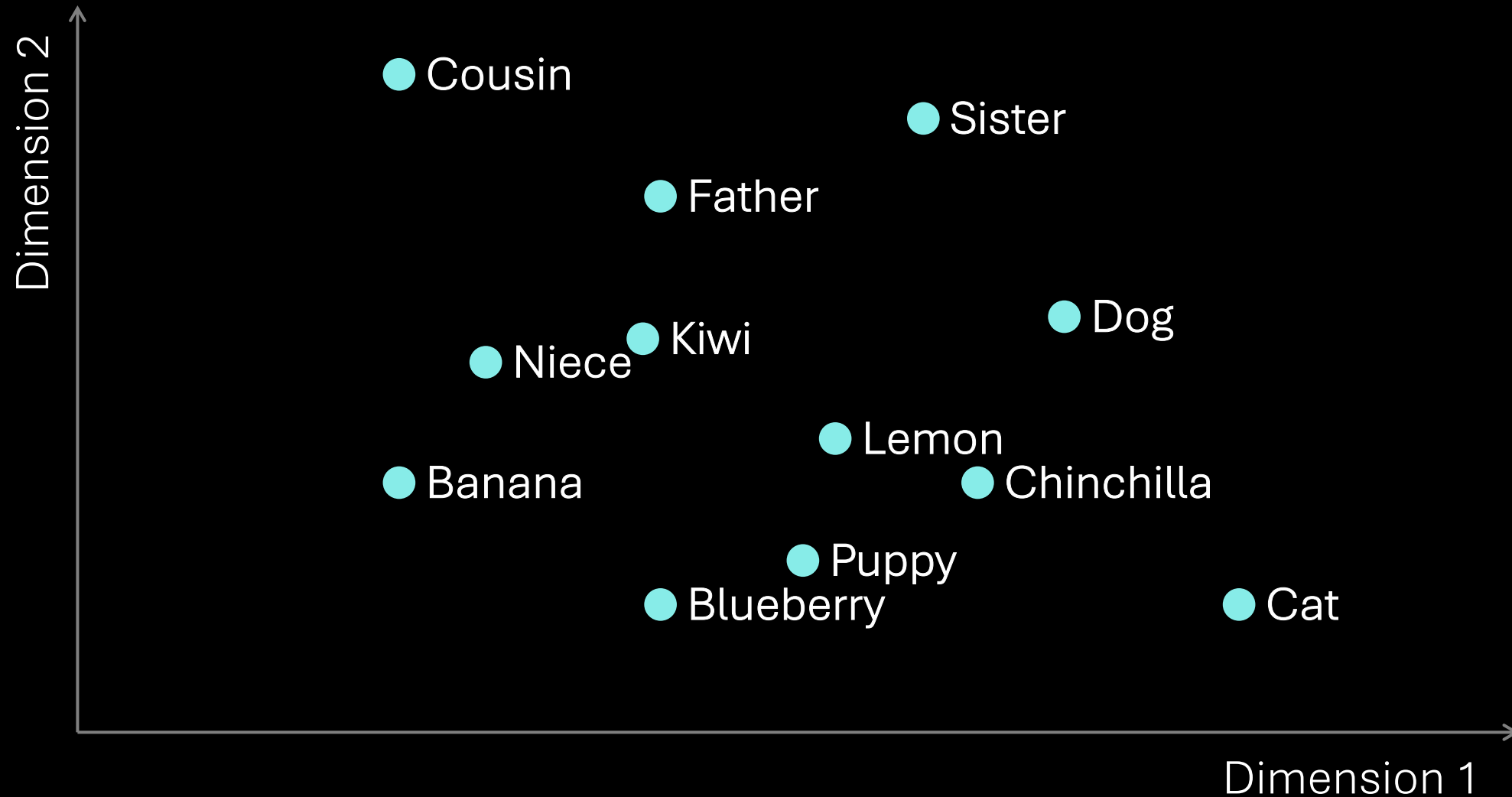
Word Embeddings Start Out Random

Twelve words, placed at random before any training.

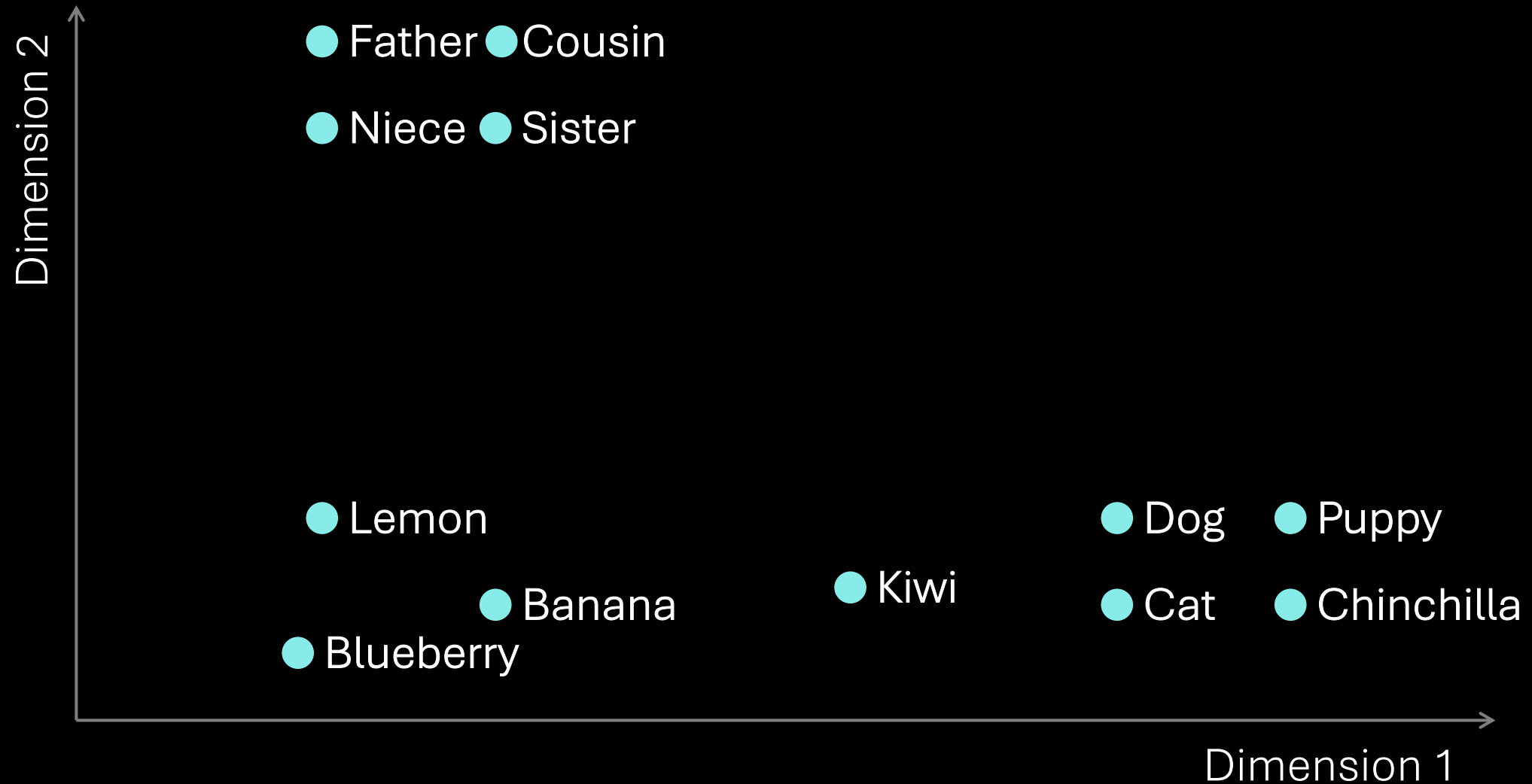


Training Pulls Related Words Together

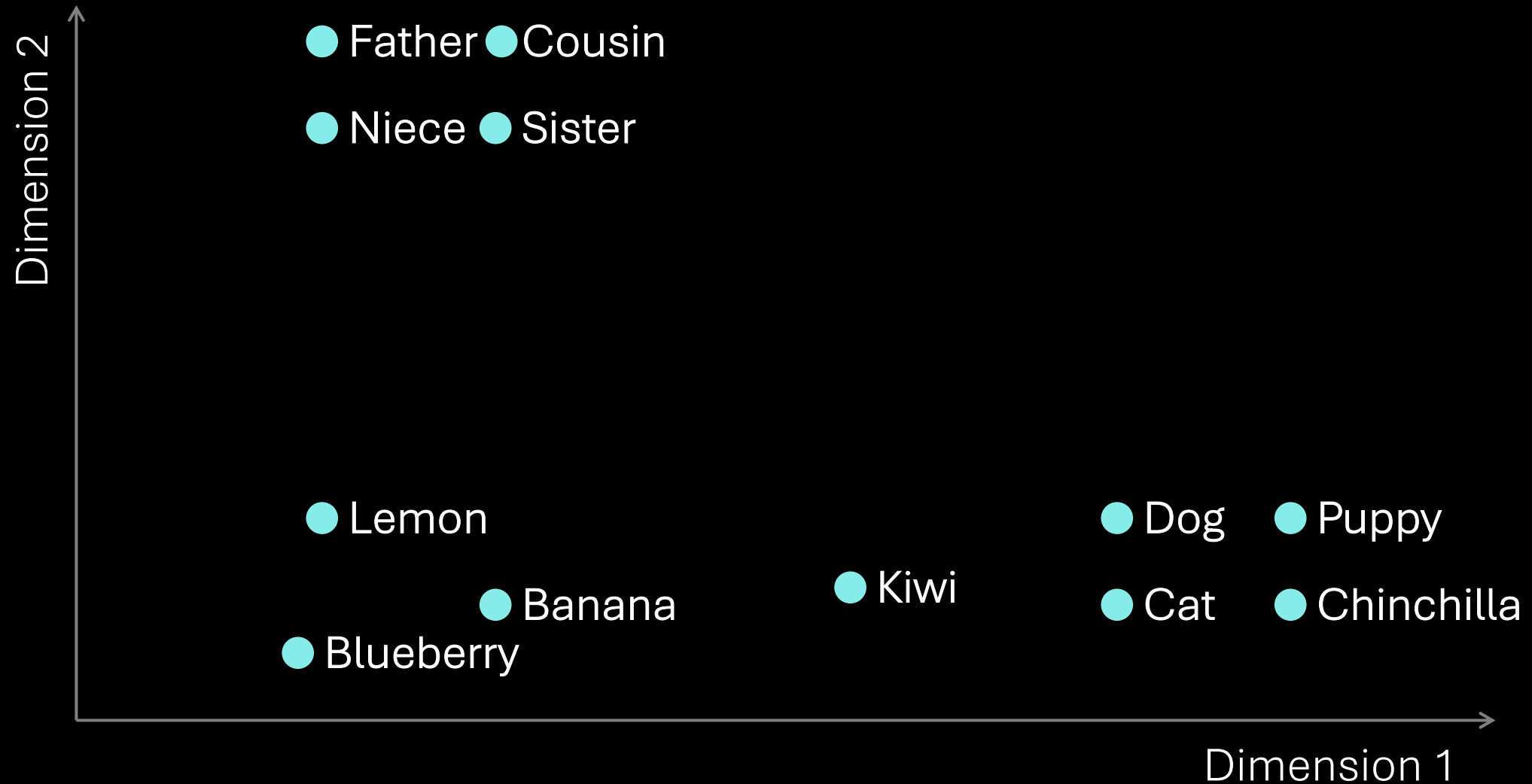
Training nudges related words closer together.



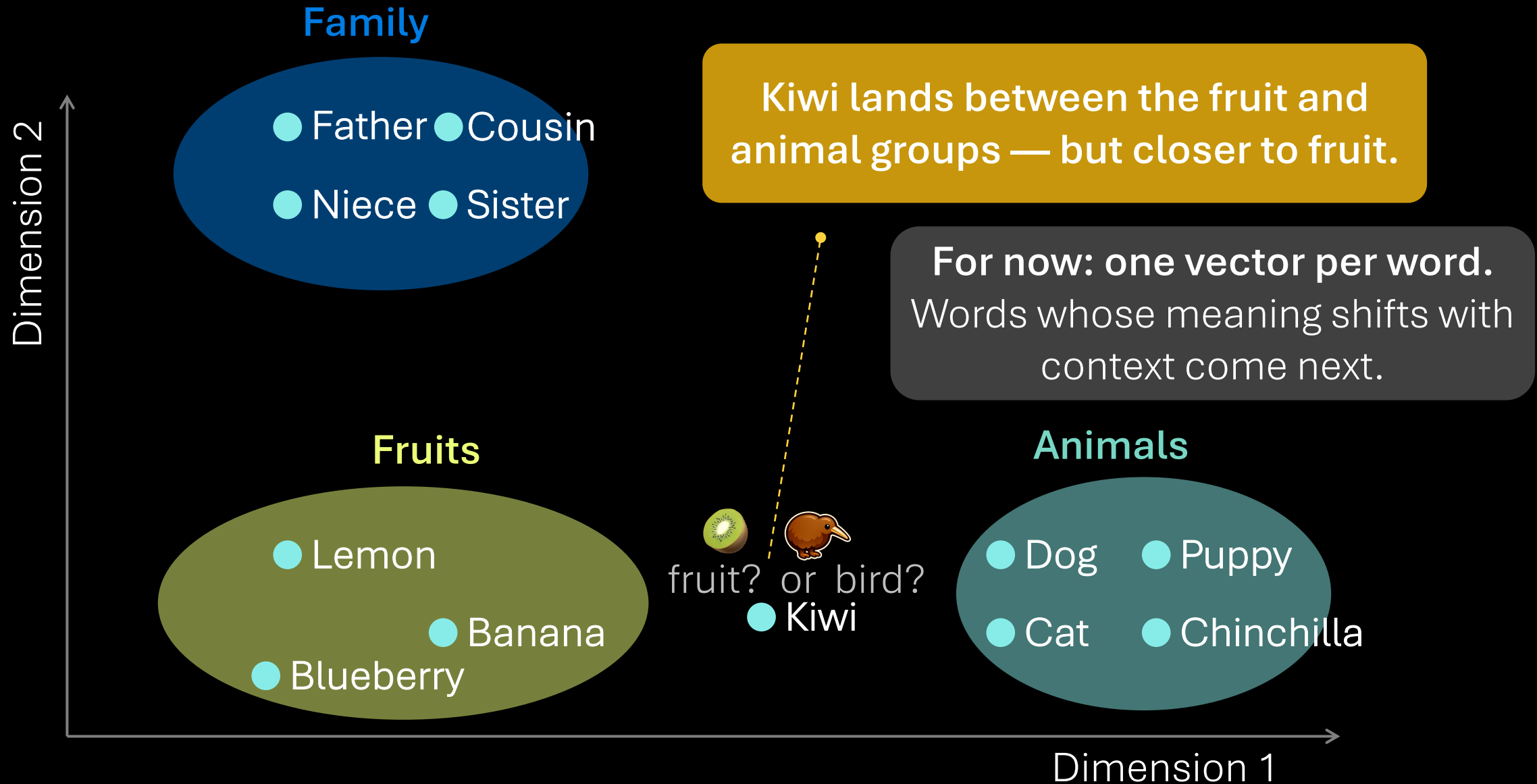
Learned Geometry Becomes Meaningful



Learned Geometry Becomes Meaningful



Clusters Form — and the Outliers are Informative



Learned Numbers | The Raw Values

Each cell is one learned floating-point scalar.

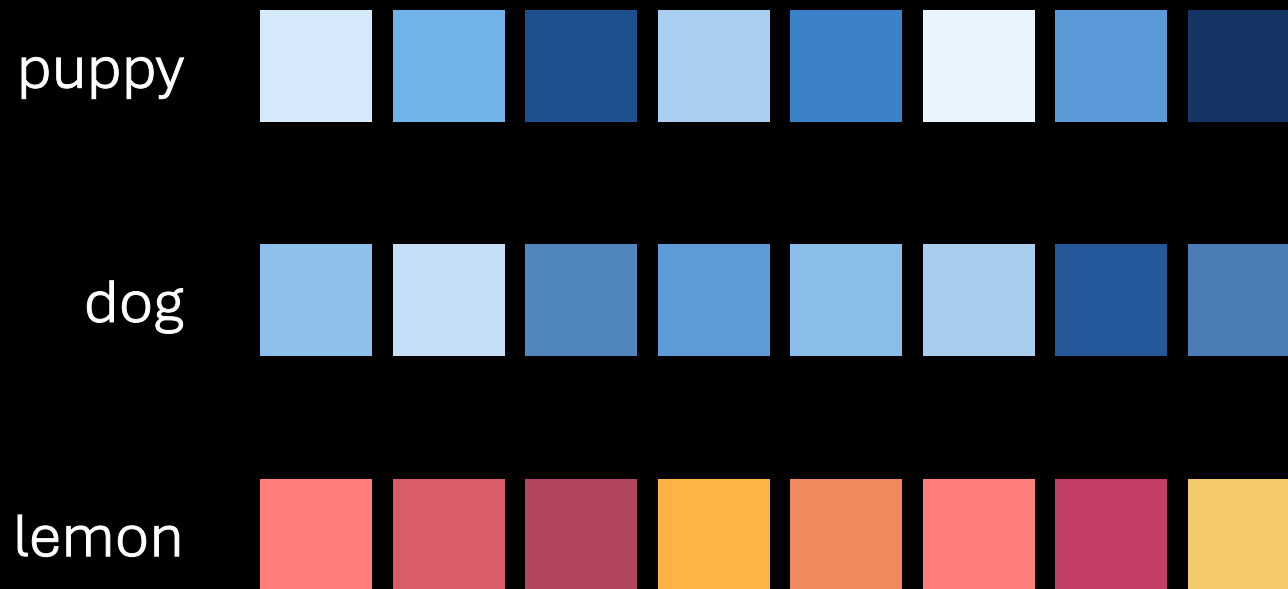
puppy	0.10	0.37	0.77	0.23	0.56	0.05	0.46	0.89
dog	0.29	0.15	0.55	0.45	0.31	0.24	0.73	0.58
lemon	-0.40	-0.53	-0.63	-0.39	-0.44	-0.40	-0.61	-0.32

Which two rows are similar? Hard to tell from raw numbers...



Learned Numbers Create Useful Geometry

Each cell is one learned floating-point scalar.



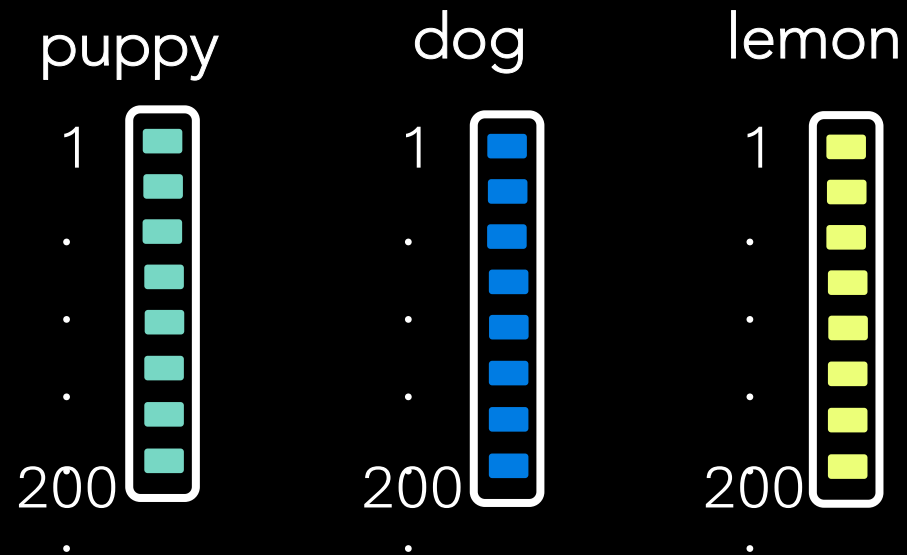
Similar words can land close together.

nearby vectors

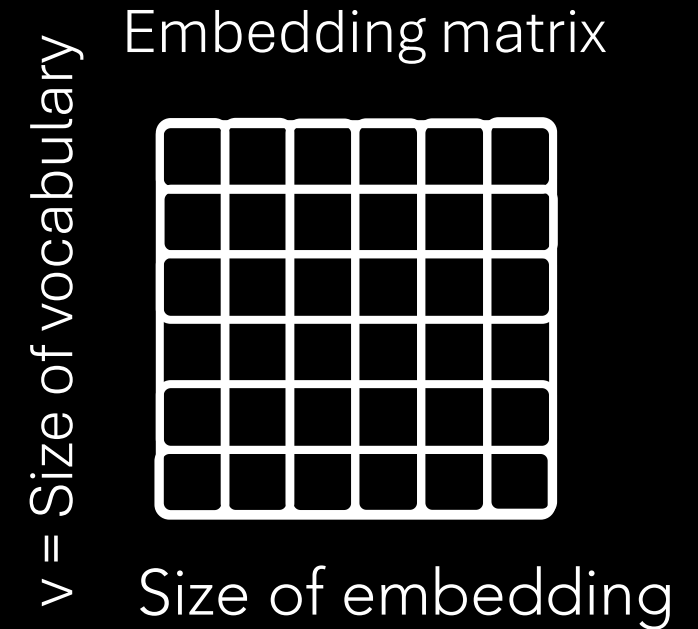
The individual dimensions are generally not interpretable.

Embedding Matrix | One Vector per Word

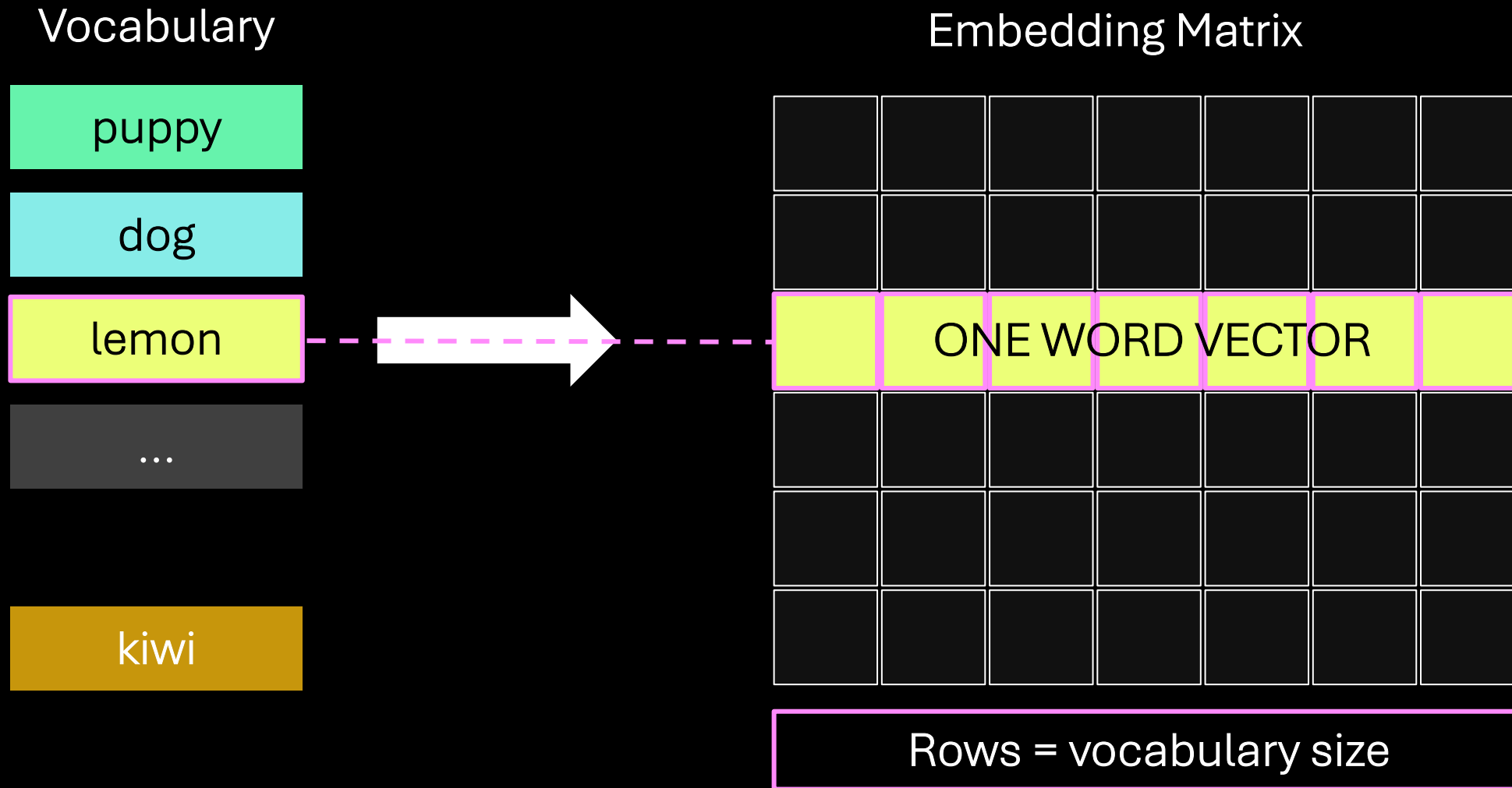
The embedding matrix stores one vector for every word in the vocabulary.



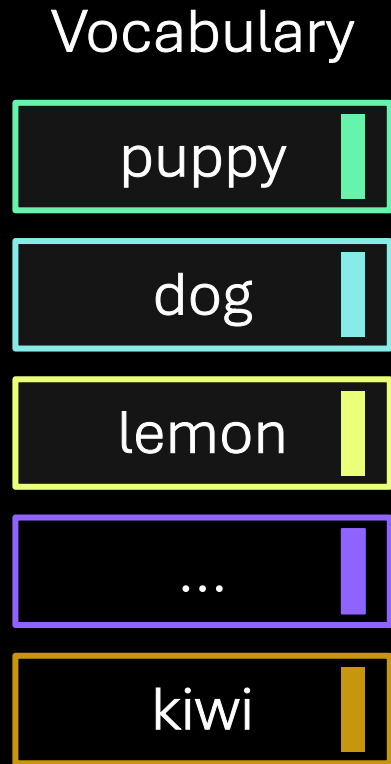
$X =$



Embedding Matrix | One Row per Word

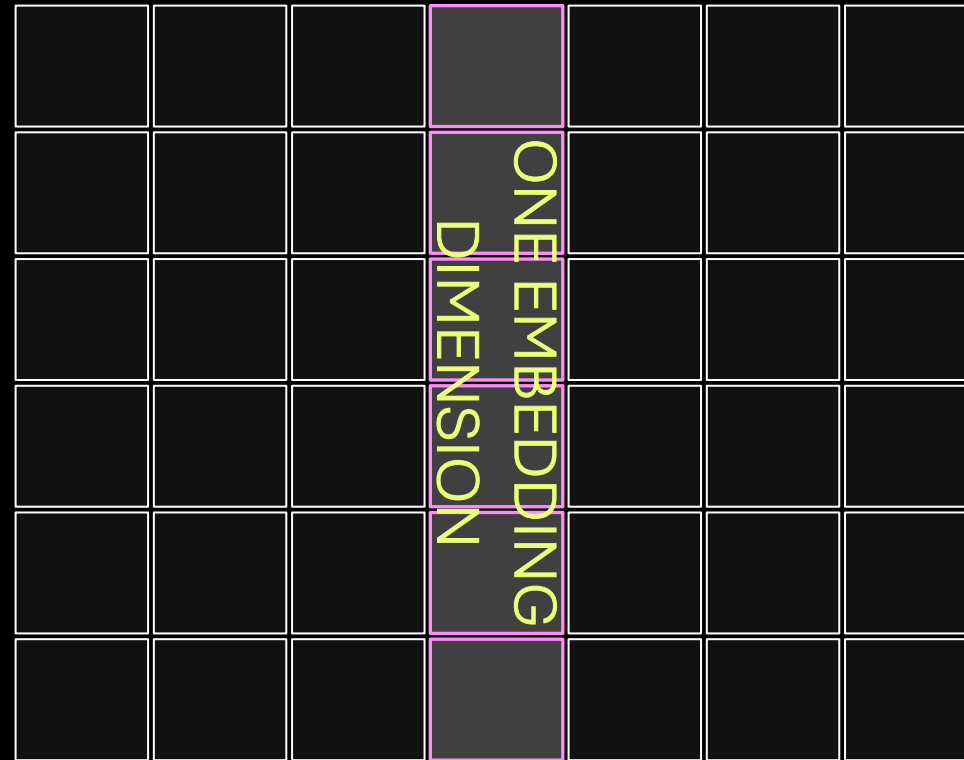


Embedding Matrix | One Column per Dimension



the same
component (the
nth) of every
embedding

Embedding matrix



d is a hyperparameter we choose,
e.g., 32, 64, 128 (768+ in modern LLMs)

Columns = embedding dimension d

In Practice, the Embedding Matrix Already Exists

Before we build a language model, we usually start from an embedding matrix that has already been trained.

Word2Vec Mikolov et al., 2013

Learns its vectors by predicting nearby words from context, across a huge collection of text.

GloVe Pennington et al., 2014

Learns its vectors from how often words co-occur with each other across the whole corpus.

Either way, the result is the same: one ready-made vector per word, the embedding matrix we plug into our model.

From Embeddings to a Language Model

How can we use these embeddings to build a Language Model?

Remember, we only need a system that can estimate:

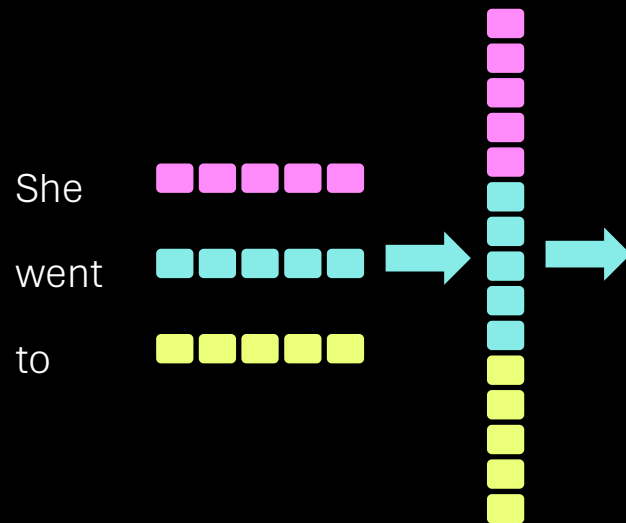
$$P(\underbrace{x_{t+1}}_{\text{next word}} \mid \underbrace{x_t, x_{t-1}, \dots, x_1}_{\text{previous words}})$$

Example input sentence



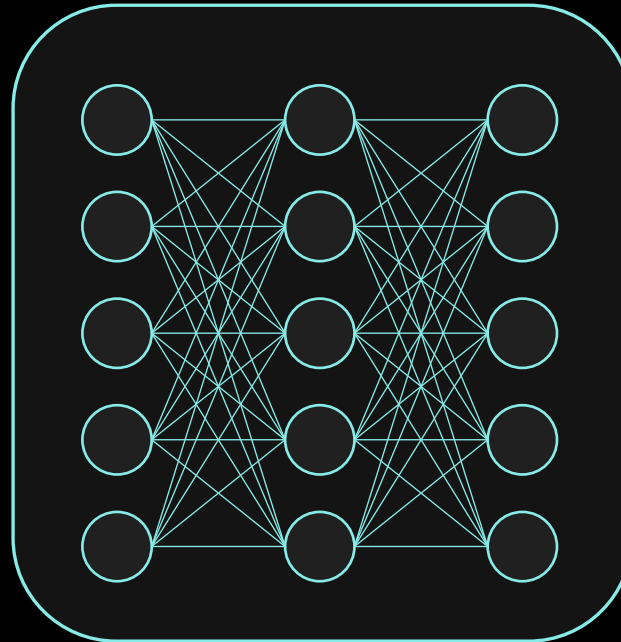
Inside the Box | Concatenate, Hidden Layers, Softmax

1. Concatenate the window



One long input vector: 3 words × 200 dims = 600 numbers
Each color block is one word's embedding — concatenation is the only “combining” here.

2. Hidden layers



hidden layers — fully connected, learning feature combinations

3. Output + softmax



softmax
scores → probabilities that sum to 1

Review | What the Hidden Layer and Softmax Do

Hidden layer

$$h = f(W_1x + b_1)$$

linear map, then an elementwise nonlinear activation

- › The linear part mixes every input dimension — each unit weights all 600 numbers.
- › The nonlinearity f (tanh or ReLU) is what lets the model represent non-linear structure.
- › h is a learned feature vector, not a probability — its entries can be any real number.

Softmax

$$p_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

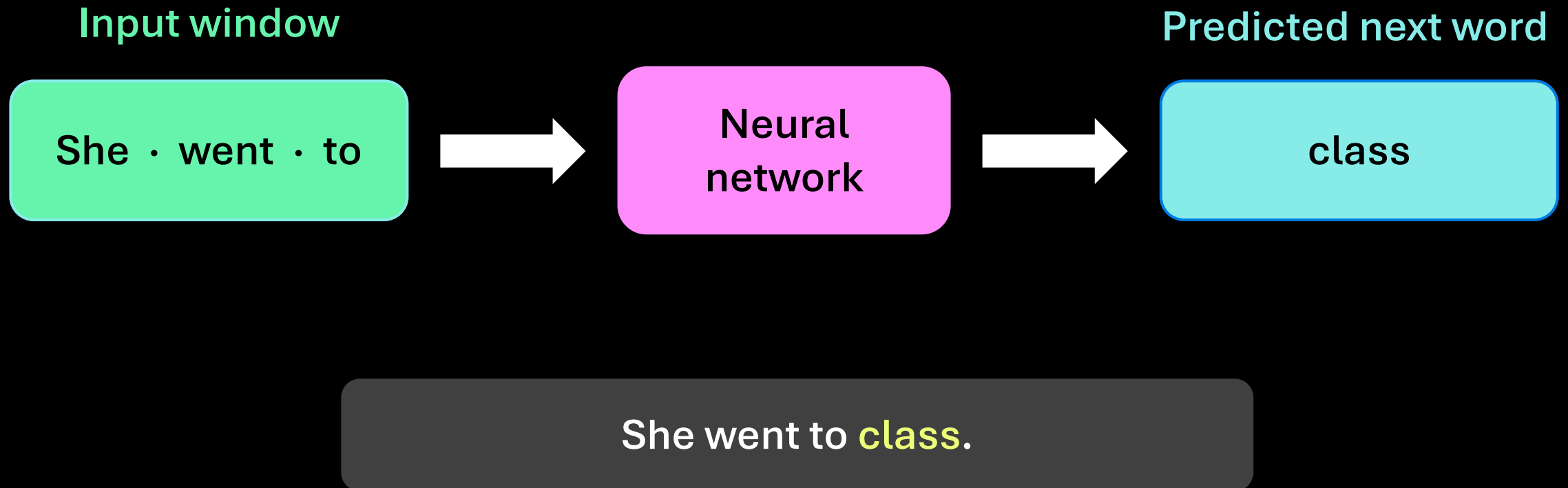
$$z = W_2h + b_2$$

- › Takes one real score per vocabulary word and returns a probability for each.
- › Exponentiating keeps every value positive; dividing by the sum forces them to add to 1.
- › Order is preserved: the largest score always gets the largest probability.

Both are standard building blocks — this slide is a refresher before we use them.

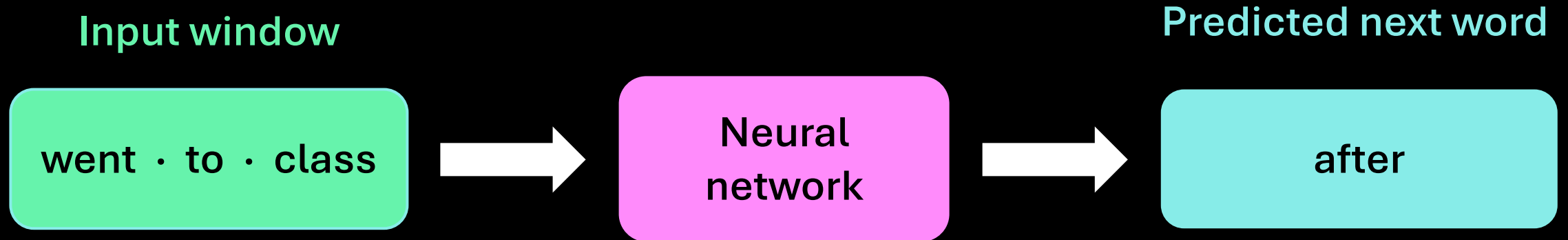
The Model Predicts the Next Word

Highest-probability continuation



The Same Network Predicts the Next Position

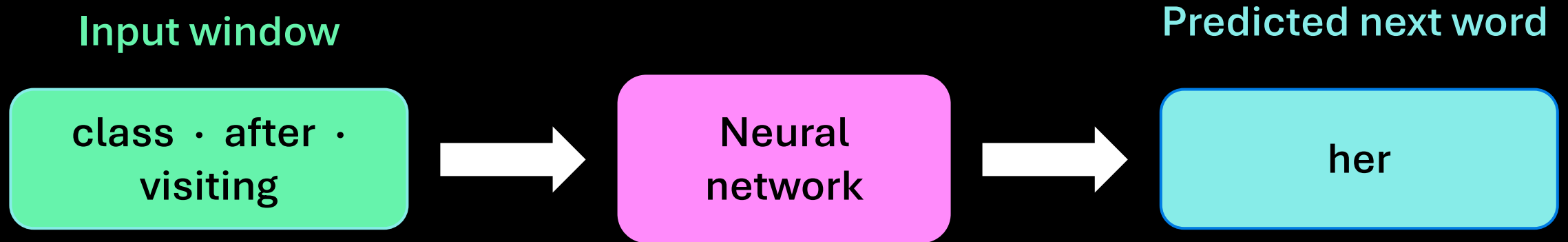
Move the fixed window one word forward.



Same neural network, new input window.

A Fixed Window Limits What the Model Can See

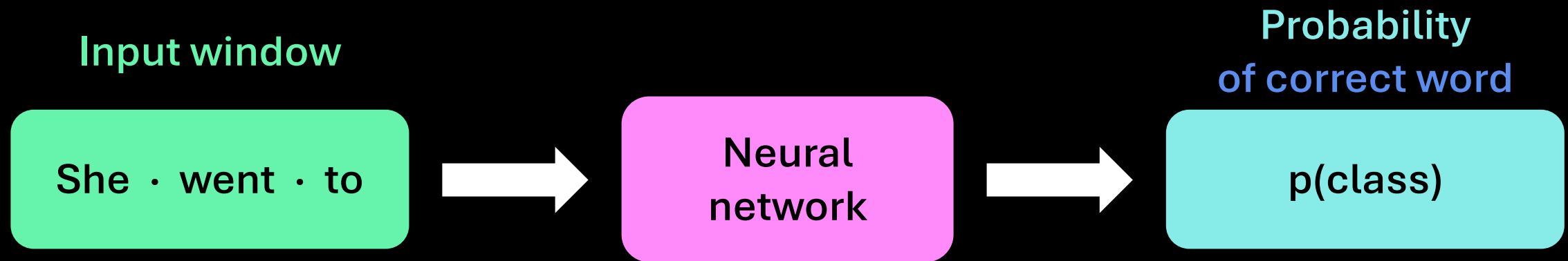
The model receives only the words in the current window.



The fixed window is all the model can see.

Next: how training pushes probability onto the correct word.

Training Rewards Probability on the Correct Next Word

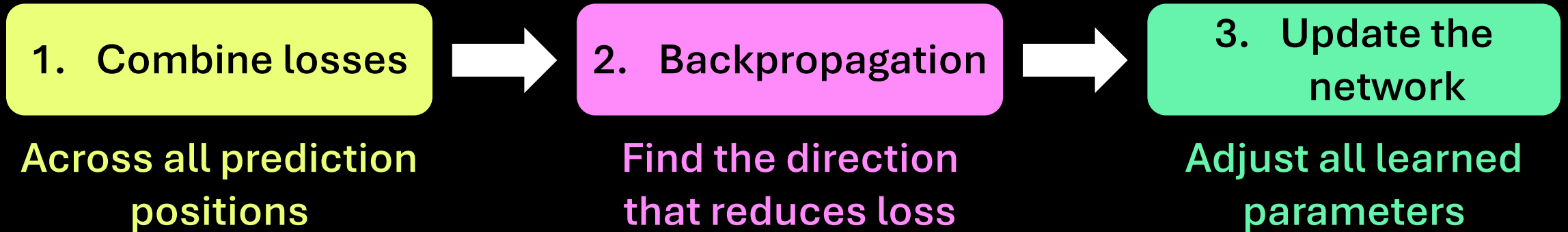


Correct next word: class

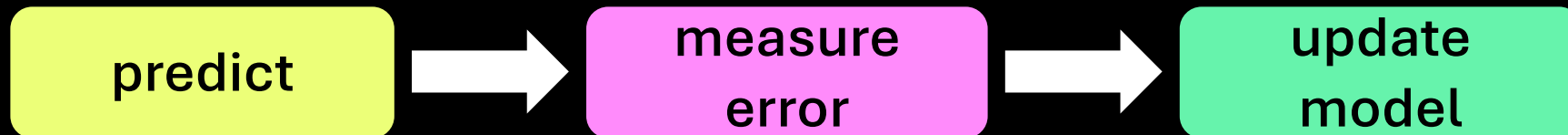
$$L = -\log p(\text{class})$$

Categorical cross-entropy for this example

Backpropagation Updates the Neural Network



Training loop



Repeat over many examples.

Feed-Forward Language Models Have a Fixed Window

Strengths

Dense embeddings
reduce sparsity

Learned parameters replace
explicit n-gram counts

Limitation

The input window has
a fixed size

A larger window adds many
more weights and still cannot
cover every dependency

We need a more flexible memory.



Outline

- › Why Do We Need Neural Networks?
- › Word Embeddings: Representing Words as Vectors
- › A Feed-Forward Neural Language Model
- › Training the Network: Loss and Backpropagation
- › **Where We Go From Here**

Up Next

Next: attention and transformers.



COUNT EVERY 4-GRAM YOU EVER SAW



SHOVE 4 WORDS INTO A NEURAL NET



JUST REMEMBER THE WHOLE SENTENCE