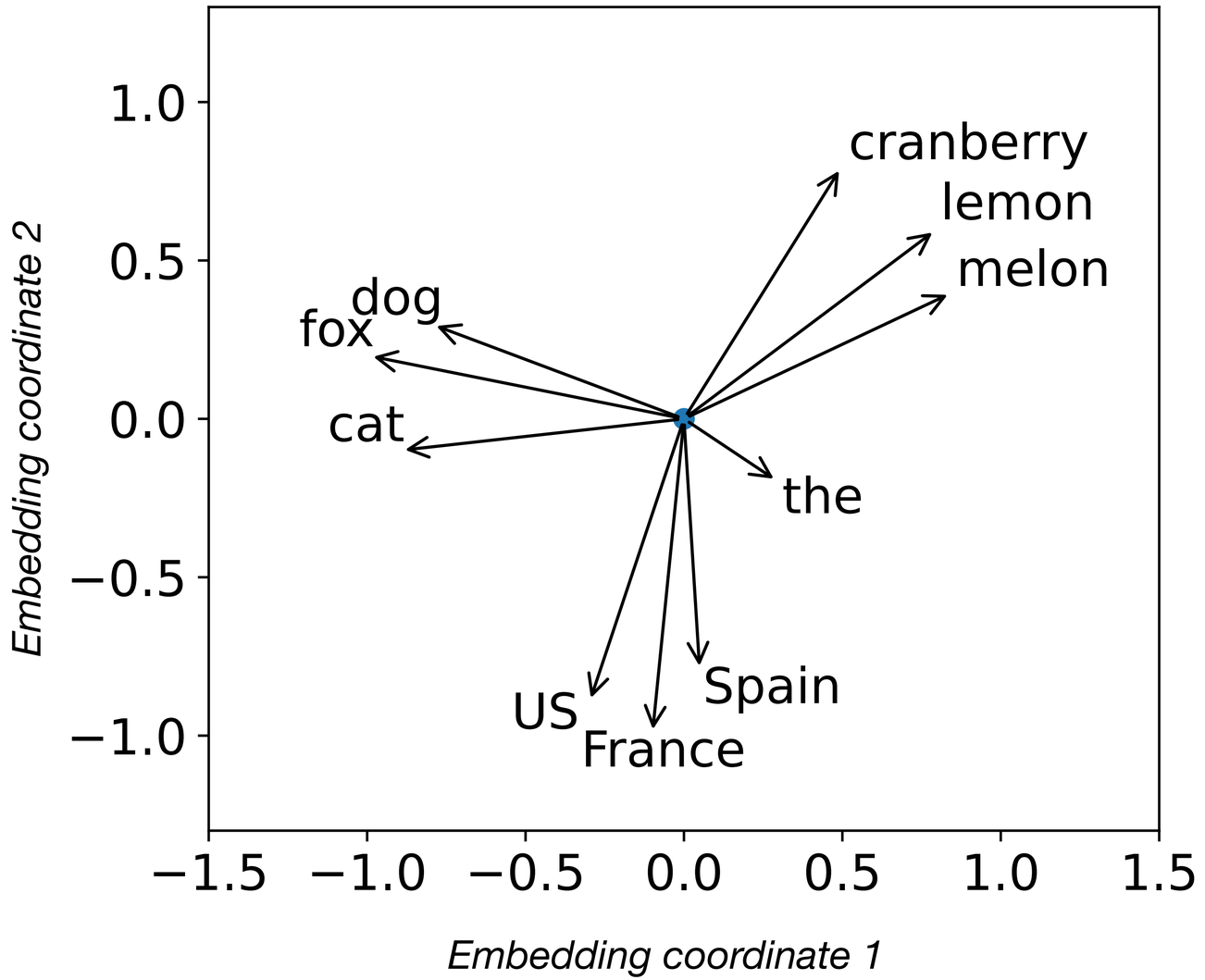
Vector-space embeddings

10-701 Introduction to Machine Learning

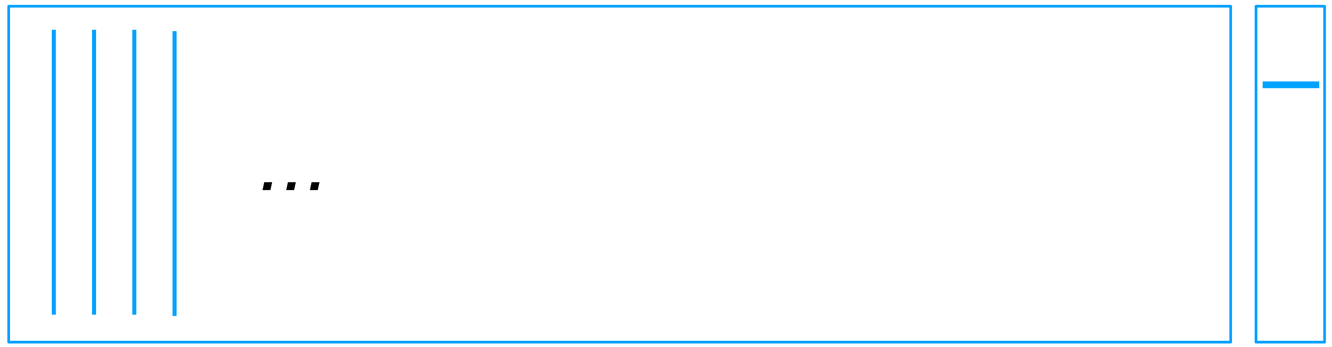
Geoff Gordon and Pradeep Ravikumar

Vector-space embedding



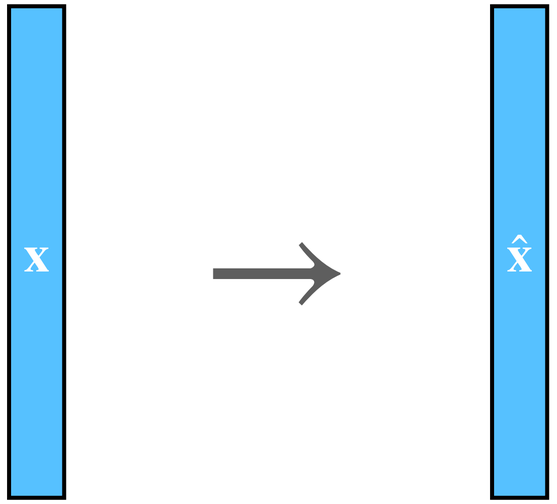
Arrows = embeddings $\mathbf{v}_j$ *(here, in* $\mathbb{R}^2$*)*

Where do embeddings come from?

$V e_{\text{dog}} =$ 

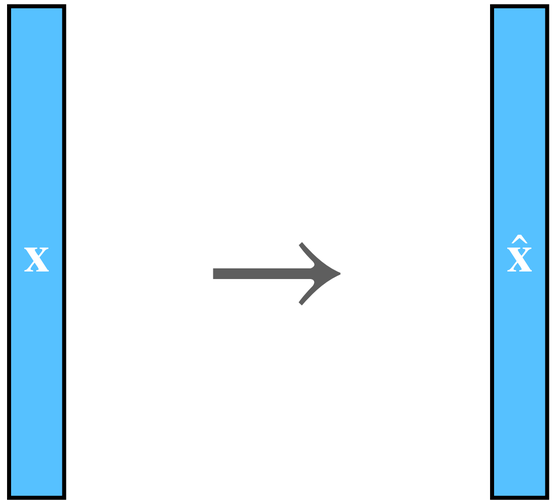
- If we present tokens as one-hot in *any* task, first layer automatically learns a vector-space encoding $V = W_1$
- But we can often improve embeddings w/ a ***surrogate*** or ***proxy*** task
  - a task we don't directly care about, but which is related enough to help learning the task we do care about, and for which we can easily/cheaply get lots of training data
- Method: pre-train on proxy task, or train w/ positive linear combination of losses from true and proxy tasks

Proxy task: autoencoder



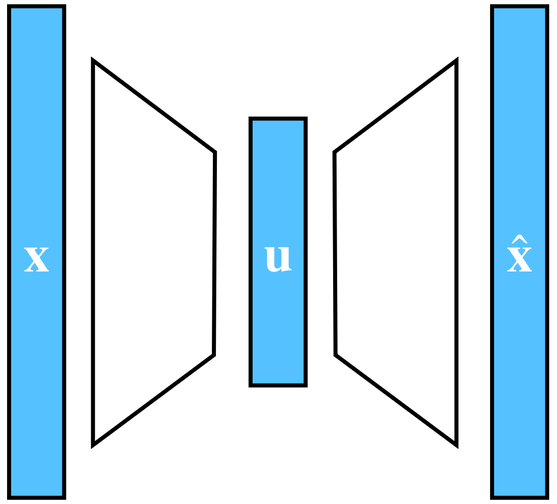
- Predict a sequence of tokens $\mathbf{x}^{(i)}$ from input $\mathbf{x}^{(i)}$

Proxy task: autoencoder



- Predict a sequence of tokens $\mathbf{x}^{(i)}$ from input $\mathbf{x}^{(i)}$
  - seems kind of easy

Proxy task: autoencoder



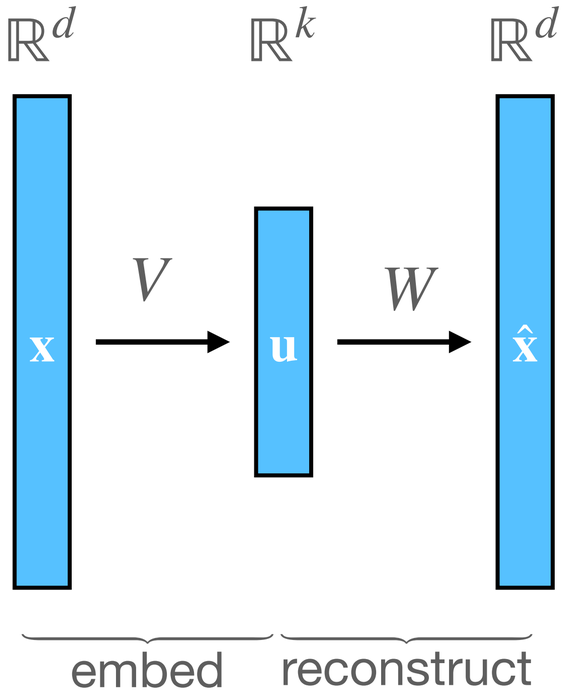
Why the name? "Code" = hidden activations, so $\mathbf{x}$ encodes (and then decodes) itself

- Predict a sequence of tokens $\mathbf{x}^{(i)}$ from input $\mathbf{x}^{(i)}$
  - seems kind of easy
- The catch: something about the model (the ***bottleneck***) prevents us from just copying input to output
  - e.g., hidden layer $\mathbf{u}^{(i)}$ w/ too few dimensions
  - e.g., regularizer that disfavors straight copying

Why auto-encoding?

- Autoencoding is a good proxy task because:
  - it's related to many other tasks: the hidden vector $\mathbf{u}$ has to be a decent summary of the token string
  - it's really easy to get training data: any natural string of tokens is fair game (a novel, a blog post, a product review, a newspaper article)
- Hyperparameter: granularity
  - could pass in an entire newspaper article, or split it into paragraphs, or take smaller fixed-length windows of tokens

Linear autoencoder



$$\hat{\mathbf{x}} = W\mathbf{u} = WV\mathbf{x}$$

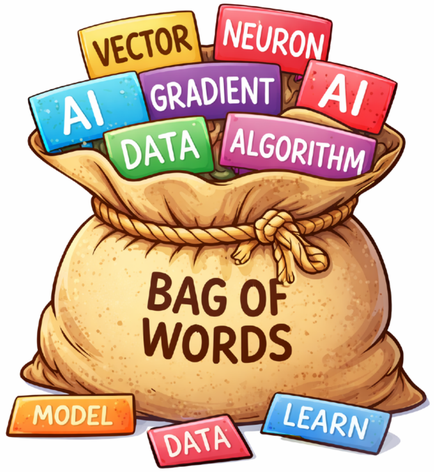
$$V \in \mathbb{R}^{k \times d}, W \in \mathbb{R}^{d \times k}$$

$$k \ll d$$

- Simplest autoencoder: one hidden layer, no nonlinearities
  - min sum squared error: $\min_{V,W} \sum_{i=1}^{N} \|WV\mathbf{x}^{(i)} - \mathbf{x}^{(i)}\|^2$
  - called *reconstruction error*
  - can solve e.g. w/ SGD or linear algebra

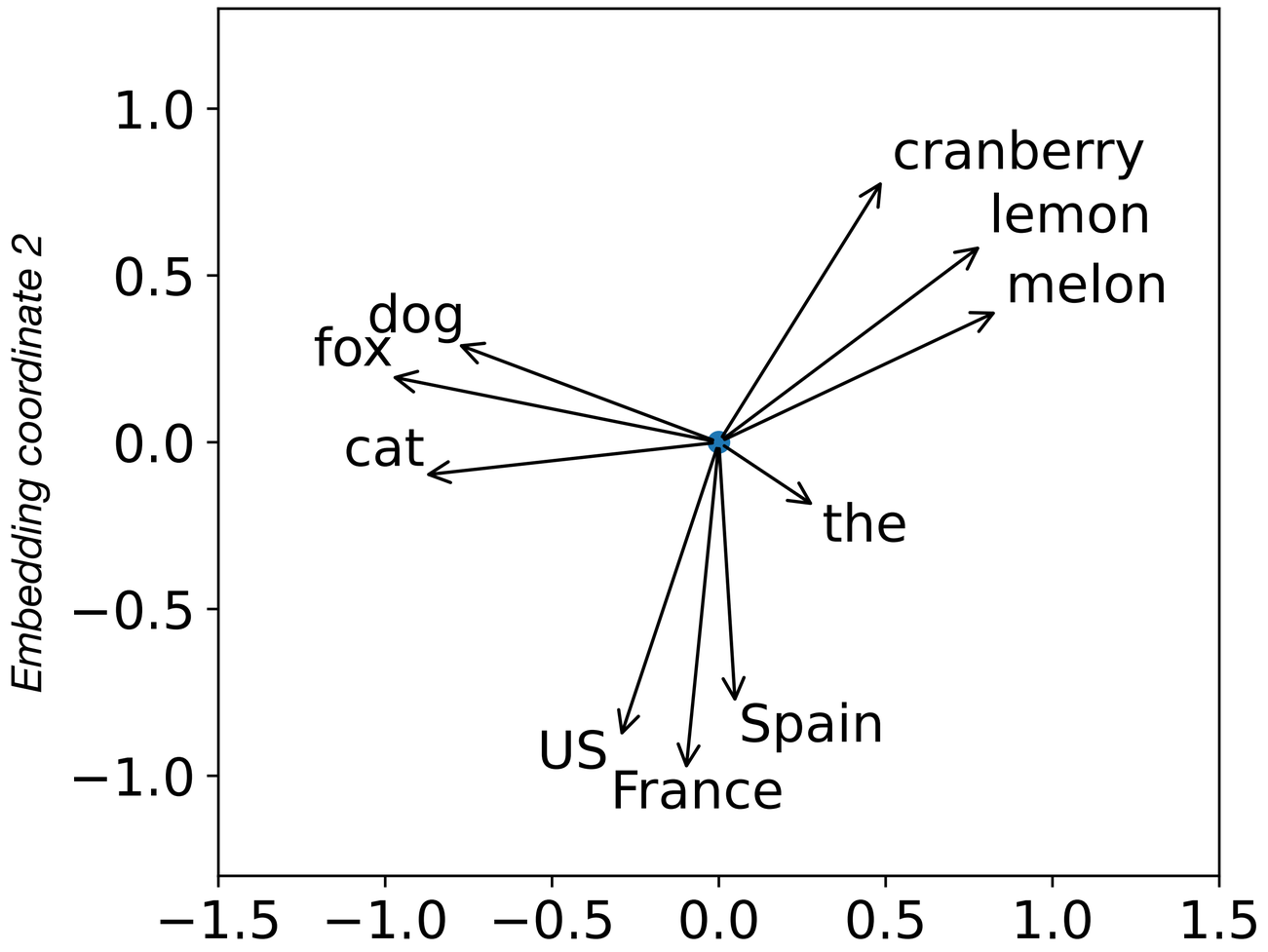
*Linear autoencoders are strongly related to the **principal components analysis** model (covered soon)*

Bag of words



- One way to get vector-space embeddings: fit a linear autoencoder to a dataset of strings of tokens
- How can we pass a token string to linear autoencoder?
  - simplest way: sum up all the tokens, “quick brown fox” $\rightarrow \mathbf{e}_{\text{quick}} + \mathbf{e}_{\text{brown}} + \mathbf{e}_{\text{fox}} \rightarrow \mathbf{v}_{\text{quick}} + \mathbf{v}_{\text{brown}} + \mathbf{v}_{\text{fox}}$
  - permutation-invariant (order doesn’t matter)
  - i.e., a ***multiset*** or ***bag*** of words
- BoW assumption is only to make model simpler: “man bites dog” really is different from “dog bites man”

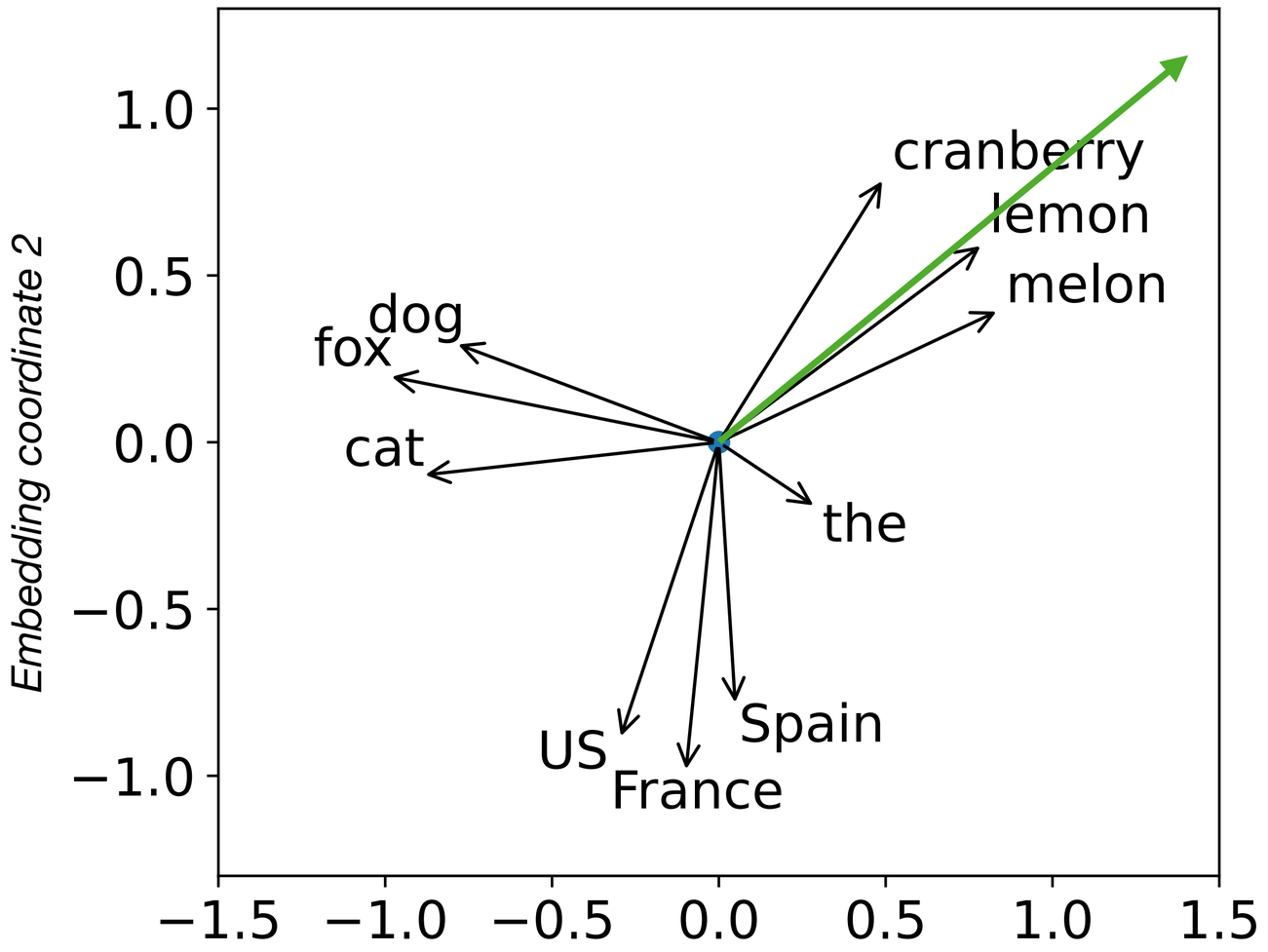
BoW example



Arrows = embeddings $\mathbf{v}_j$ *(here, in* $\mathbb{R}^2$*)*

- Embed “cranberry melon”

BoW example

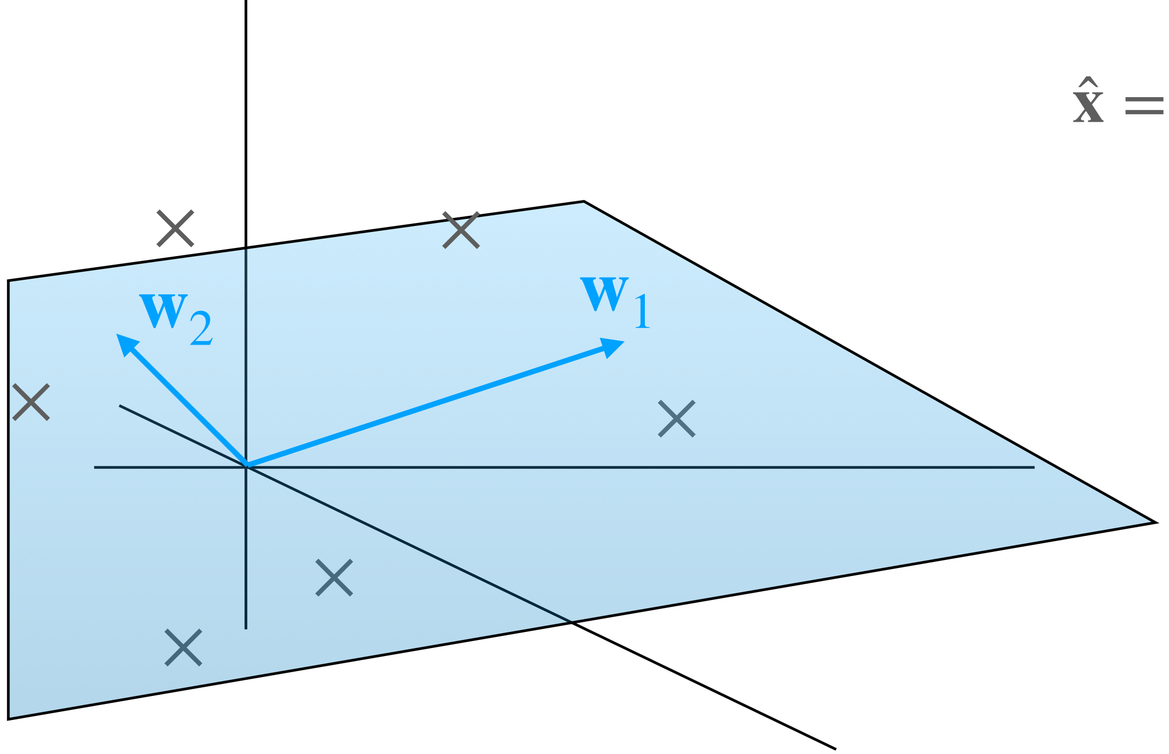


Arrows = embeddings $\mathbf{v}_j$ (here, in $\mathbb{R}^2$)

- Embed “cranberry melon”

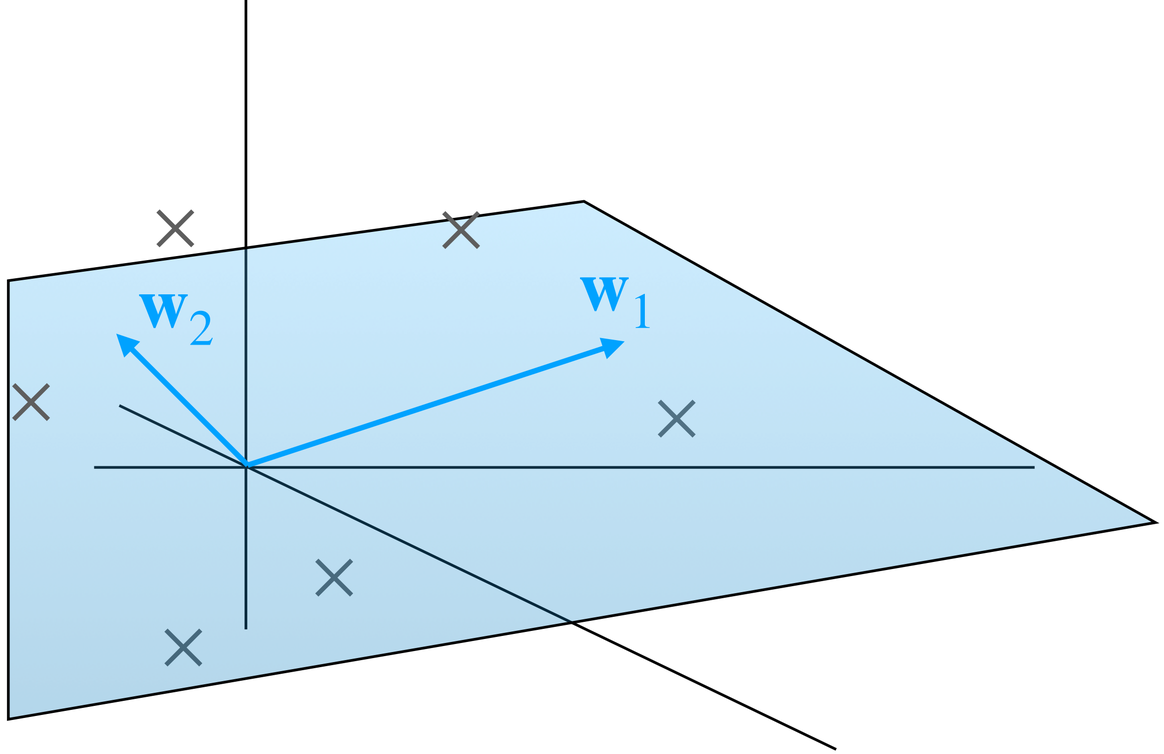
Interpreting autoencoder weights and activations

$$\hat{\mathbf{x}} = W\mathbf{u} = WV\mathbf{x}$$



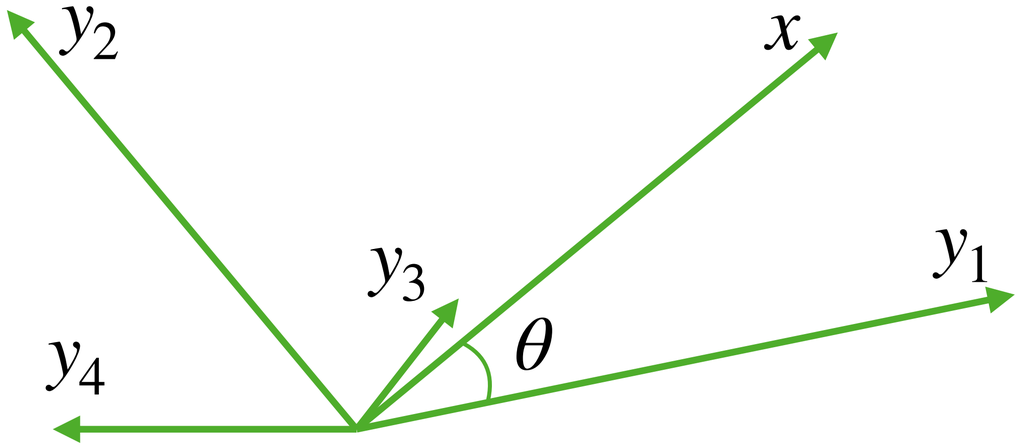
- Columns $\mathbf{w}_j$ of W: a basis for reconstructed examples $\hat{\mathbf{x}}^{(i)}$ (predicted bags of tokens)
- Hidden vectors $\mathbf{u}^{(i)}$: coefficients for the basis vectors in the representation of $\hat{\mathbf{x}}^{(i)}$ — this is our embedding
- Columns of V: what will $\mathbf{u}^{(i)}$ be if input $\mathbf{x}^{(i)}$ is a unit vector — i.e., column for $\mathbf{e}_{\text{dog}}$ is embedding of dog

Interpreting autoencoder weights and activations



- Given optimal W, the matching V is unique
  - can find it by SGD or linear algebra: it's $V = W^\dagger$
  - best V finds closest $\hat{\mathbf{x}}$ to a given $\mathbf{x}$ within the subspace we can represent — i.e., it projects onto basis $\{\mathbf{w}_j\}$

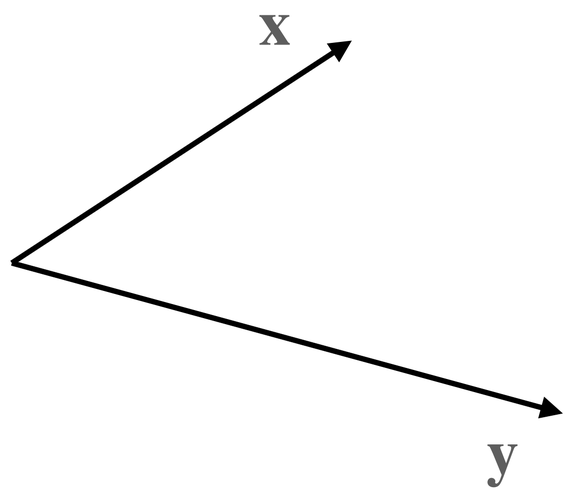
Reminder: dot product



- Dot product $\mathbf{x} \cdot \mathbf{y}$ depends on both *norm* and *direction* of input vectors $\mathbf{x}$ and $\mathbf{y}$
  - $\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos\theta$
- Large dot product means
  - $\mathbf{x}$ and $\mathbf{y}$ point in in approximately the *same direction*
  - $\mathbf{x}$ and $\mathbf{y}$ both have *large norm*

Reminder: vector projection

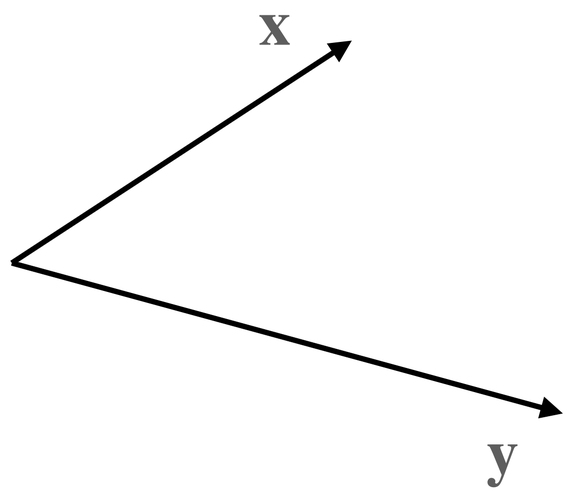
- Projection of $\mathbf{x}$ onto $\mathbf{y}$: best multiple $\hat{\mathbf{x}} = \lambda \mathbf{y}$ that minimizes distance (or squared distance) $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Length of projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\|\hat{\mathbf{x}}\| = \mathbf{y}^\top \mathbf{x}$ if $\|\mathbf{y}\| = 1$



- Projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\hat{\mathbf{x}} = \mathbf{y}(\mathbf{y}^\top \mathbf{x})$ if $\|\mathbf{y}\| = 1$

Reminder: vector projection

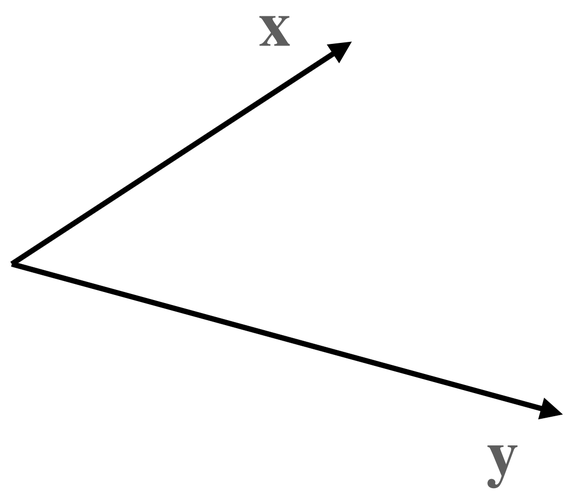
- Projection of $\mathbf{x}$ onto $\mathbf{y}$: best multiple $\hat{\mathbf{x}} = \lambda \mathbf{y}$ that minimizes distance (or squared distance) $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Length of projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\|\hat{\mathbf{x}}\| = \dfrac{\mathbf{y}^\top \mathbf{x}}{\|\mathbf{y}\|}$ if $\|\mathbf{y}\| = 1$ o/w



- Projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\hat{\mathbf{x}} = \mathbf{y}(\mathbf{y}^\top \mathbf{x})$ if $\|\mathbf{y}\| = 1$

Reminder: vector projection

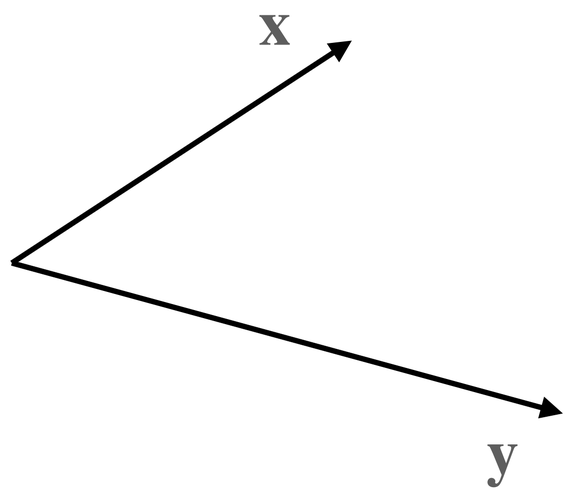
- Projection of $\mathbf{x}$ onto $\mathbf{y}$: best multiple $\hat{\mathbf{x}} = \lambda \mathbf{y}$ that minimizes distance (or squared distance) $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Length of projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\|\hat{\mathbf{x}}\| = \dfrac{\mathbf{y}^\top \mathbf{x}}{\|\mathbf{y}\|}$ if $\|\mathbf{y}\| = 1$ o/w



- Projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\hat{\mathbf{x}} = \dfrac{\mathbf{y}(\mathbf{y}^\top \mathbf{x})}{\|\mathbf{y}\|^2}$ if $\|\mathbf{y}\| = 1$ o/w

Reminder: vector projection

- Projection of $\mathbf{x}$ onto $\mathbf{y}$: best multiple $\hat{\mathbf{x}} = \lambda \mathbf{y}$ that minimizes distance (or squared distance) $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Length of projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\|\hat{\mathbf{x}}\| = \dfrac{\mathbf{y}^\top \mathbf{x}}{\|\mathbf{y}\|}$ if $\|\mathbf{y}\| = 1$ o/w

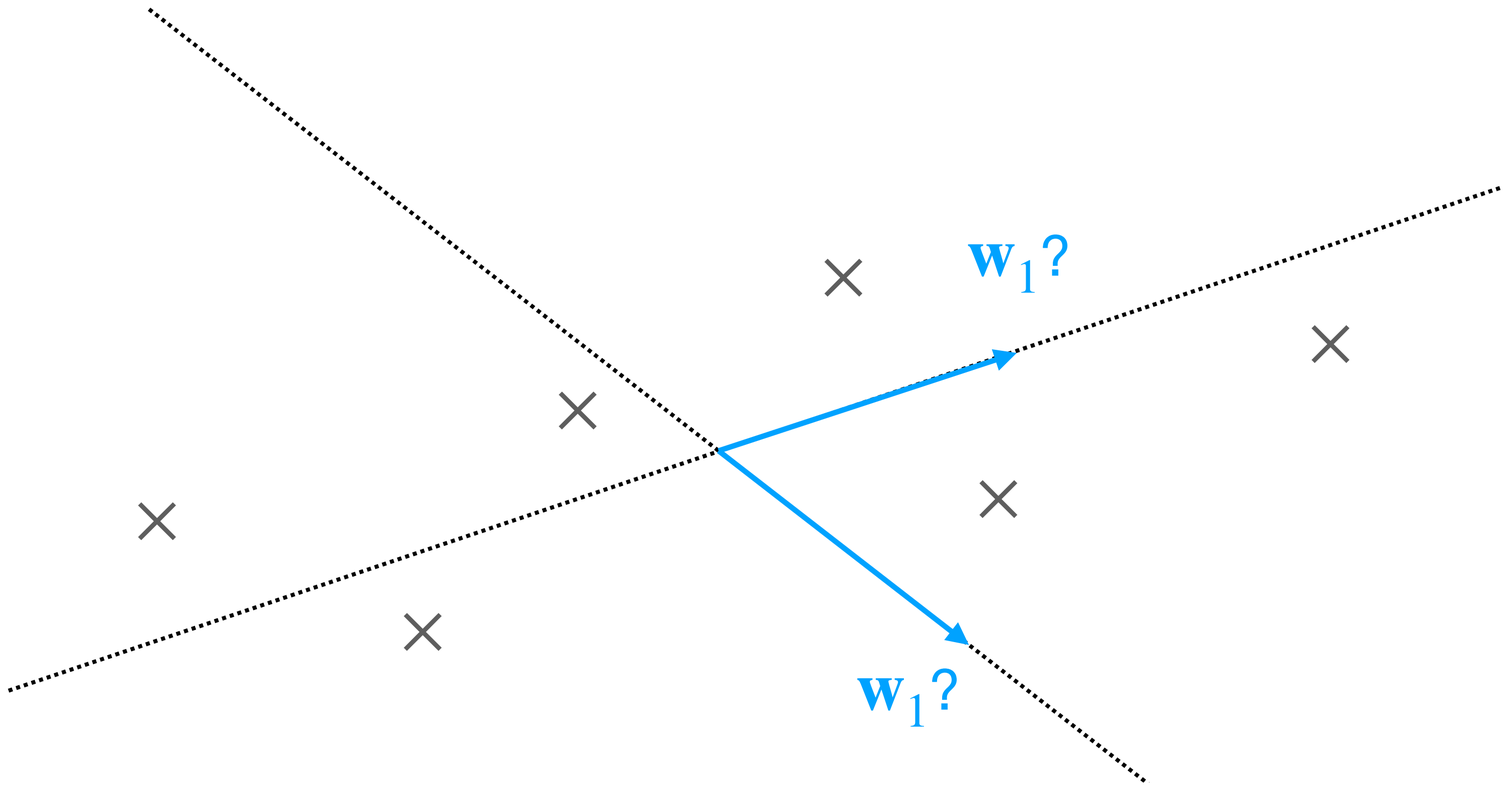


- Projection of $\mathbf{x}$ onto $\mathbf{y}$:
  - $\hat{\mathbf{x}} = \dfrac{\mathbf{y}(\mathbf{y}^\top \mathbf{x})}{\|\mathbf{y}\|^2}$ if $\|\mathbf{y}\| = 1$ o/w

λ

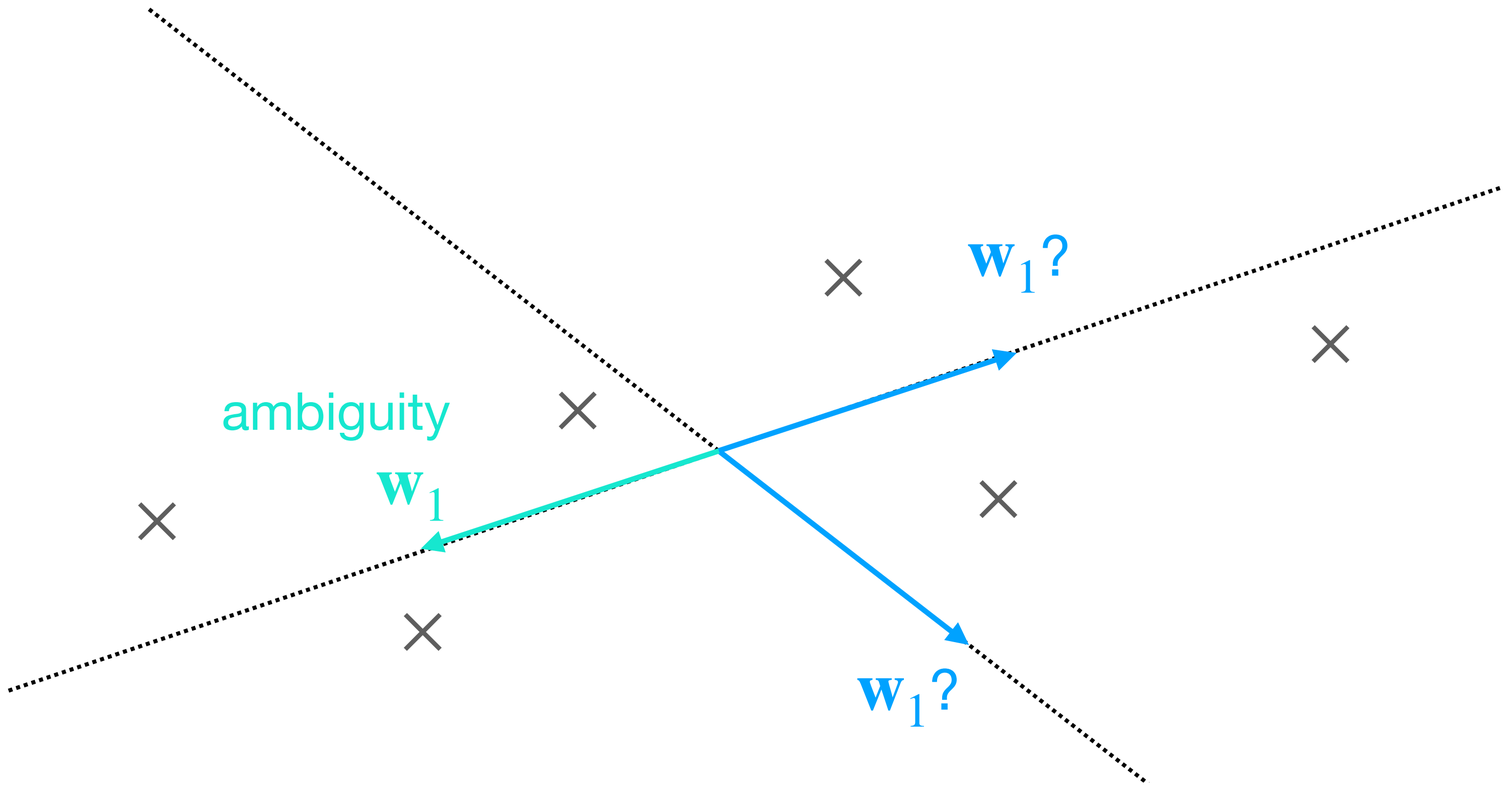
1-d autoencoder

First col of $W = \mathbf{w}_1$



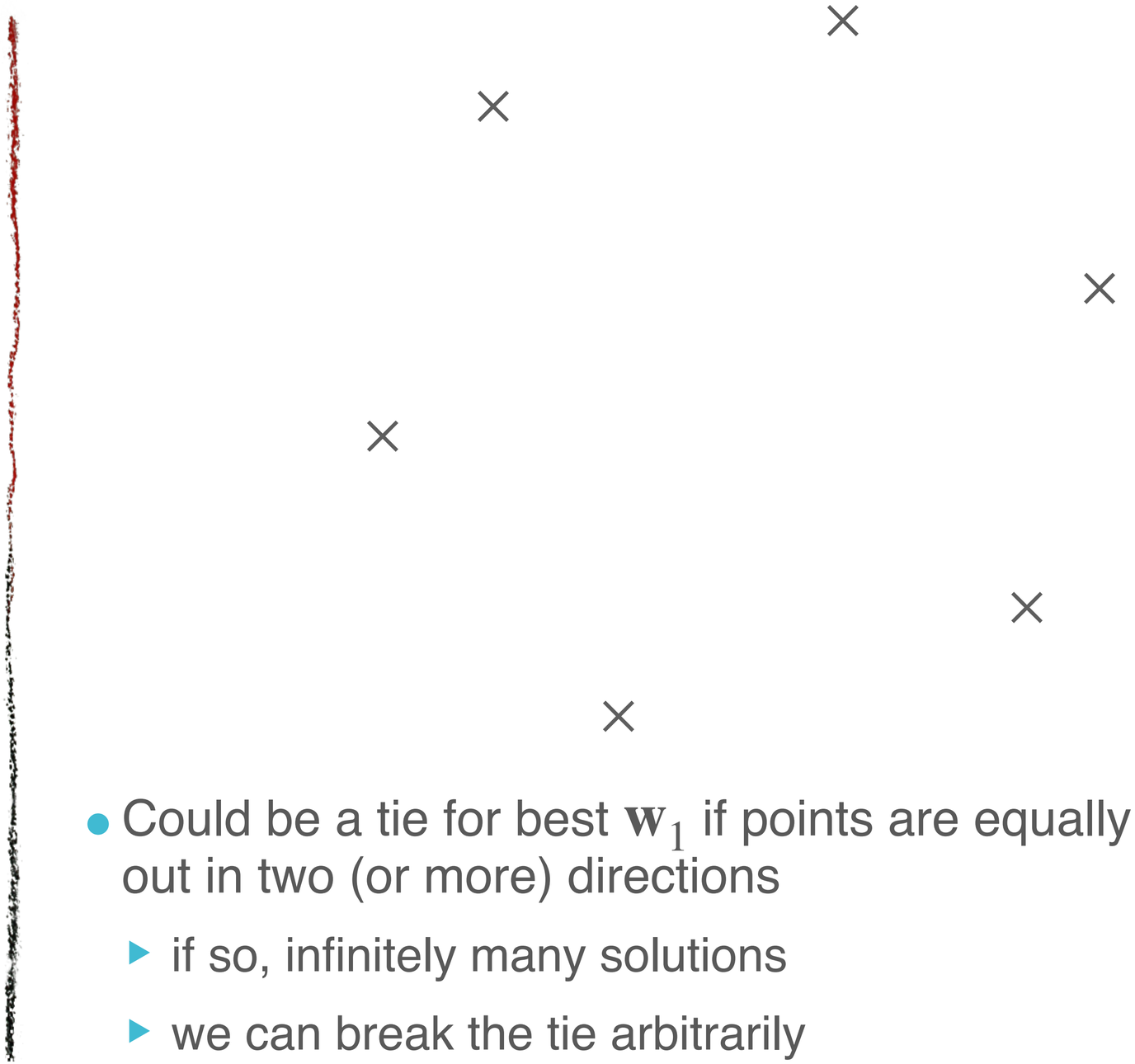
- Visualize solution to 1-d autoencoder
  - find $\mathbf{w}_1$ by min sum-squared error
  - intuitively: if datapoints are spread out, $\mathbf{w}_1$ should point along the long direction

1-d autoencoder



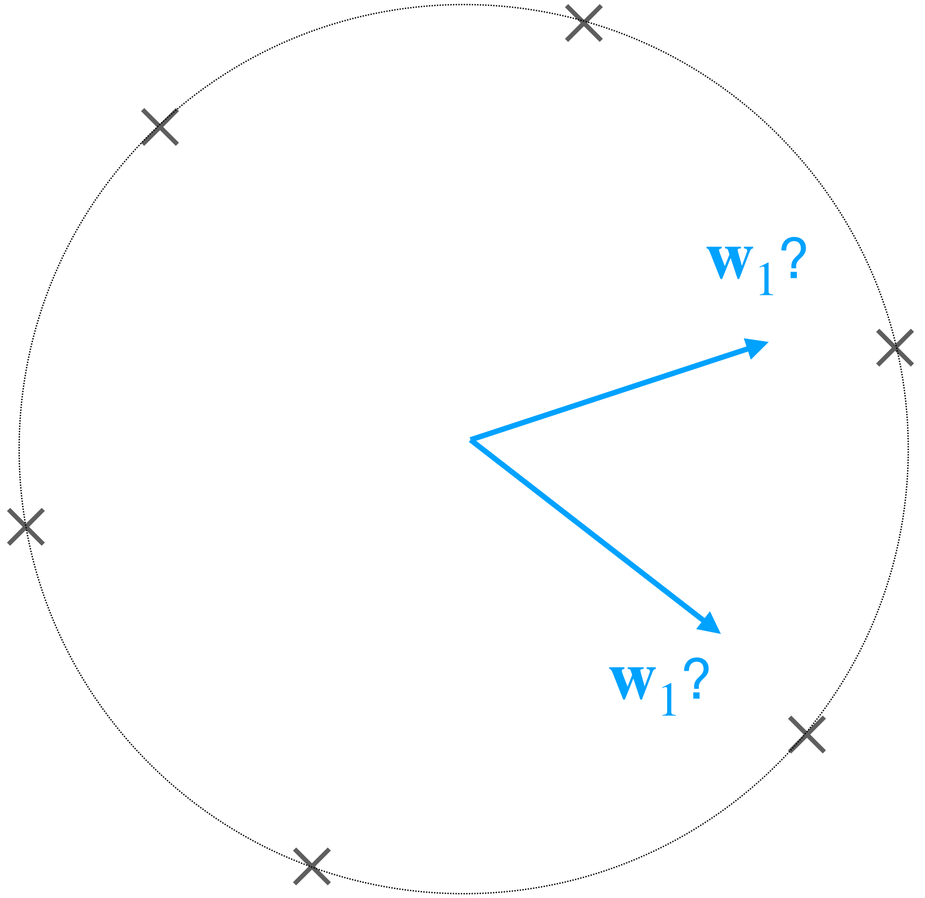
- Visualize solution to 1-d autoencoder
  - find $\mathbf{w}_1$ by min sum-squared error
  - intuitively: if datapoints are spread out, $\mathbf{w}_1$ should point along the long direction

Ambiguity



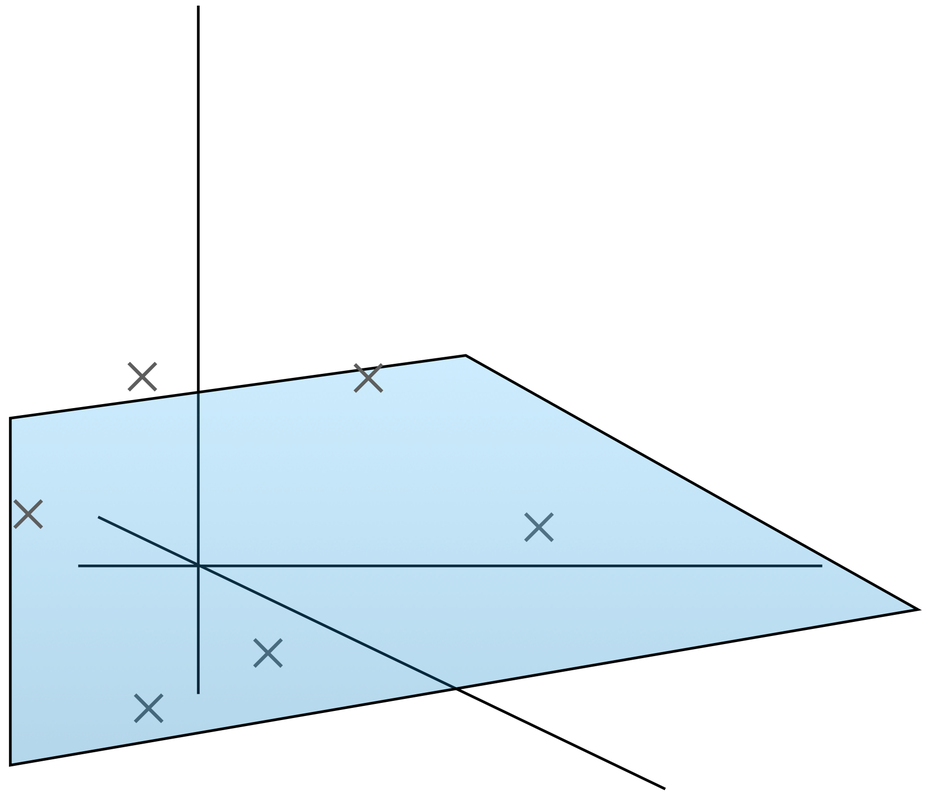
- Could be a tie for best $\mathbf{w}_1$ if points are equally spread out in two (or more) directions
  - if so, infinitely many solutions
  - we can break the tie arbitrarily

Ambiguity



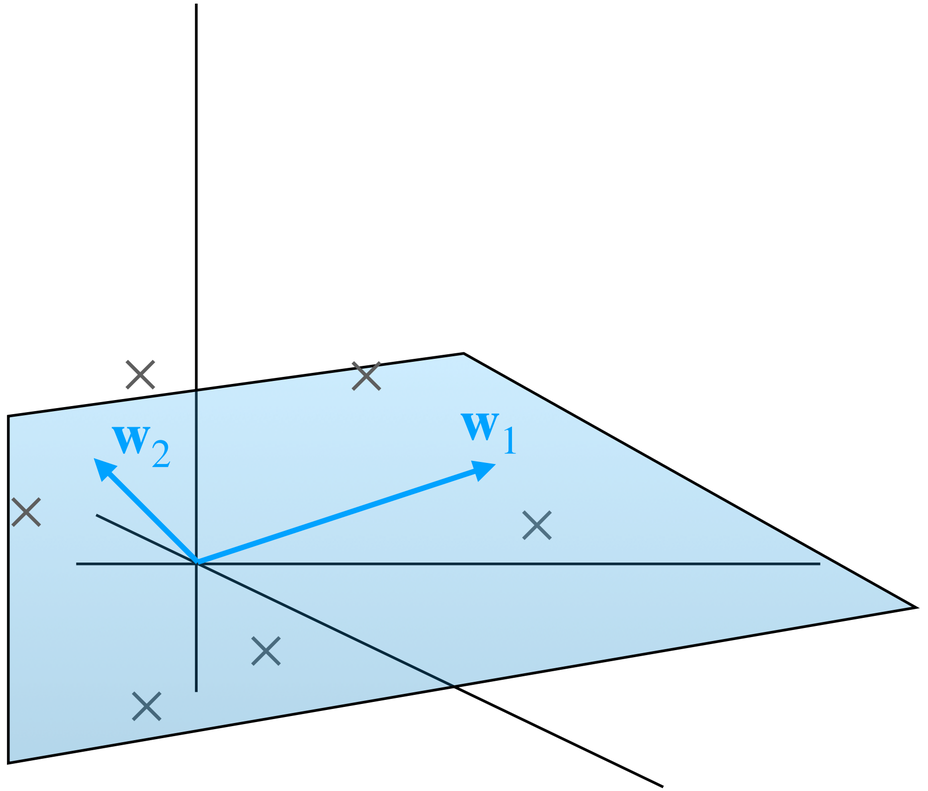
- Could be a tie for best $\mathbf{w}_1$ if points are equally spread out in two (or more) directions
  - if so, infinitely many solutions
  - we can break the tie arbitrarily

Best plane



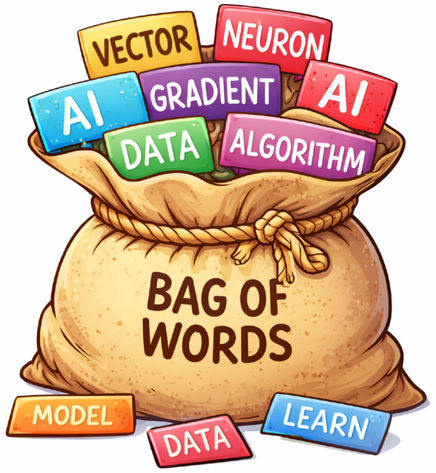
- What about $k = 2$?
  - project onto best 2D plane (min squared error)

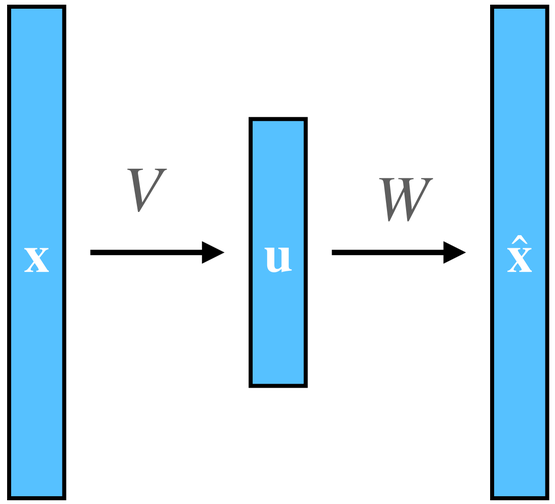
Ambiguity: basis



- Can choose any basis for the plane — reconstruction (and reconstruction error) will be the same
- Can (mostly) resolve this ambiguity by requiring $V = W^\top$
  - embeddings = columns of V = rows of W
  - basis = rows of V = columns of W

LSA

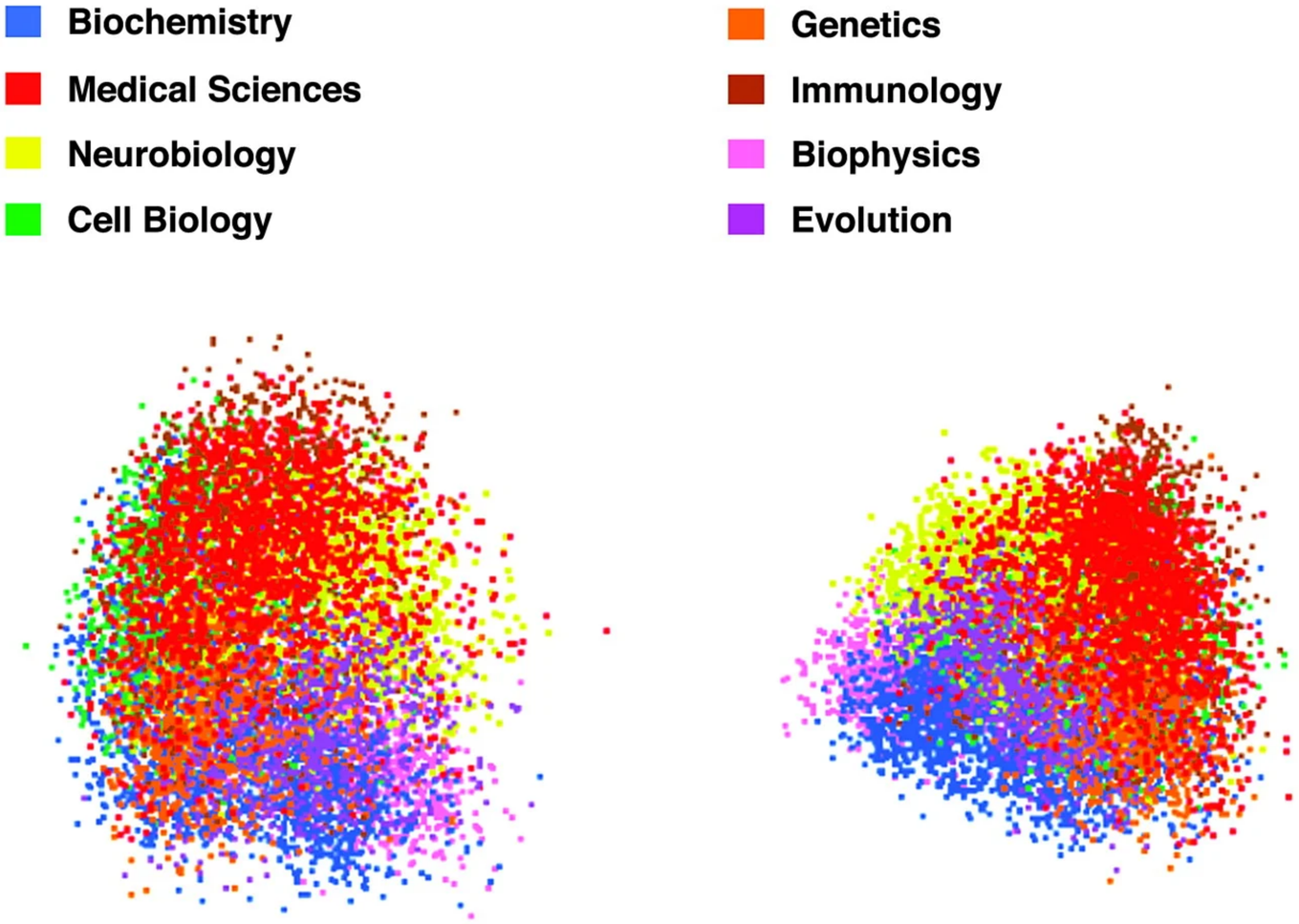




- OK, we got vector-space embeddings by fitting a linear autoencoder to a dataset of bags of tokens
  - maybe each bag is a document, or maybe it's a shorter window of a few consecutive tokens
- This model is called ***latent semantic analysis***
  - details: stopwords, IDF weighting, no mean subtraction

LSA example

Landauer et al., PNAS 2004 https://www.pnas.org/doi/10.1073/pnas.0400341101



- Ex: LSA of articles from PNAS shows that article embeddings reveal broad topic

Poll

https://forms.gle/qD6YjXGtFciCpTr57

- When we calculate gradients for an RNN via backprop through time, what are the differences from backprop in a plain MLP?
  A. there are often multiple prediction heads (overall loss is the sum of losses for individual predictions)
  B. there is typically parameter sharing (weight matrices are the same at each time step)
  C. we don't have to worry about vanishing gradients
  D. Both A and B
  E. Both B and C

Proxy task: masked token prediction

masked token prediction

The quick brown fox [???] over the lazy dog

- Masked token prediction:
  - pick a token index i
  - *input:* a window of tokens near index i but ***not*** token i: $j \in \{i-k, \ldots, i+k\} \setminus \{i\}$ (the ***context***)
  - could also mask more than 1 token in a context window
  - *target:* the masked token i

Word2vec

masked token prediction

The quick brown fox [???] over the lazy dog

- Early successful *nonlinear* embedding learner
- Train embedding matrix V by masked token prediction
  - reconstruction weights are tied, just as in LSA
- Two layer network:
  - first layer: embed and sum tokens in context

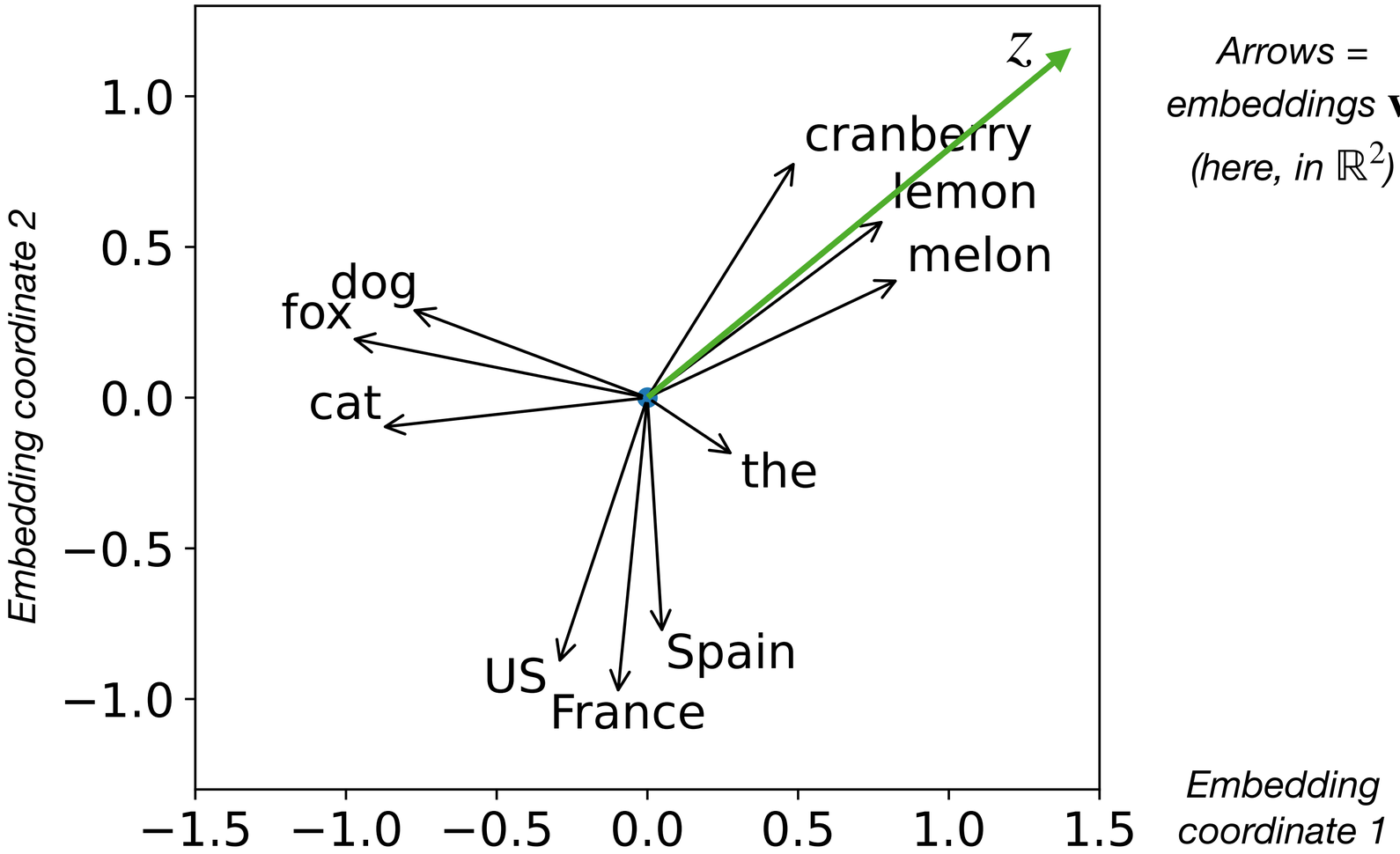
$$\mathbf{z} = \sum_{j=i-k,\, j\neq 0}^{i+k} V\mathbf{x}_j$$

  - second layer: softmax over vocabulary

$$P(\mathbf{y} \mid \text{input}) = \frac{\exp(\mathbf{z} \cdot V\mathbf{y})}{\sum_{\mathbf{y}'} \exp(\mathbf{z} \cdot V\mathbf{y}')}$$

  - note: approximations to make it fast

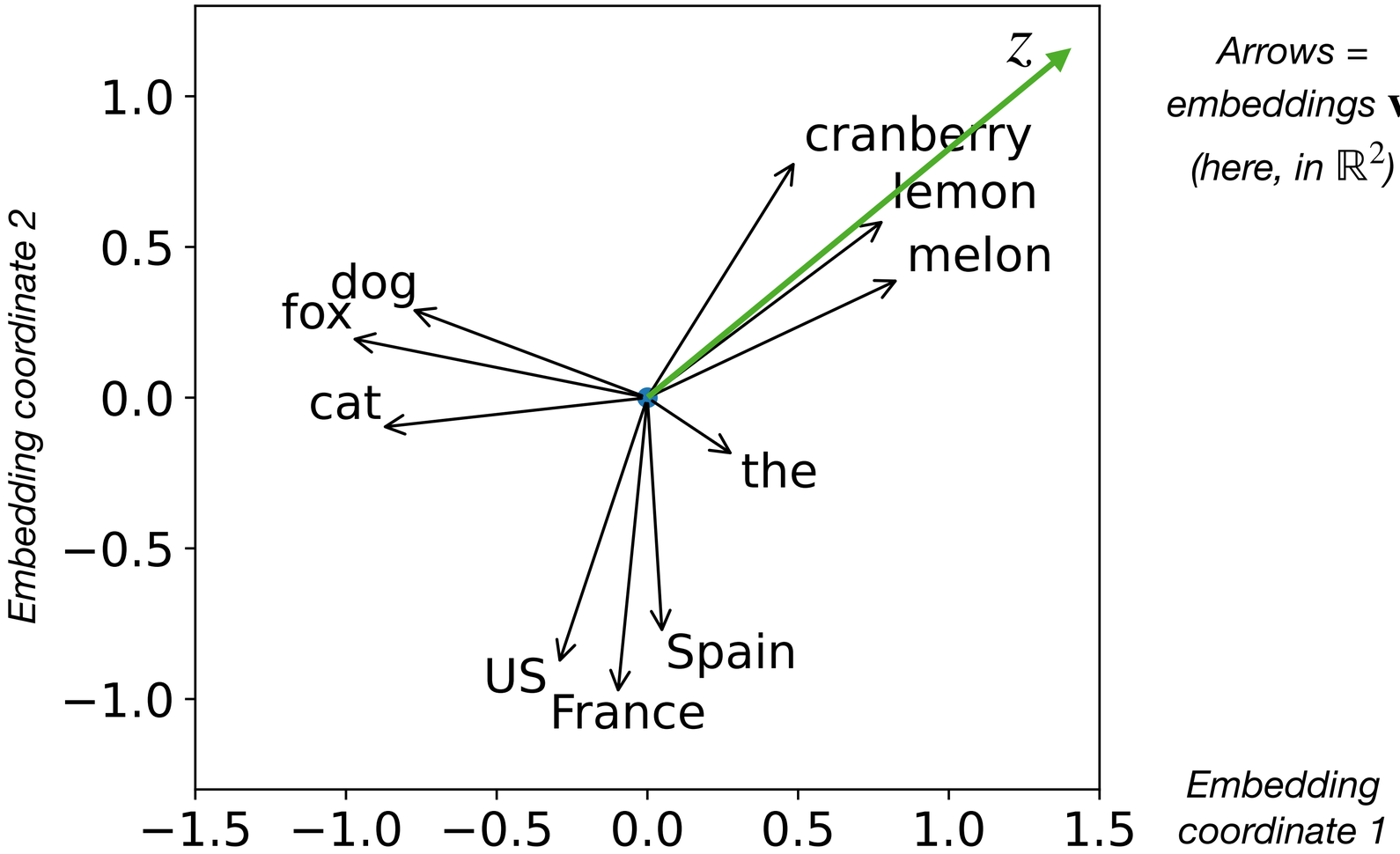
First layer



prediction task: … cranberry [???] melon …

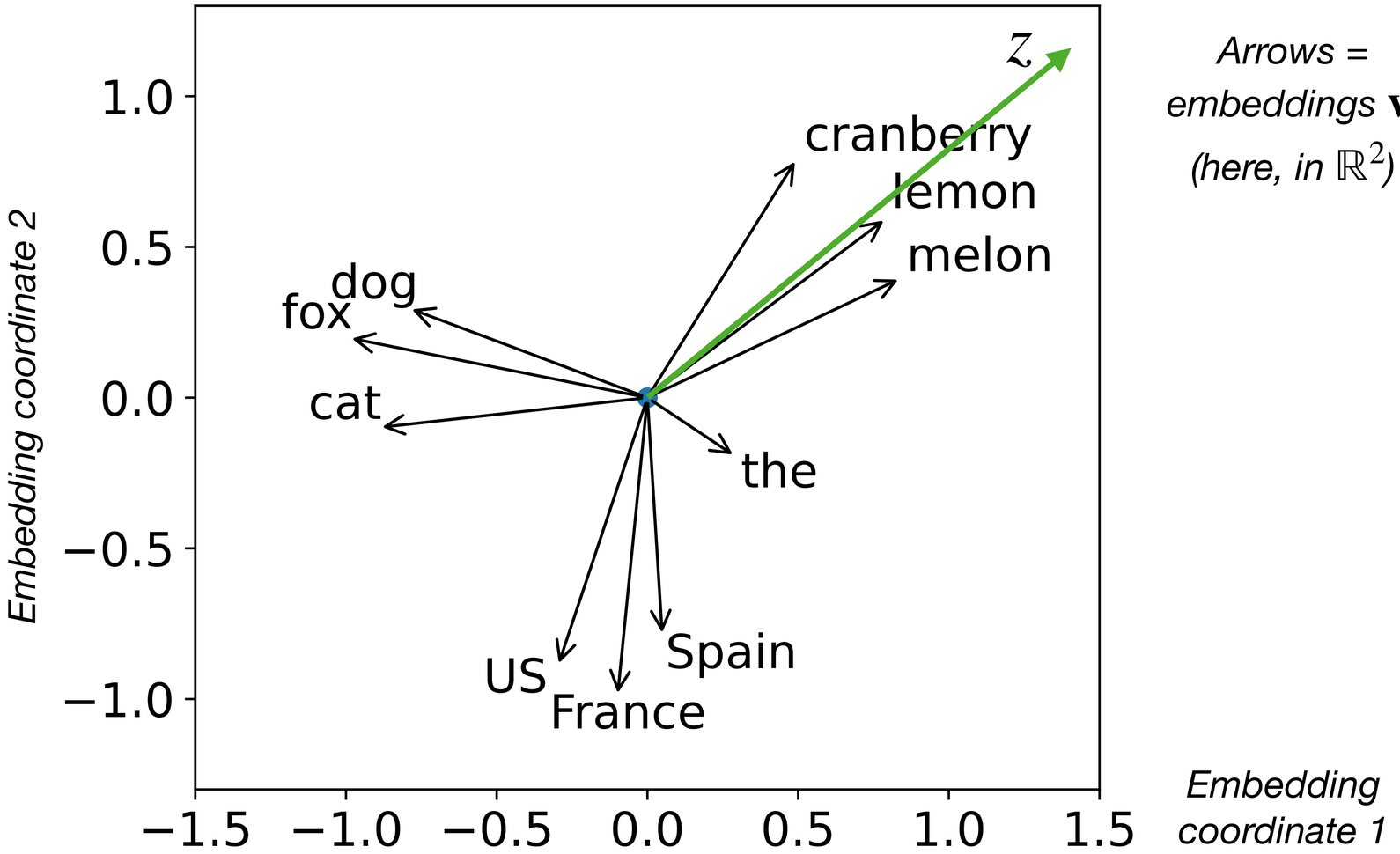
- $\mathbf{z}$ = sum of context tokens (i.e., context as a BoW)

Second layer



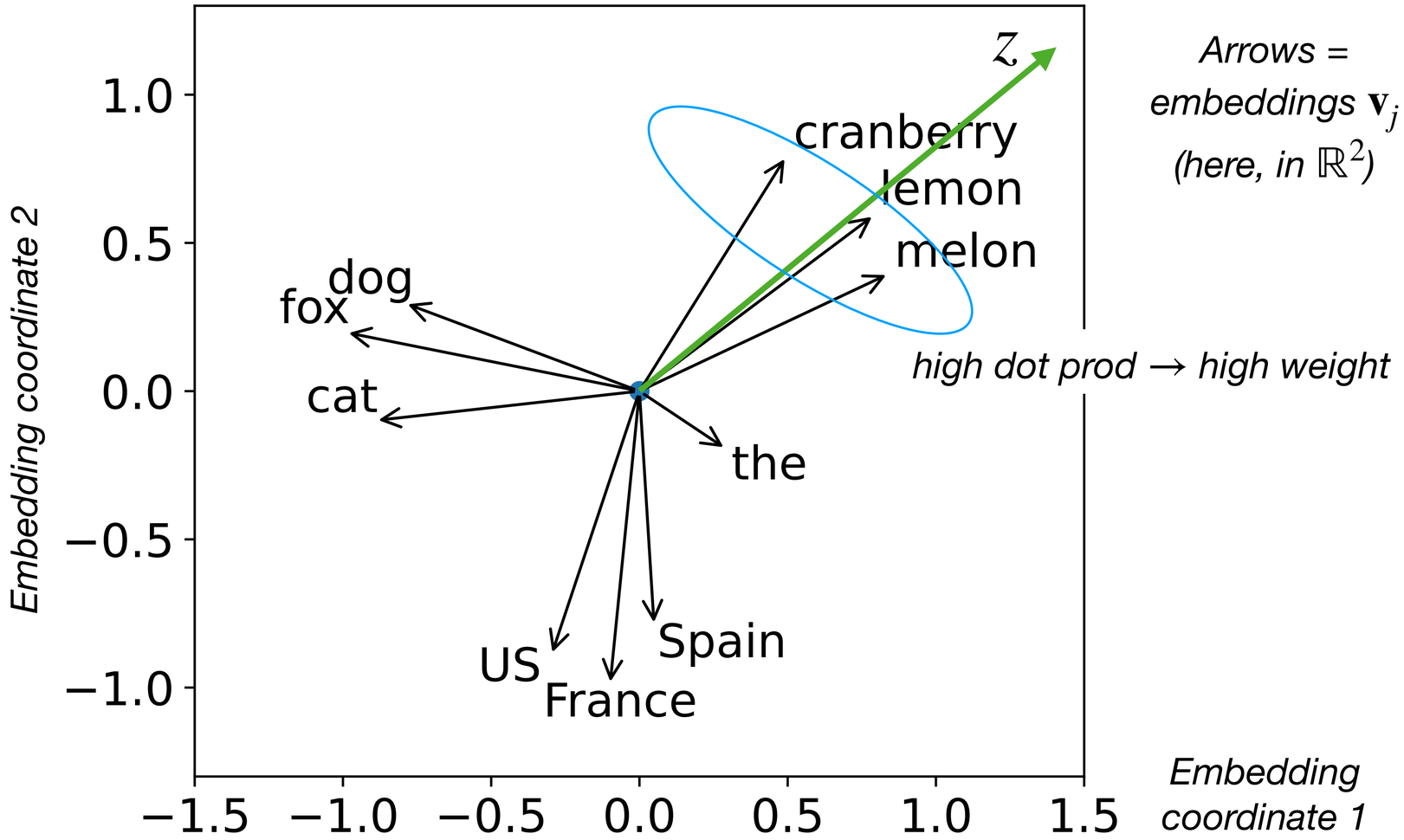
- Input to second layer = a vector $\mathbf{z}$, summary of context
- Goal of second layer: search through a set of vectors (vocabulary embeddings) to find similar ones to context

Second layer



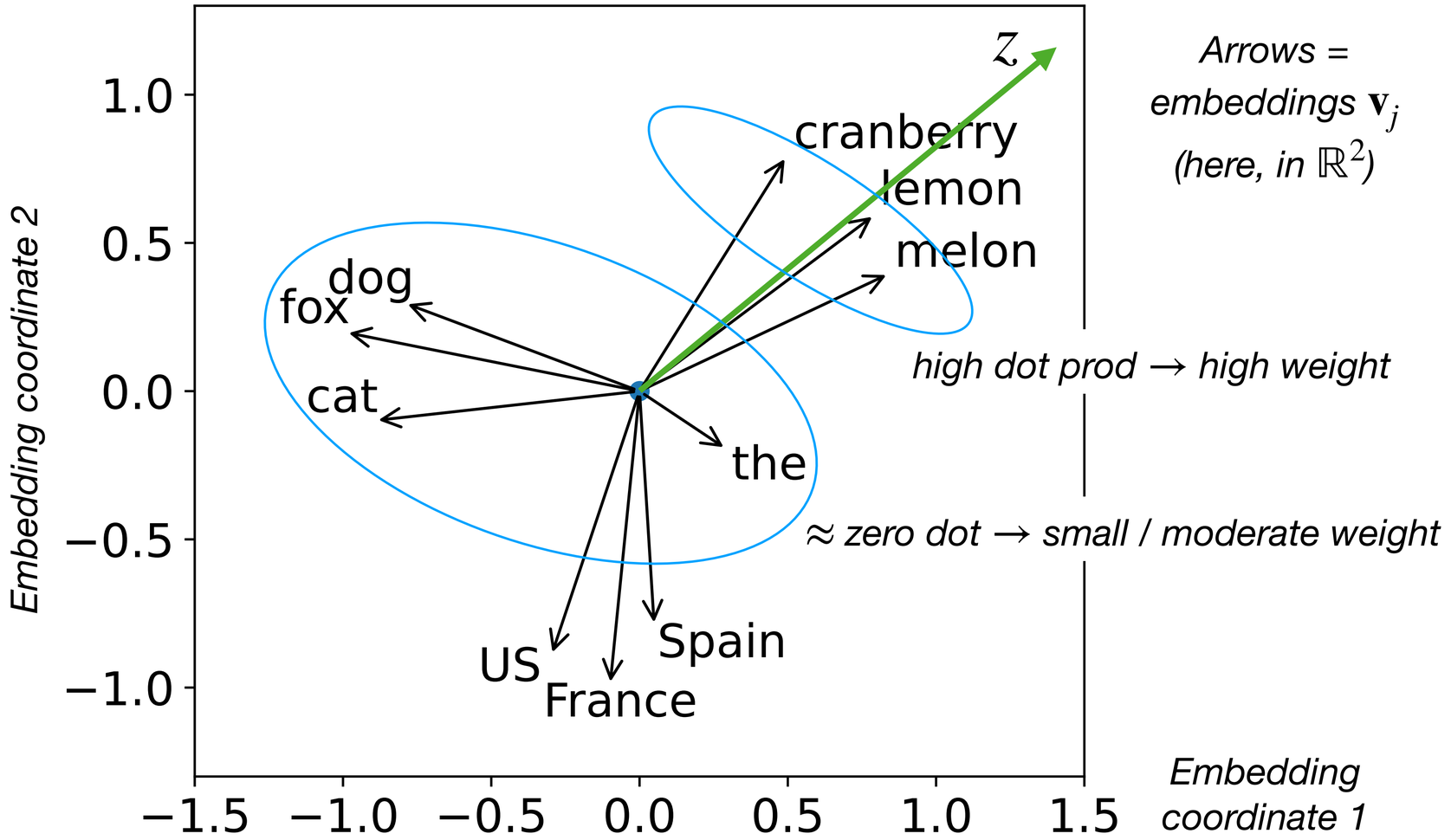
- Dot product $\mathbf{v}_j \cdot \mathbf{z}$ is highest for token embeddings that are both *high norm* and *similar direction* to $\mathbf{z}$
- The $\exp$ in softmax further emphasizes highest values

Second layer



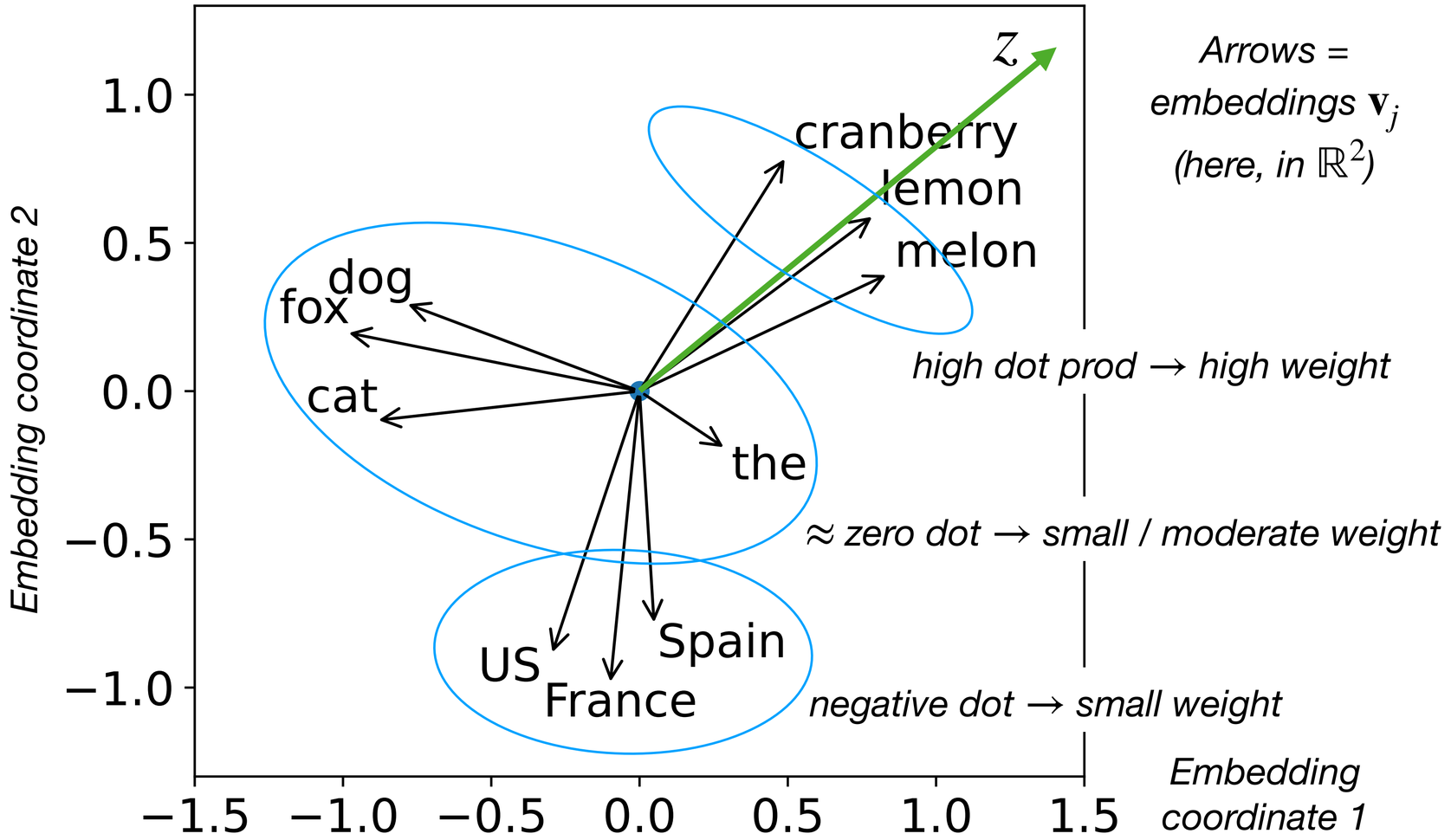
- Dot product $\mathbf{v}_j \cdot \mathbf{z}$ is highest for token embeddings that are both *high norm* and *similar direction* to $\mathbf{z}$
- The $\exp$ in softmax further emphasizes highest values

Second layer



- Dot product $\mathbf{v}_j \cdot \mathbf{z}$ is highest for token embeddings that are both *high norm* and *similar direction* to $\mathbf{z}$
- The $\exp$ in softmax further emphasizes highest values

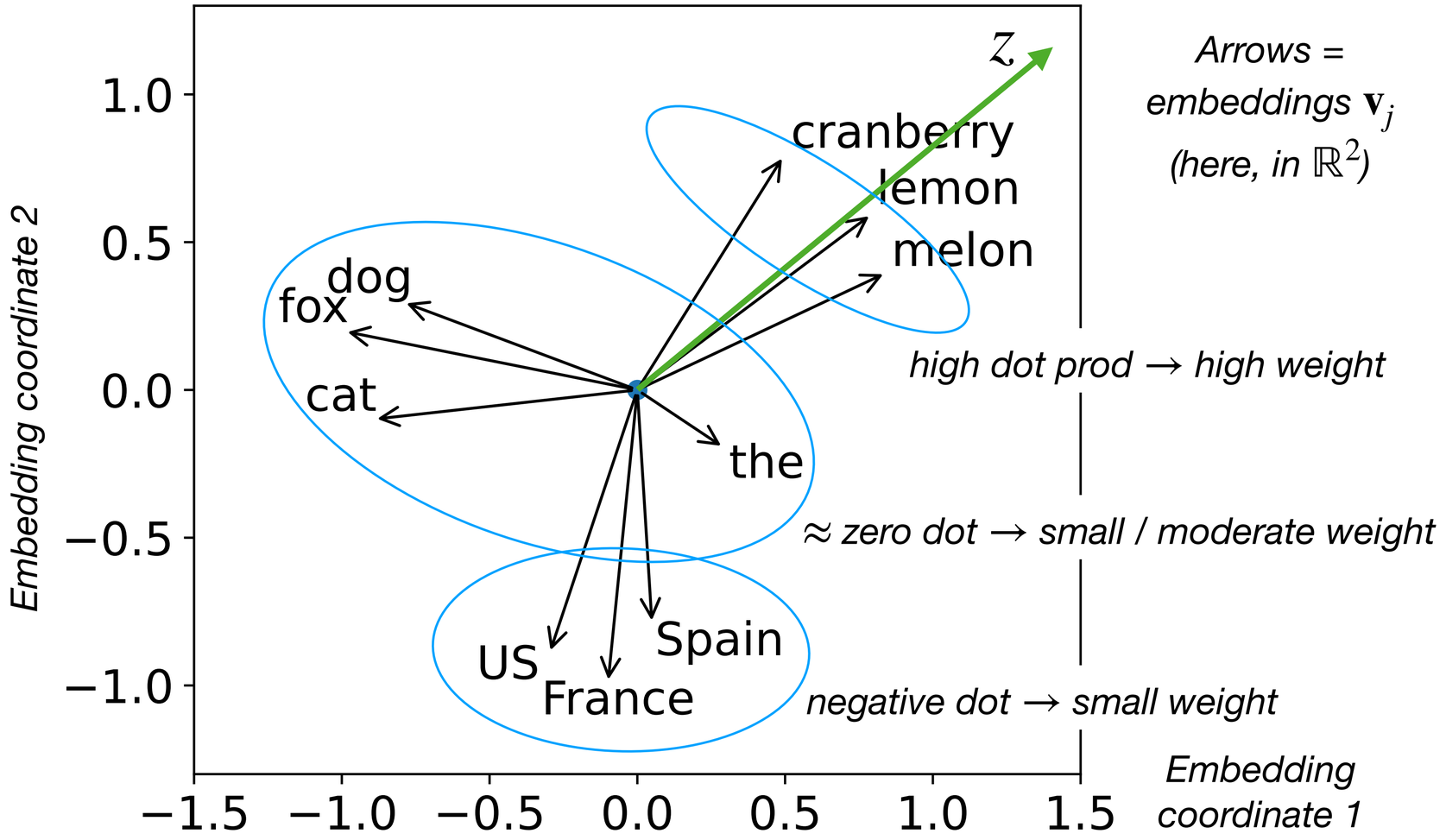
Second layer



- Dot product $\mathbf{v}_j \cdot \mathbf{z}$ is highest for token embeddings that are both *high norm* and *similar direction* to $\mathbf{z}$
- The $\exp$ in softmax further emphasizes highest values

Second layer

*This use of softmax (to find vocabulary words that have high dot product w/ context) is strongly related to **attention** (coming up soon)*

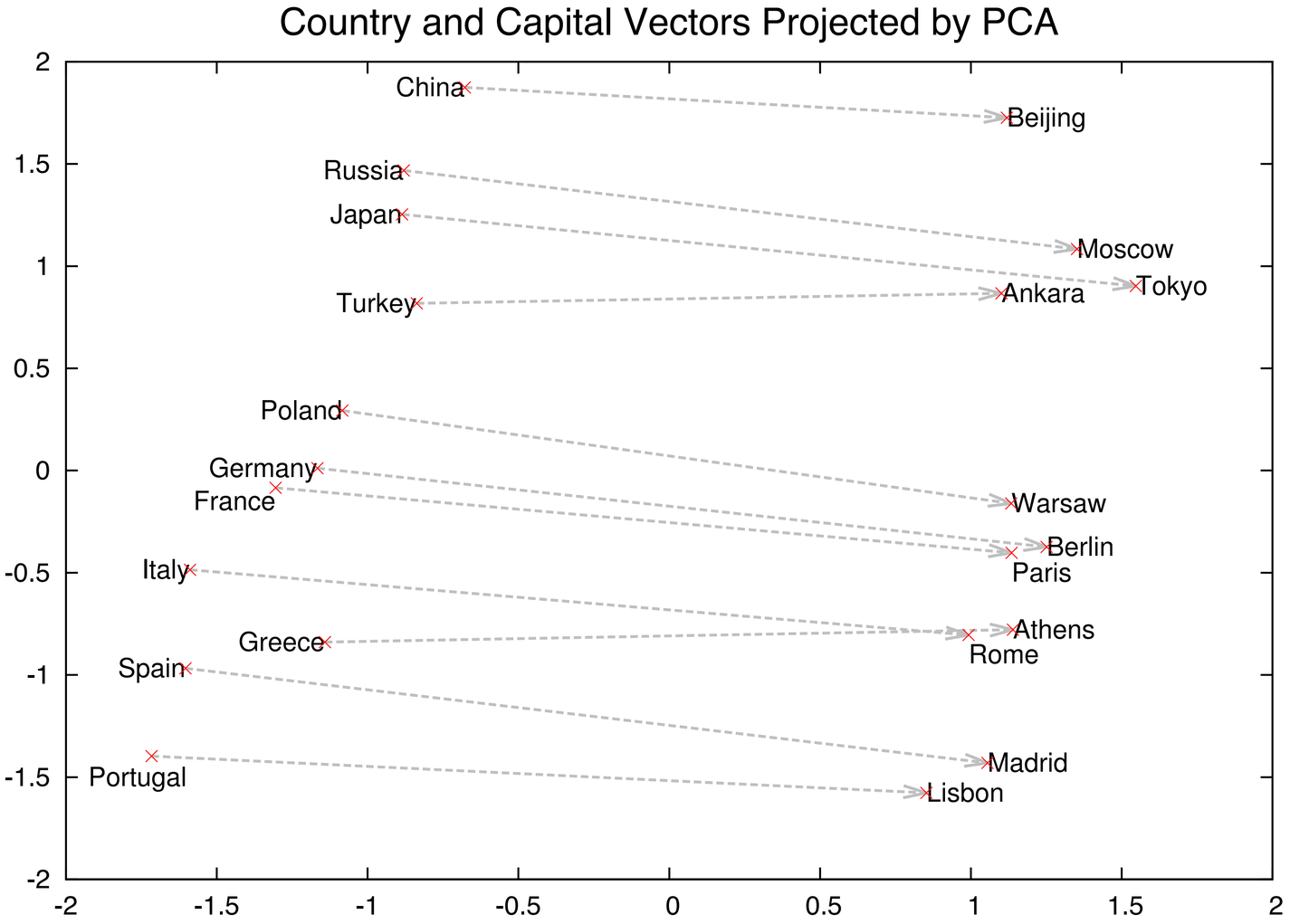


- Dot product $\mathbf{v}_j \cdot \mathbf{z}$ is highest for token embeddings that are both *high norm* and *similar direction* to $\mathbf{z}$
- The $\exp$ in softmax further emphasizes highest values

SGD step

- Each optimization step will try to
  - shift embedding of (words in the) query to better retrieve correct masked word (put more attention weight on it)
  - shift the correct masked word's embedding to better match query (have higher dot product with it)
  - shift all other words' embeddings to have lower dot products with the query

Some learned embeddings



- A 2d projection of some learned vector-space embeddings from word2vec

Mikolov et al., https://arxiv.org/abs/1310.4546

Learning goals: deep nets

- What is a deep neural network?
- What are depth, width, and work, and why do they matter?
- What are common elements/blocks of deep nets? For each one, why might you include it in a network?
- How do you calculate the loss and its gradient, given a training example (x, y)?
- How can you set up and use autodiff in a library like PyTorch?
- What are vanishing or exploding gradients? Why are they a problem? How can we mitigate this problem?

Learning goals: CNNs

- What is a tensor, and what are common tensor operations?
- What are Einstein summation and tensor contraction?
- What is a (continuous time or space) signal? How does it relate to a (discrete time or space) sampled vector, matrix, or tensor?
- What is a convolution? What tasks can it accomplish?
- What variations exist of the convolution operation?
- What are equivariance and invariance, and how do we implement them in a CNN?
- What are multi-head convolutions, layerwise convolutions, channels, pooling, vec/unvec?

Learning goals: RNNs

- What different kinds of sequence prediction tasks might we encounter and how can we present them to an RNN?
- What does a basic RNN block look like, and how do we use it to make predictions?
- What is a vector-space embedding, and what is it used for?
- What is long-term memory in an RNN, how does it relate to gating, and what problem does it solve?
- What is backprop through time? How is it similar or different to backprop in other deep nets?
- How can we train an RNN language model? How do we use it to predict?