Sampling from a Language Model

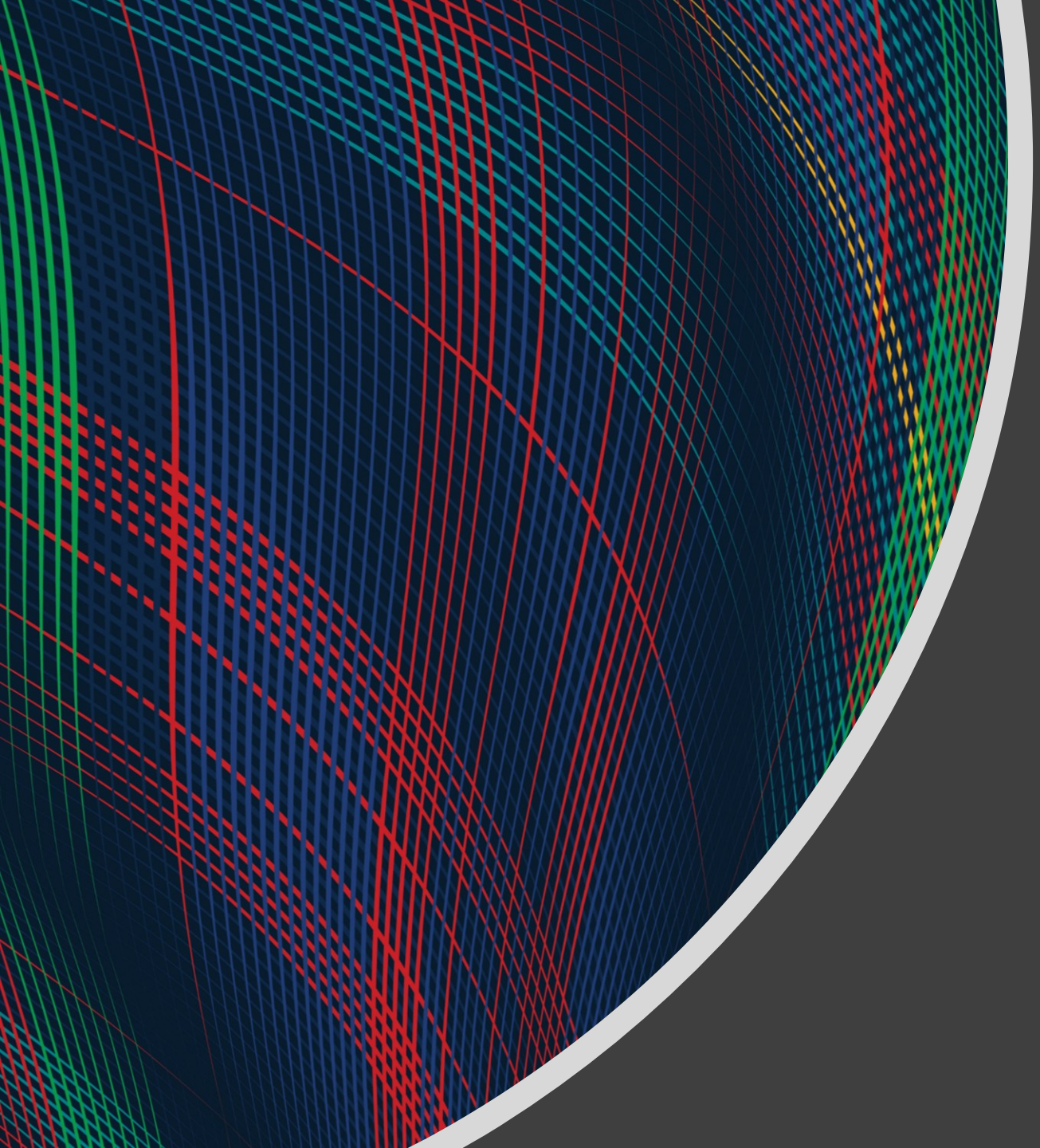
Temperature sampling

Allows us to adjust the level randomness during sampling via hyperparameter T (Temperature):

- T= 1.0: Use the original probabilities (default)
- T > 1.0: "Softer" distribution → more random, diverse outputs
- T < 1.0: "Sharper" distribution → more predictable, focused outputs
- T → 0: Effectively becomes greedy (always pick the most likely token)

Adjusted probability:

$$P_{temperature}(w_i) = \frac{e^{\log P(w_i)/T}}{\sum_j e^{\log P(w_i)/T}}$$



07-280
AI & ML I

NLP: Word Embeddings Attention

Instructors:
Pat Virtue & Nihar Shah

Natural Language Processing (NLP)

Last time

- Language Models
  - N-gram LMs
  - Training Data
  - Sampling (Decoding) (Generation) (Inference)

Today

- Feature Engineering for NLP
- Feature Learning
  - Word Embeddings
  - Image and Audio Embeddings

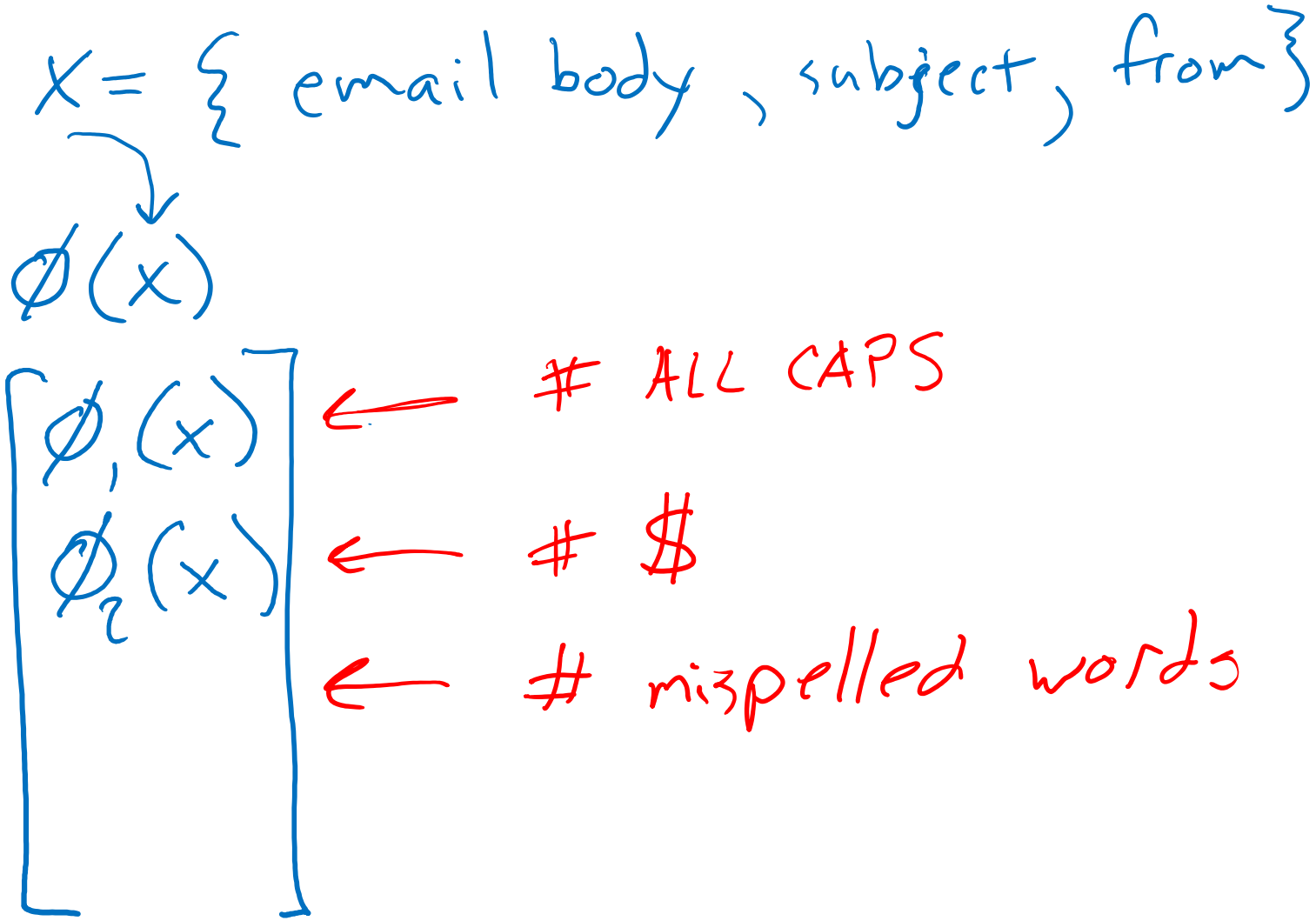
Transformer LM

- Increasing context size
  - Pos encoding
  - Attention

Feature Engineering for Text

Text Features

SPAM Classification



Text Features

Predicting Rating from Written Movie Review

$x = \{\text{text}\}$

$\phi(x) \in \mathbb{R}^M$

$$\begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_M(x) \end{bmatrix} \leftarrow \#\ \text{times fantastic}$$

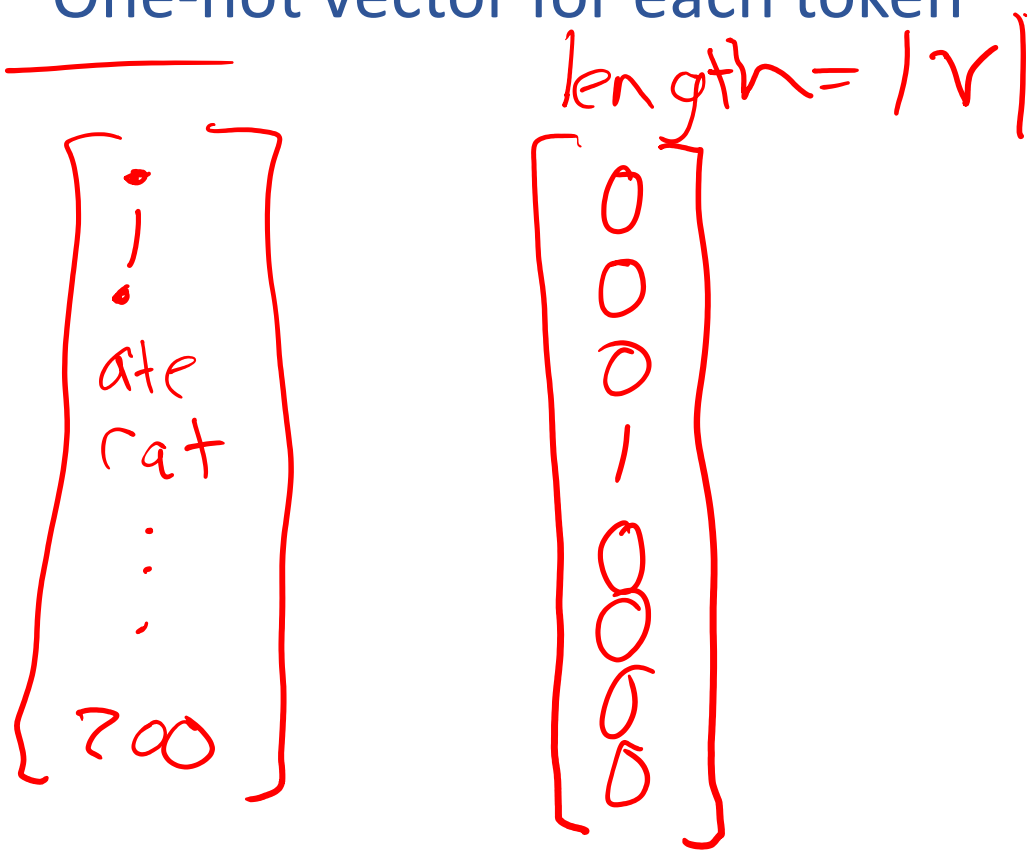
$$\hat{y} = h(\vec{x})$$

$$= g(\vec{\theta}^T \vec{x})$$

$$= g(\vec{\theta}^T \phi(x))$$

Text Features

One-hot vector for each token

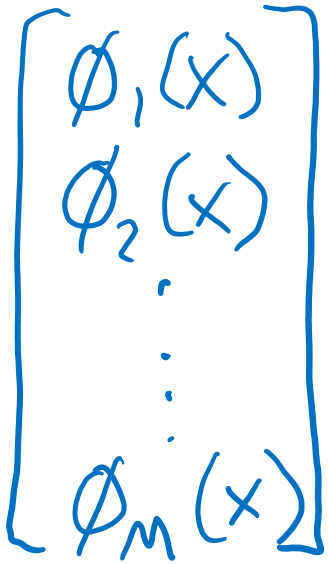


Text Features

Bag of words: Vector of length the size of the vocabulary

Two options

- Word occurrence: Binary: does the word exist (at least once) or not
- Word histogram: Integer: count of how many times the word appears

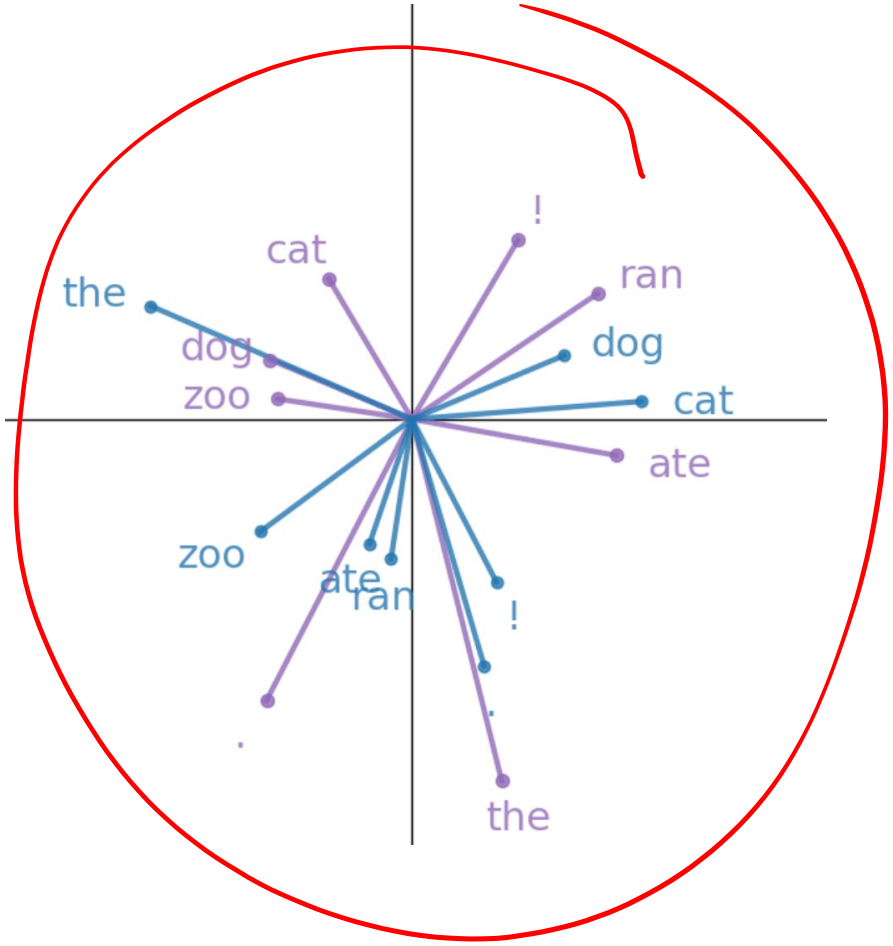


Word Embedding LMs

Word Embedding Language Models

Vector representation of vocab tokens

- Set of vectors for both previous and next token



Sampling next token

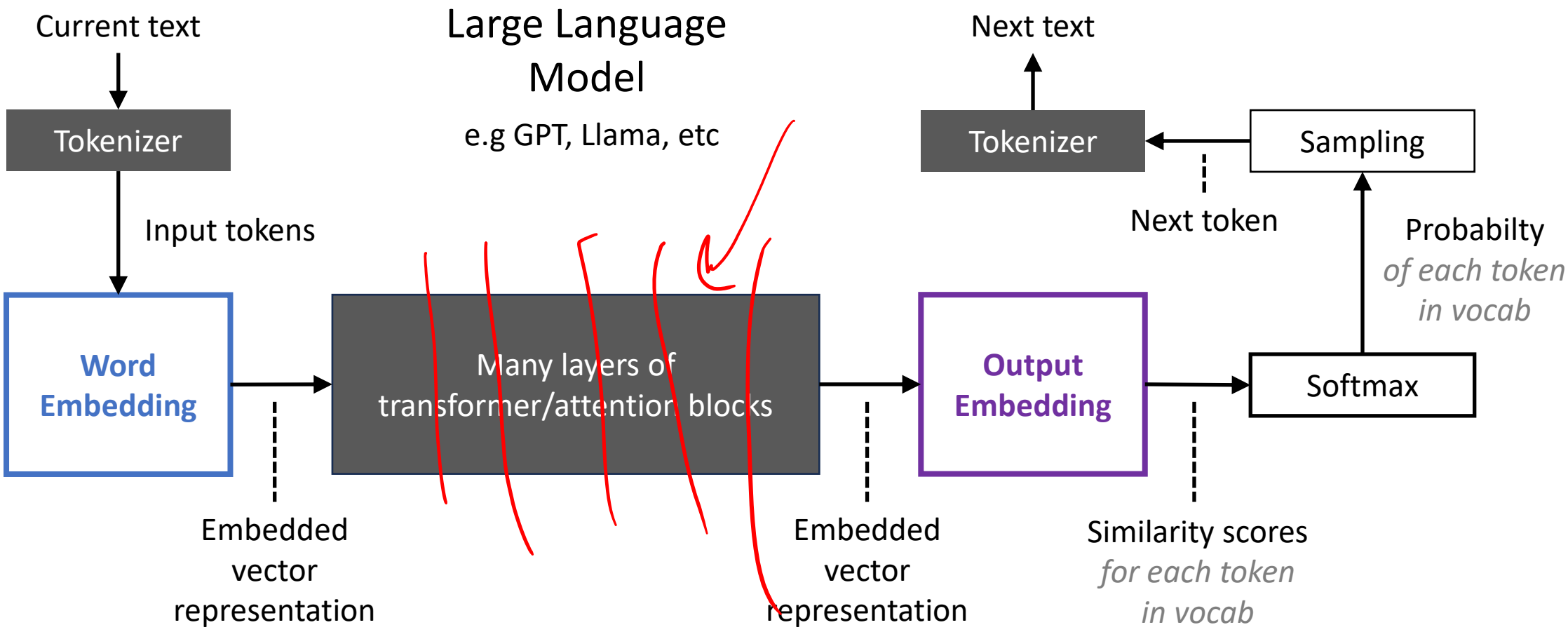
- Cosine similarity
- Softmax
- Sample from categorical distribution

Learning better vectors

- Cross-entropy loss
- SGD: looping through pairs of tokens in our corpus

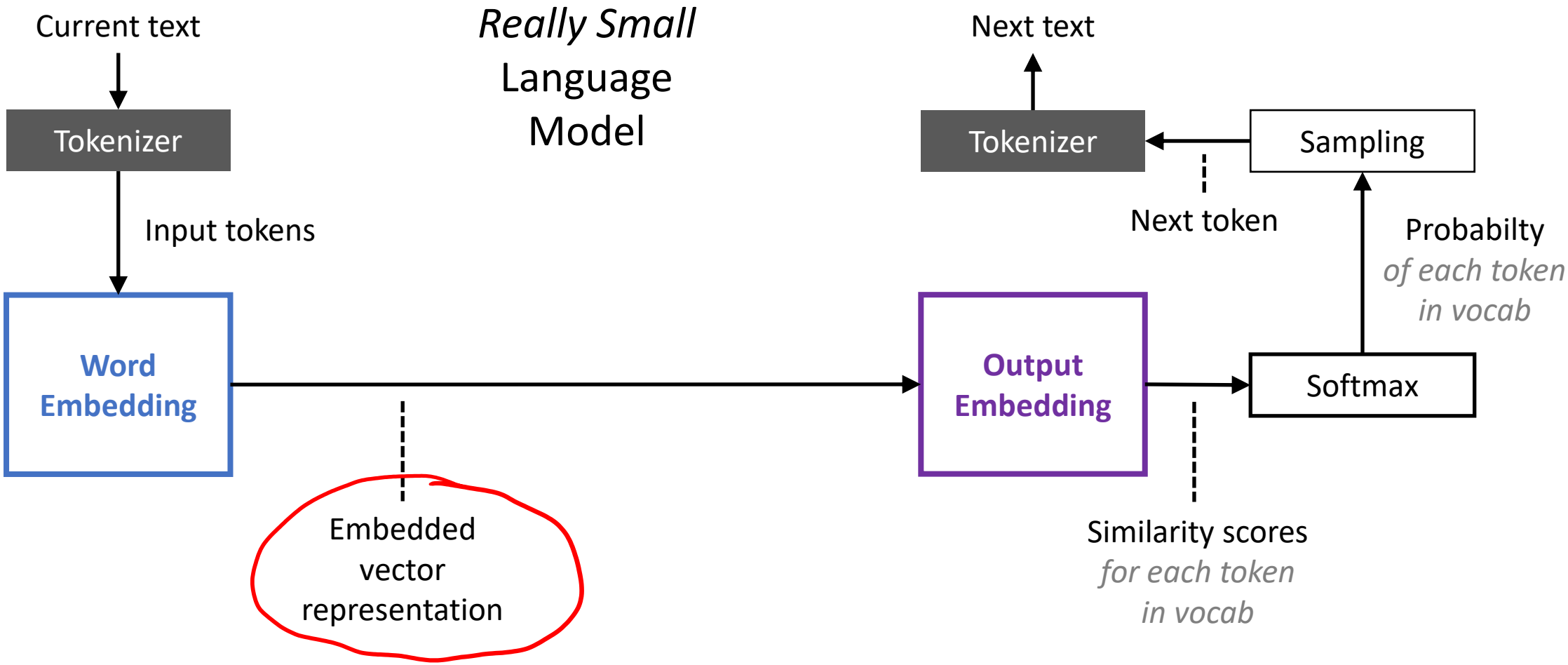
Word (Token) Embeddings

The beginning and the end of LLM networks



Simple Word Embedding LM

Building a language model with just word embedding layers ☺



Simple Word Embedding LM

Setup

Corpus:

```
The dog ran.
The dog ate.
The dog ran the zoo.
The cat ate the dog!
The cat ran the zoo.
```

(Simple) Tokenized Corpus:

```
[ 'the', 'dog', 'ran', '.', 'the',
  'dog', 'ate', '.', 'the', 'dog',
  'ran', 'the', 'zoo', '.', 'the',
  'cat', 'ate', 'the', 'dog', '!',
  'the', 'cat', 'ran', 'the',
  'zoo', '.' ]
```

Vocabulary:

```
[ '!',
  '.',
  'ate',
  'cat',
  'dog',
  'ran',
  'the',
  'zoo' ]
```

Simple Word Embedding LM

Vector representation for each token in vocabulary (initially random)

Two sets of vectors in $\mathbb{R}^M$ (we'll use $M = 2$ for better visualization)

V: to represent previous tokens

U: to represent next tokens

```
V:
!:      0.884,    0.196
.:      0.358,   -2.343
ate:   -1.085,    0.560
cat:    0.939,   -0.978
dog:    0.503,    0.406
ran:    0.323,   -0.493
the:   -0.792,   -0.842
zoo:   -1.280,    0.246
```

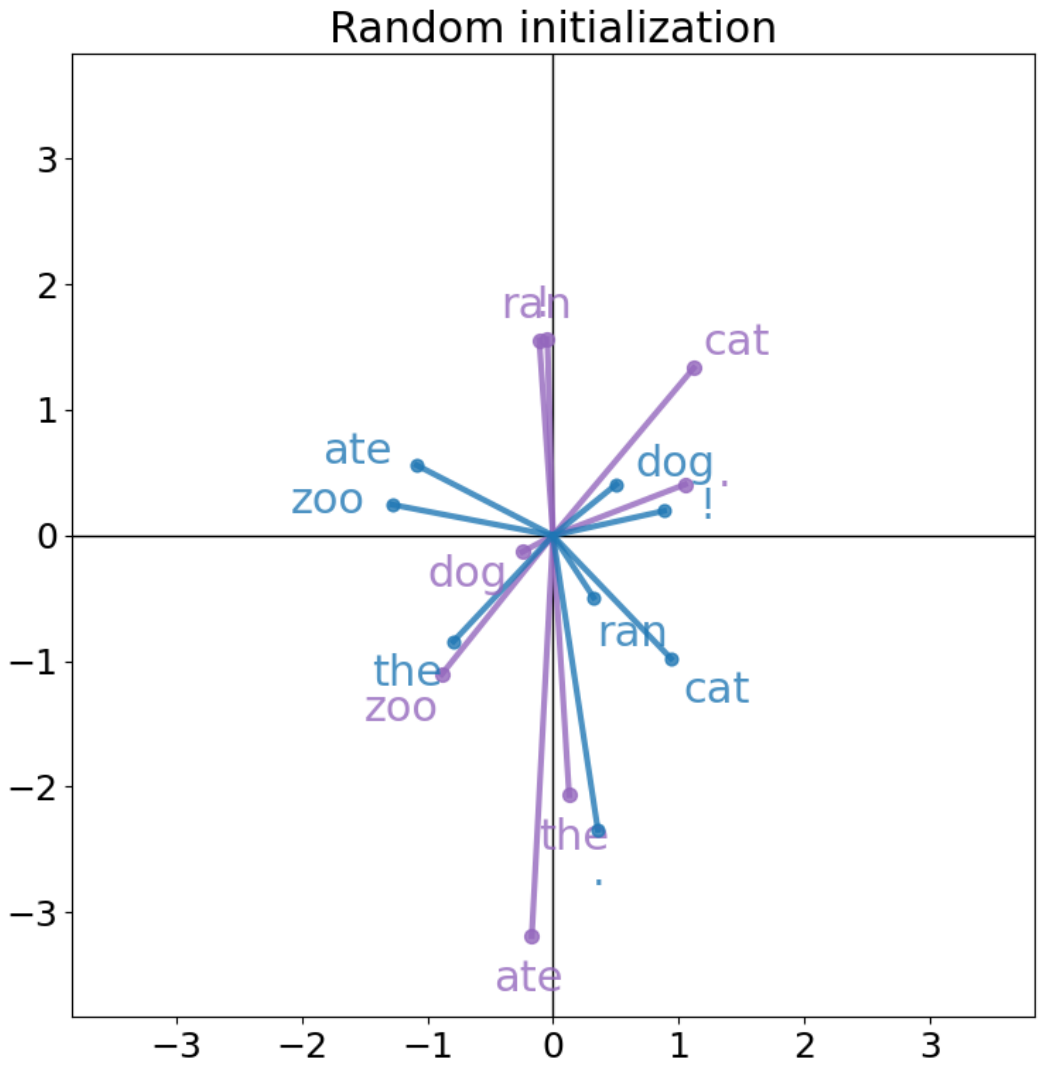
```
U:
!:     -0.044,    1.568
.:      1.051,    0.406
ate:   -0.169,   -3.190
cat:    1.120,    1.333
dog:   -0.243,   -0.130
ran:   -0.109,    1.556
the:    0.129,   -2.067
zoo:   -0.885,   -1.105
```

Simple Word Embedding LM

Vector representation for each token in vocabulary (initially random)

V: Previous

V:		
!:	0.884,	0.196
.:	0.358,	-2.343
ate:	-1.085,	0.560
cat:	0.939,	-0.978
dog:	0.503,	0.406
ran:	0.323,	-0.493
the:	-0.792,	-0.842
zoo:	-1.280,	0.246

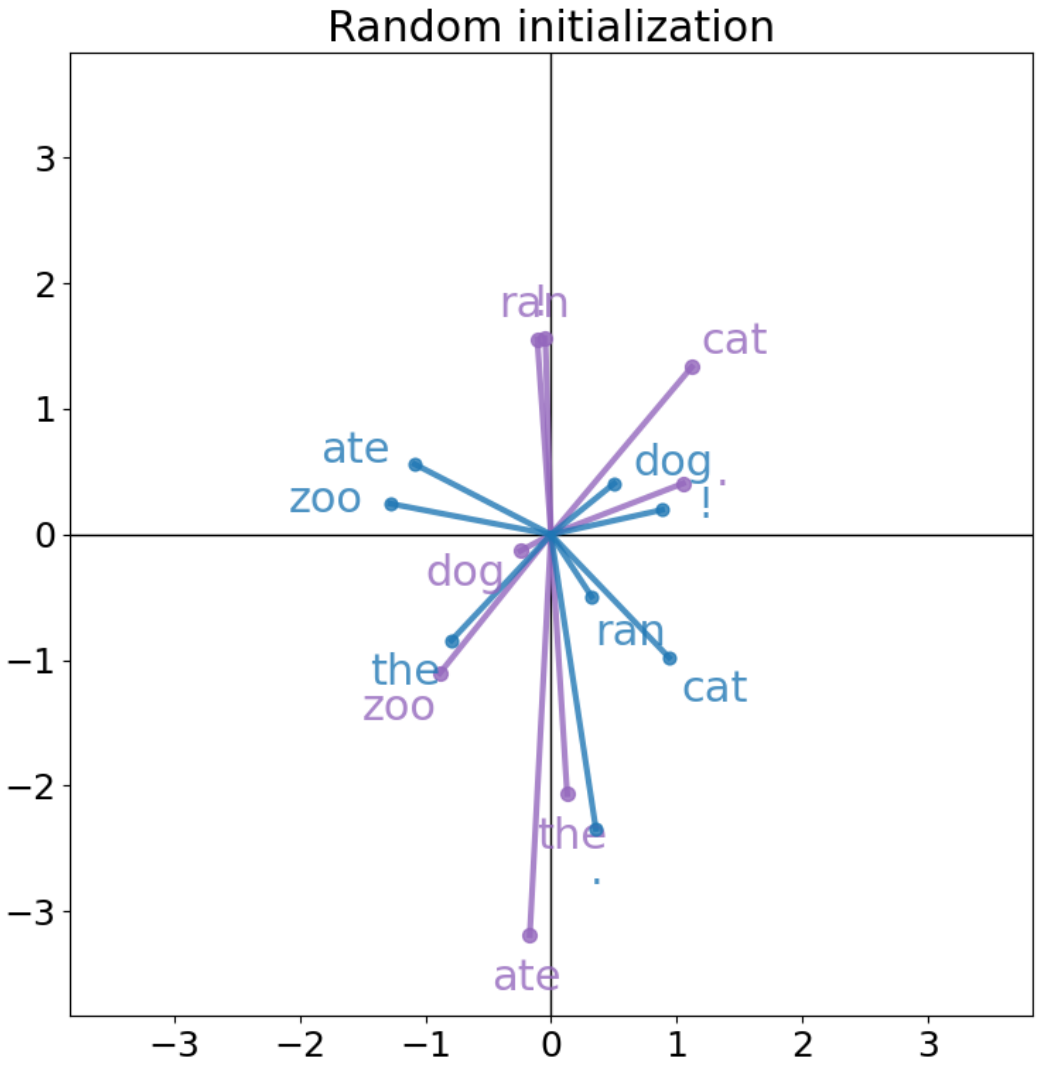


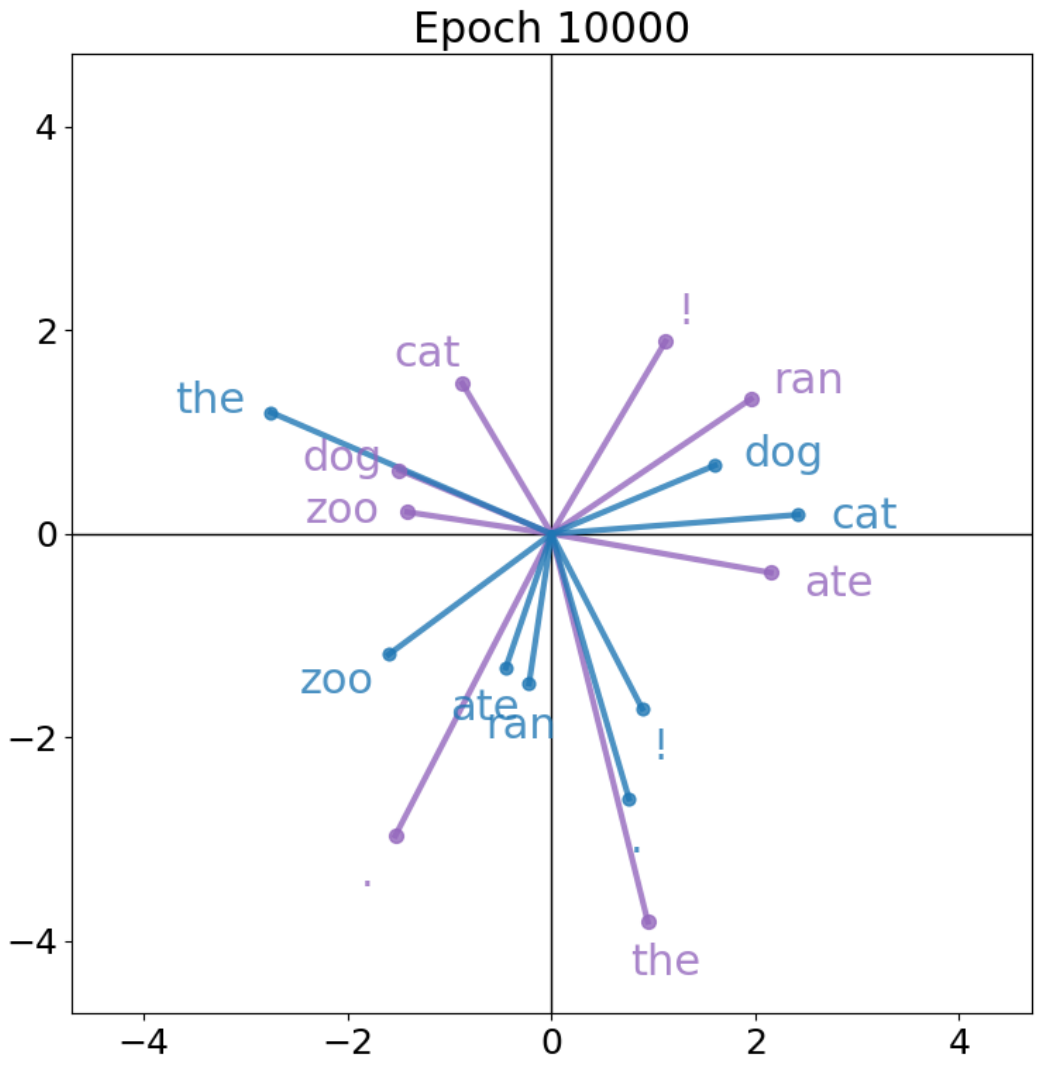
U: Next

U:		
!:	-0.044,	1.568
.:	1.051,	0.406
ate:	-0.169,	-3.190
cat:	1.120,	1.333
dog:	-0.243,	-0.130
ran:	-0.109,	1.556
the:	0.129,	-2.067
zoo:	-0.885,	-1.105

Simple Word Embedding LM

After training our LM, we'll learn more organized vectors (details later)

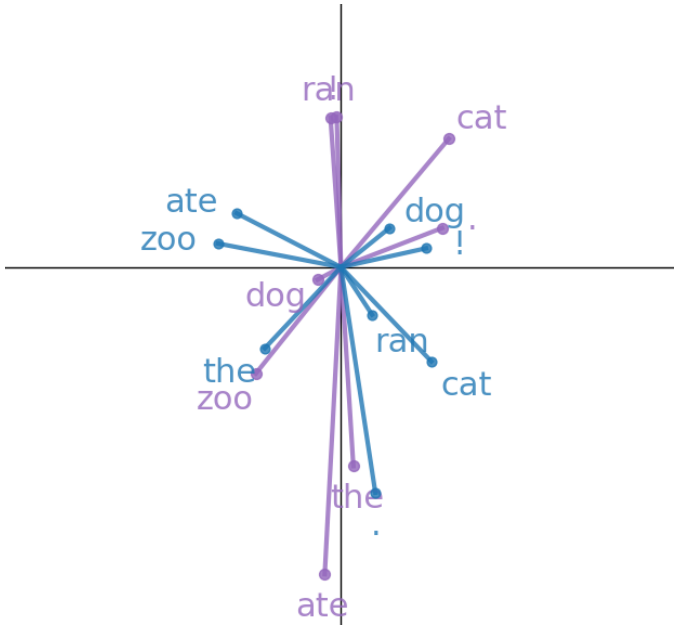




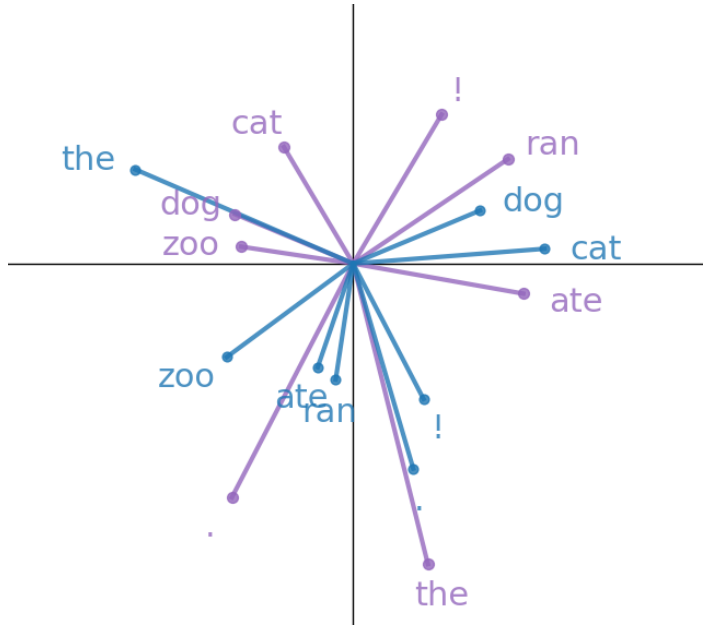
Simple Word Embedding LM

We can use either of these models to generate text, staring with "the dog"

Random vectors



Trained vectors



Generated tokens from random and trained models

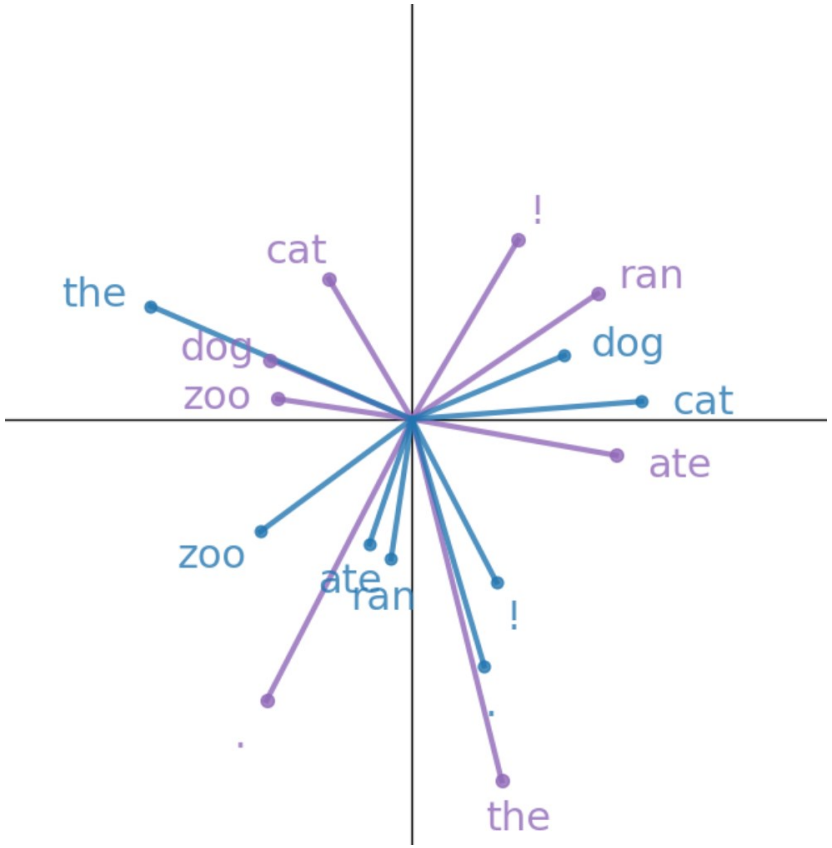
the dog cat ate ran zoo zoo
zoo dog dog cat cat ate ran .
ate . ate ate ! the ate

the dog ate . the dog ! the
zoo . the dog ran the zoo .
the dog ate the dog !

Outline: Word Embedding LM

Vector representation of vocab tokens

- Set of vectors for both previous and next token



Sampling next token

- Cosine similarity
- Softmax
- Sample from categorical distribution

Learning better vectors

- Cross-entropy loss
- SGD: looping through pairs of tokens in our corpus

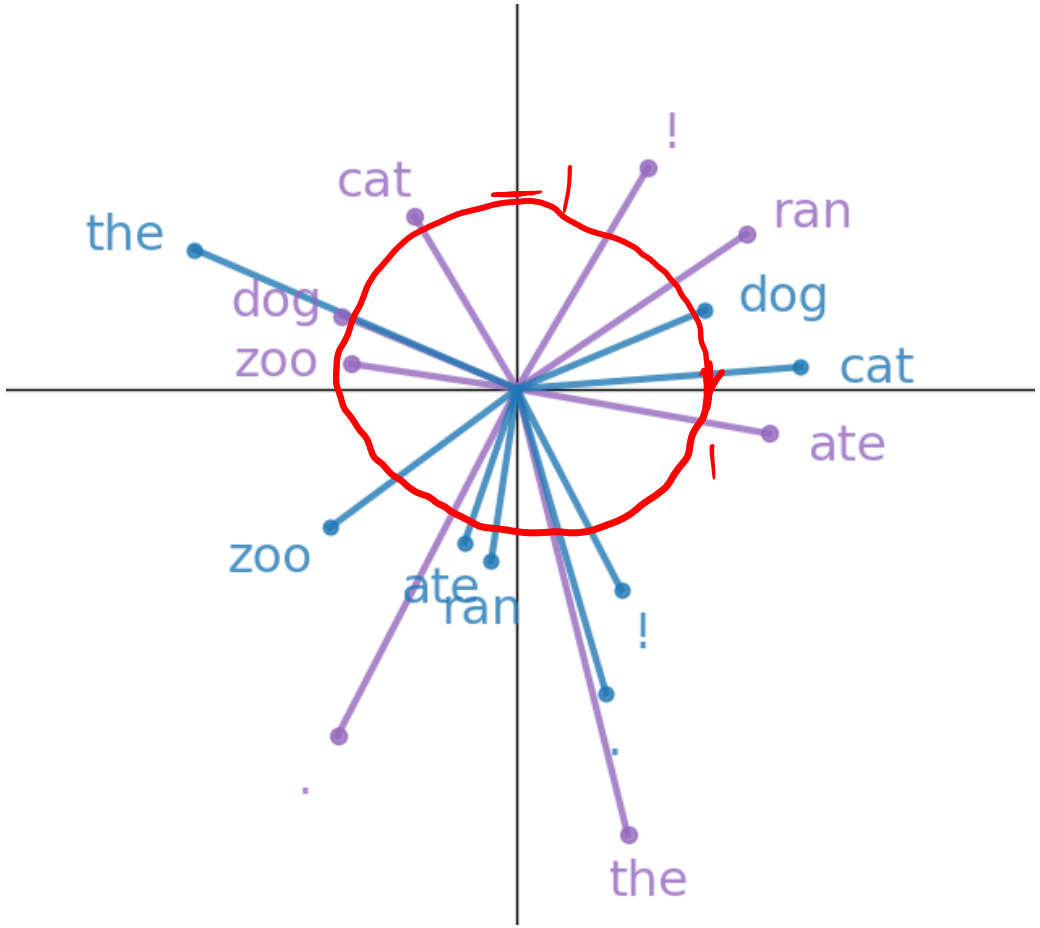
Sampling from Word Embeddings

Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

U: next tokens

Which token should be our next token?



Sampling from Word Embeddings

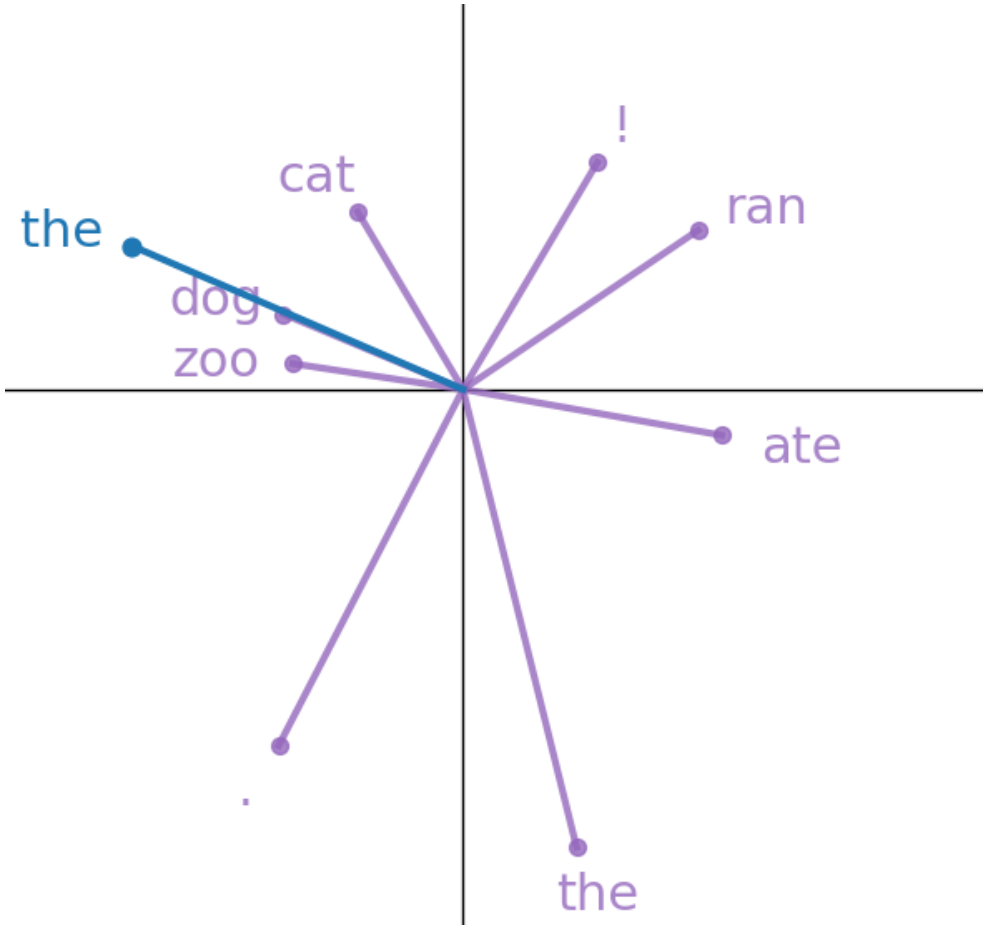
Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

U: next tokens

Which token should be our next token?

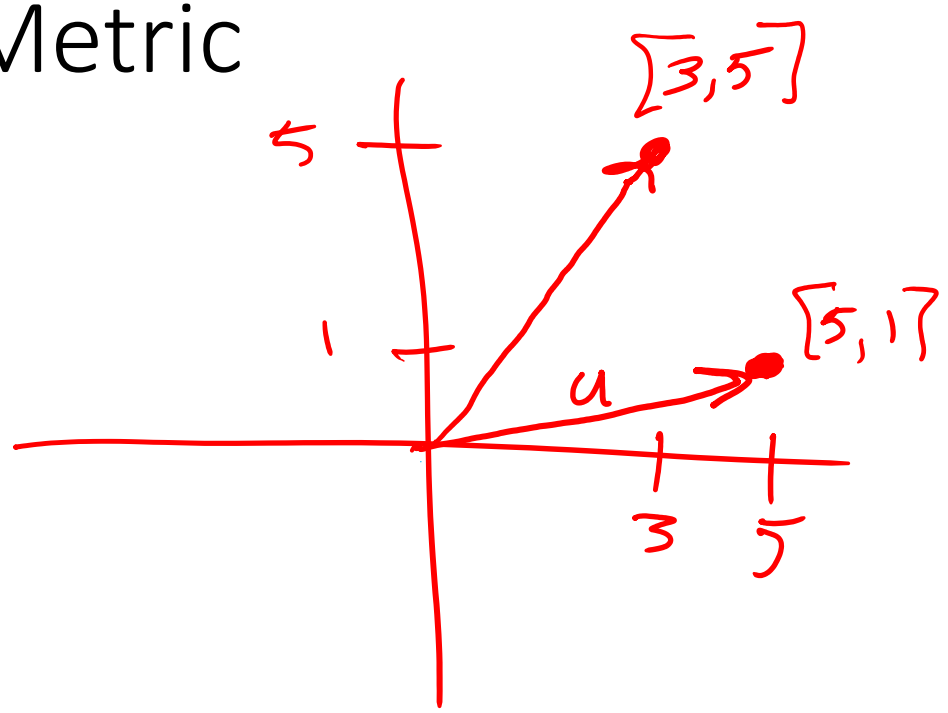
1. Lookup the index i for the vocab token 'the'
2. Access the i-th row of V, $\mathbf{v}$
3. Compare $\mathbf{v}$ to all vectors in U



(Unnormalized) Cosine Similarity Metric

We've been using Euclidean distance

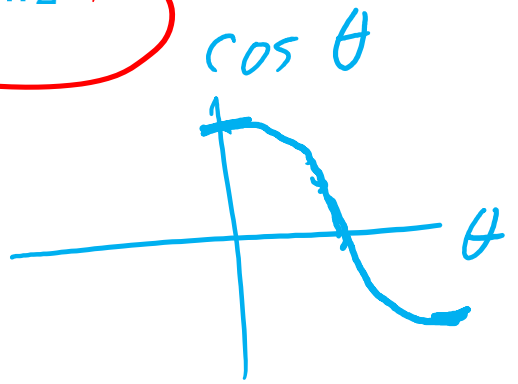
- $d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|_2$



Cosine similarity

- Two vectors are similar if their dot product is positive and big
- $f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v}$
- (Why cosine?)
  - Two vectors are similar if the angle between them is small (small angle → large $\cos\theta$)
  - $f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v} = \|\vec{u}\|_2 \|\vec{v}\|_2 \cos\theta$

$$\cos\theta = \frac{\mathbf{u}^T\mathbf{v}}{\|\mathbf{u}\|_2\|\mathbf{v}\|_2}$$

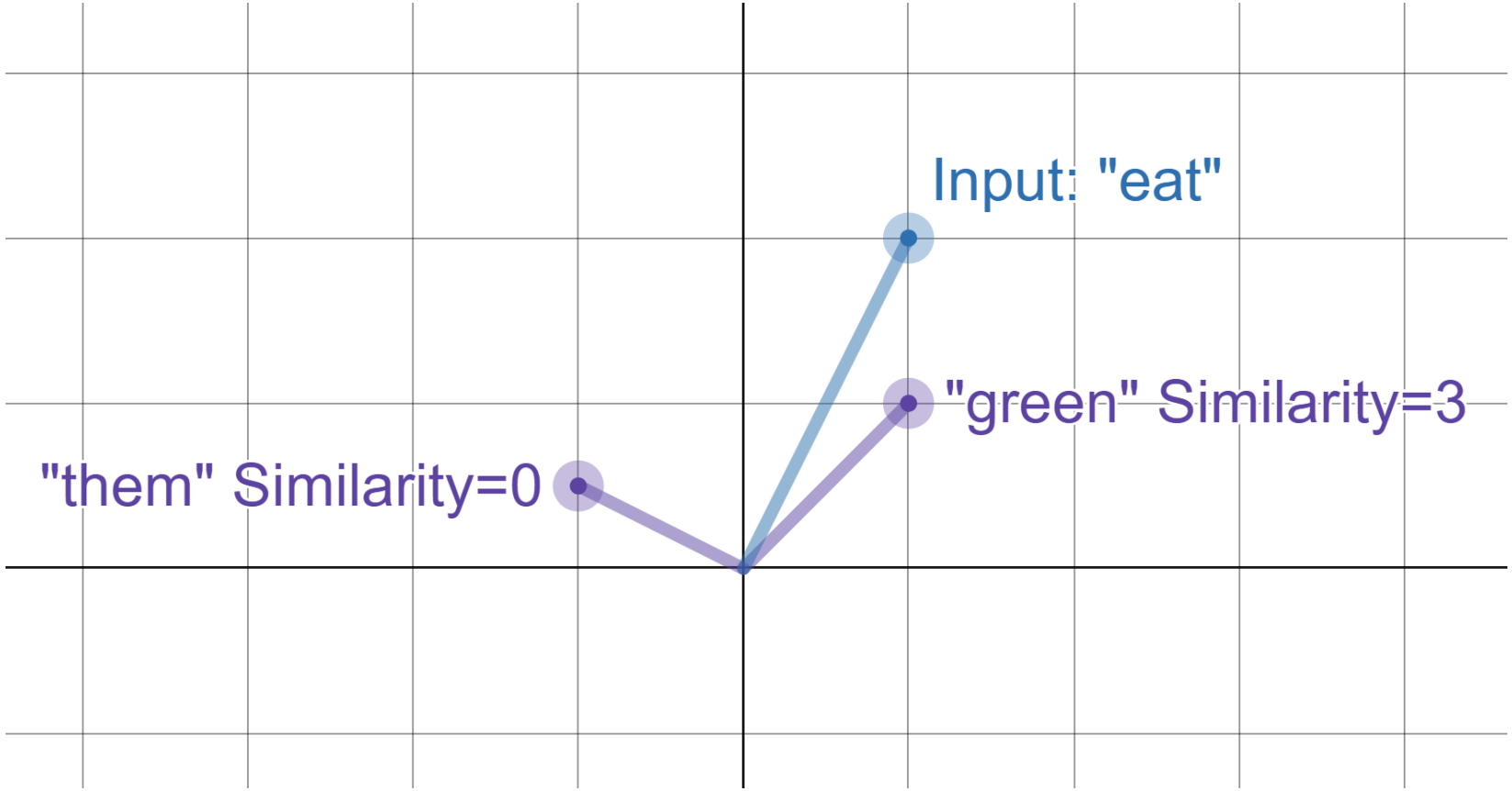


(Unnormalized) Cosine Similarity Metric

Cosine similarity Desmos demo

https://www.desmos.com/calculator/82m4zkjlkc

$$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T \mathbf{v}$$

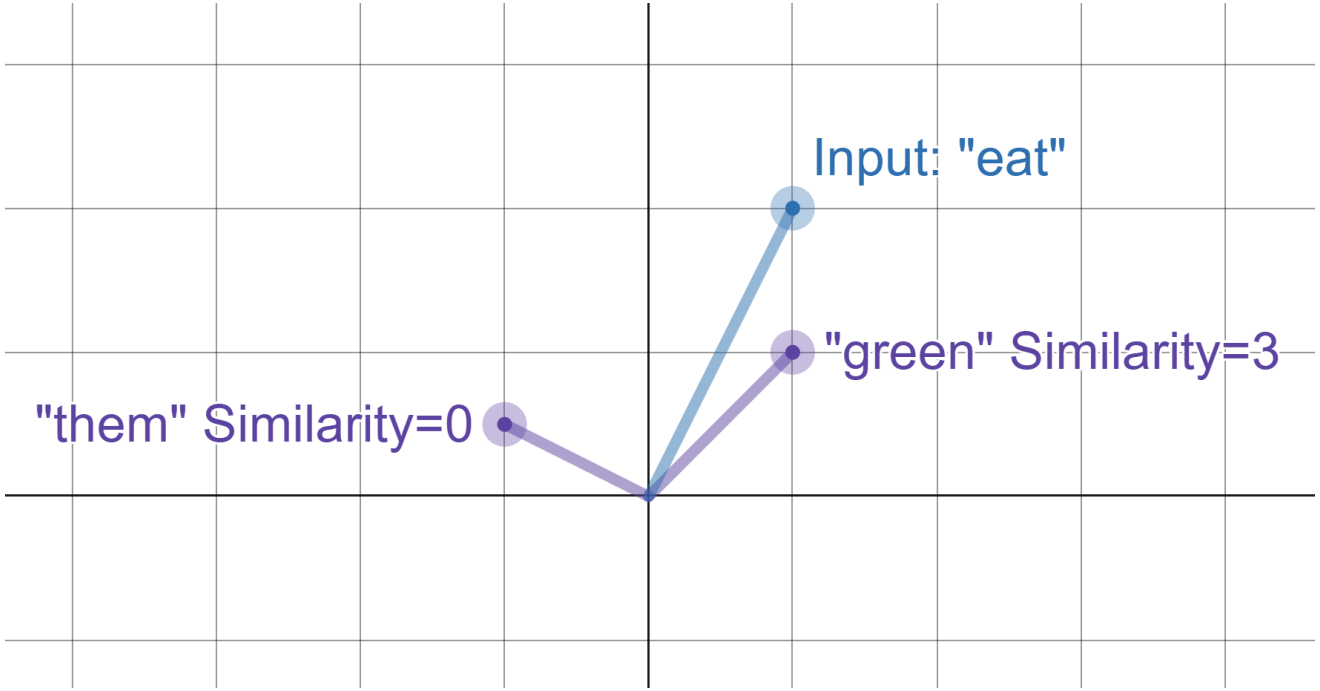


Poll 1

Take 1: Collecting vector data (about you!)

Take 2: Compute similarity score with yourself and your neighbor

$$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T \mathbf{v}$$



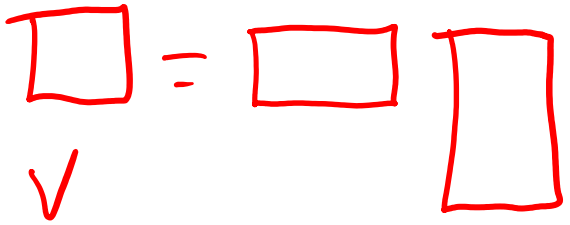
Similarity Matrix

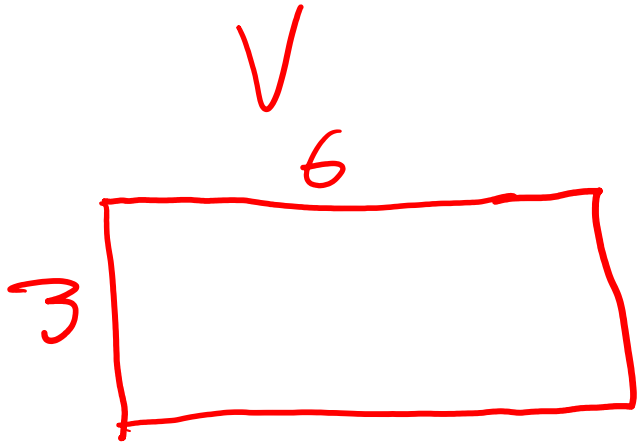
Let each row of matrix V be vector $\mathbf{v}_j^T$ representing student j in a group.

We can compare all pairs of vectors by doing matrix multiplication of V with itself.

$$f(\boldsymbol{V}) = V\,V^T$$

3 x 3





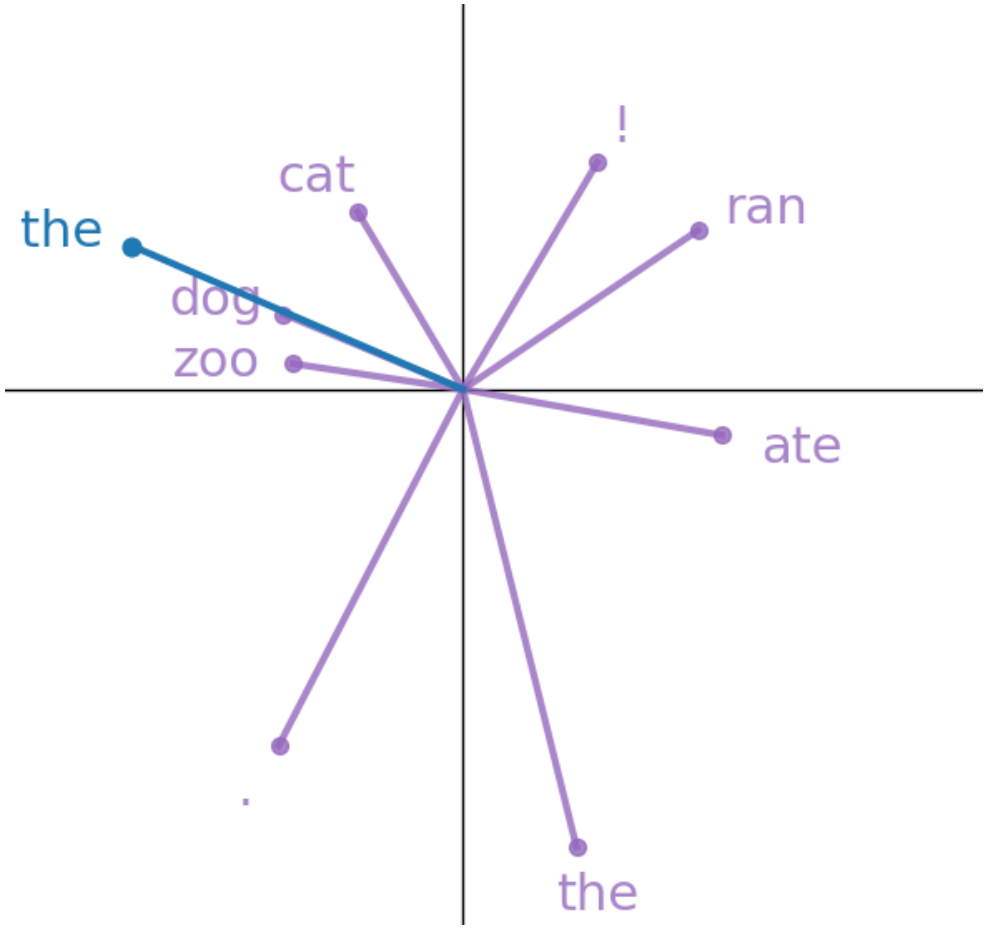
Sampling from Word Embeddings

Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

U: next tokens

1. Compute similarity scores
   $\mathbf{s} = U\mathbf{v}$

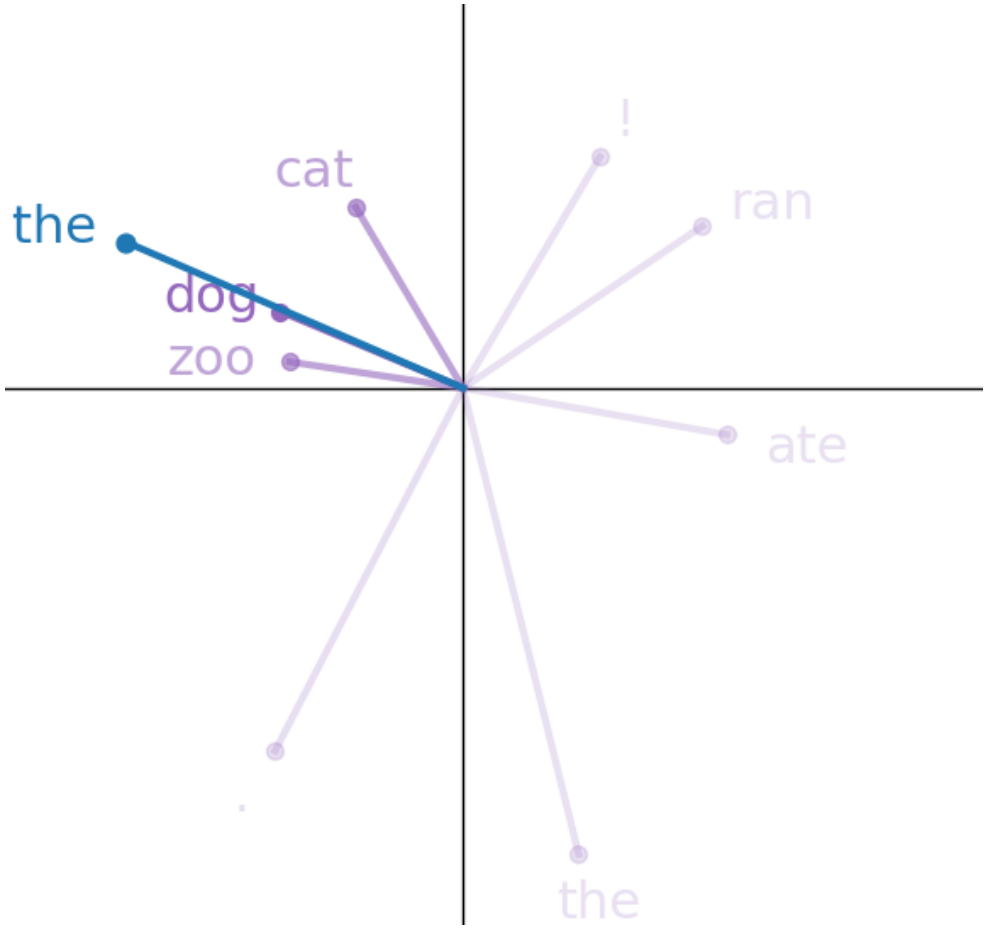


Sampling from Word Embeddings

Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

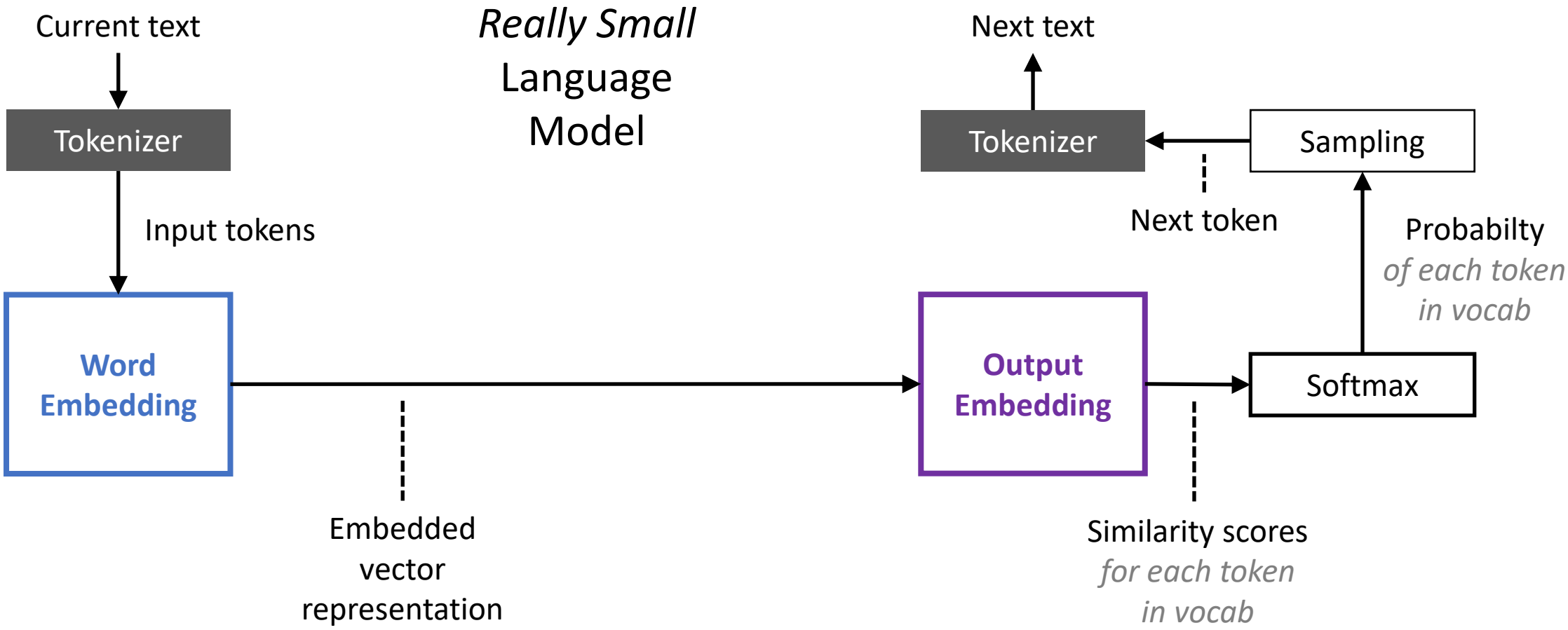
U: next tokens



1. Compute similarity scores $\mathbf{s} = U\mathbf{v}$
2. Convert to probabilities $\hat{\mathbf{y}} = g_{softmax}(\mathbf{s})$
3. Sample from Categorical distribution defined by $\hat{\mathbf{y}}$
4. LOOP: next token → prev

Simple Word Embedding LM

Building a language model with just word embedding layers ☺



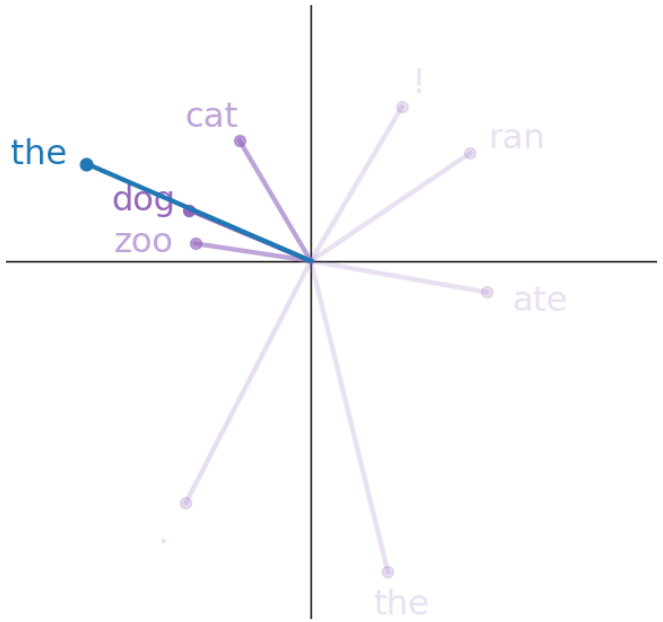
Poll 2

Given a vocab size of 10 and an embedded dimension of size 2 how many parameters do have in a word embedding language model?

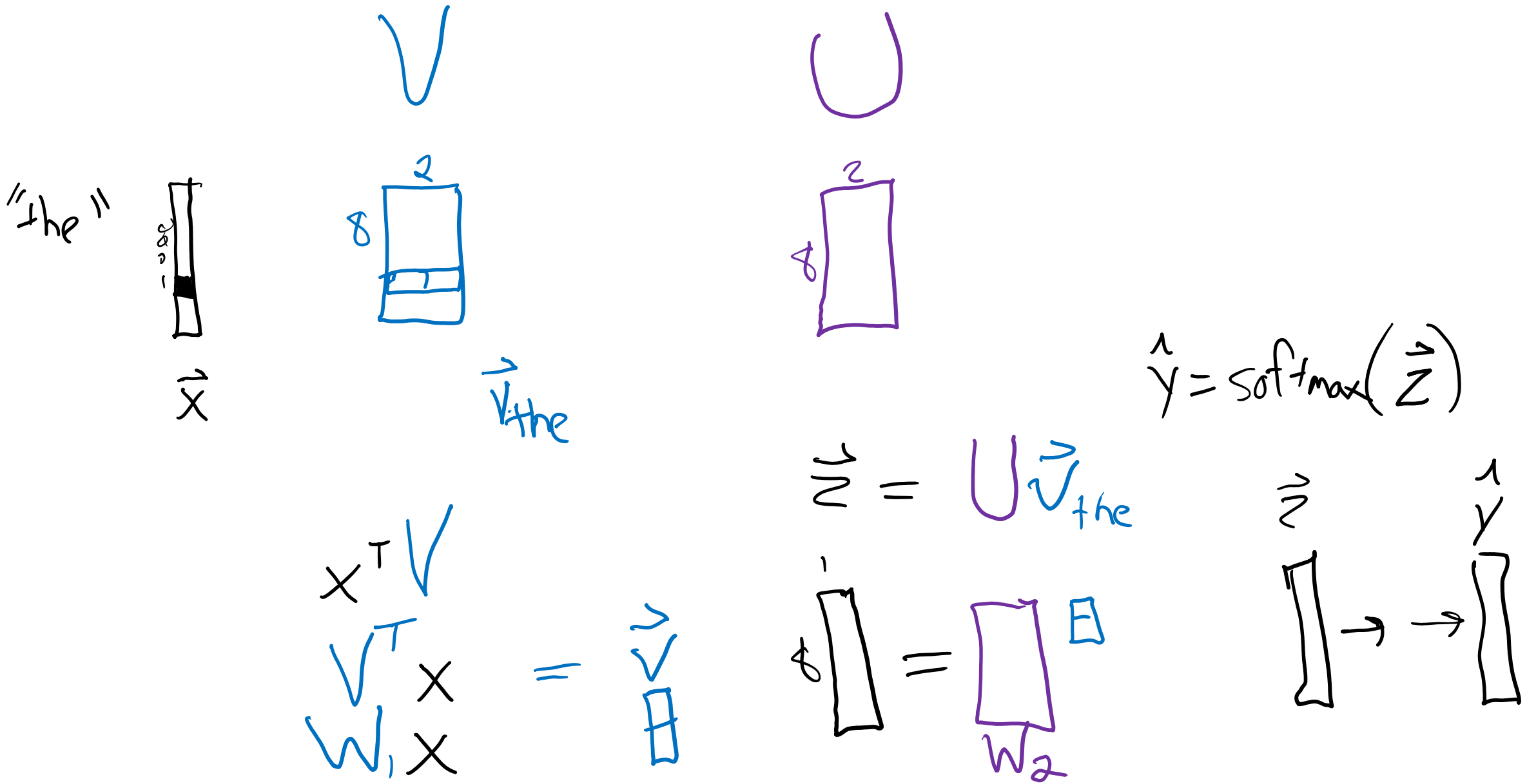
V: previous tokens

U: next tokens

1. Compute similarity scores $\mathbf{s} = U\mathbf{v}$
2. Convert to probabilities $\hat{\mathbf{y}} = g_{softmax}(\mathbf{s})$
3. Sample from Categorical distribution defined by $\hat{\mathbf{y}}$
4. LOOP: next token → prev



Linear Layer for Word Embeddings

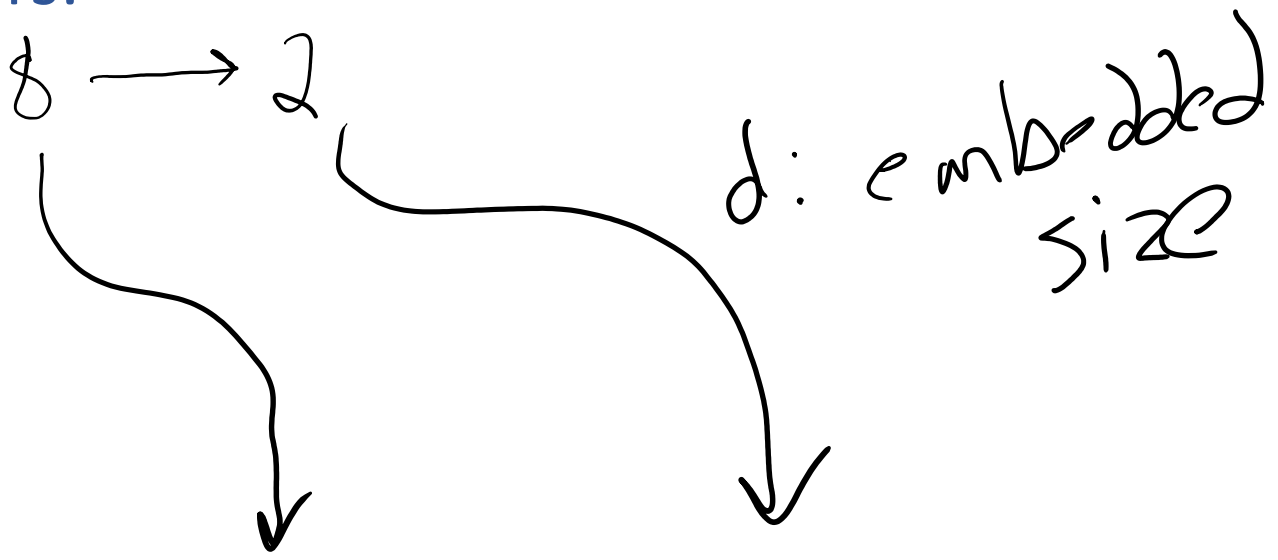


PyTorch for Word Embedding LM

Two matrices of features vectors:

W_1: for context

W_2: for next token



Using PyTorch

```
WordEmbedLM(
  (encode): Linear(in_features=vocab_size, out_features=2)
  (decode): Linear(in_features=2, out_features=vocab_size)
)
```

```
F.softmax(model.forward(x_onehot))
```

PyTorch for Word Embedding LM

Two matrices of features vectors:

W_1: for context

W_2: for next token

Using PyTorch

`torch.nn.Embedding(num_embeddings, embedding_dim)`

```
WordEmbedLM(
  (encode): Linear(in_features=vocab_size, out_features=2)
  (decode): Linear(in_features=2, out_features=vocab_size)
)
```

`x_index`

```
F.softmax(model.forward(x_onehot))
```

Learning Better Vectors

(Simple) Tokenized Corpus:

```
[ 'the', 'dog', 'ran', '.', 'the',
  'dog', 'ate', '.', 'the', 'dog',
  'ran', 'the', 'zoo', '.', 'the',
  'cat', 'ate', 'the', 'dog', '!',
  'the', 'cat', 'ran', 'the',
  'zoo', '.' ]
```

Classic ML recipe:

1. Training data

2. Hypothesis function

$$\hat{\mathbf{y}} = g_{softmax}(U\mathbf{v})$$

3. Formulate objective

   Loss:

   Objective: $\frac{1}{\mathrm{N}}\sum_i^N J^{(i)}(U,V)$ $J^{(i)}(U,V)$

4. SGD to find parameters that optimize the objective

Feature Learning
Image and Audio

Feature Learning

Learning a lower dimensional representation of our data rather than doing feature engineering to represent the data

Also called feature embedding

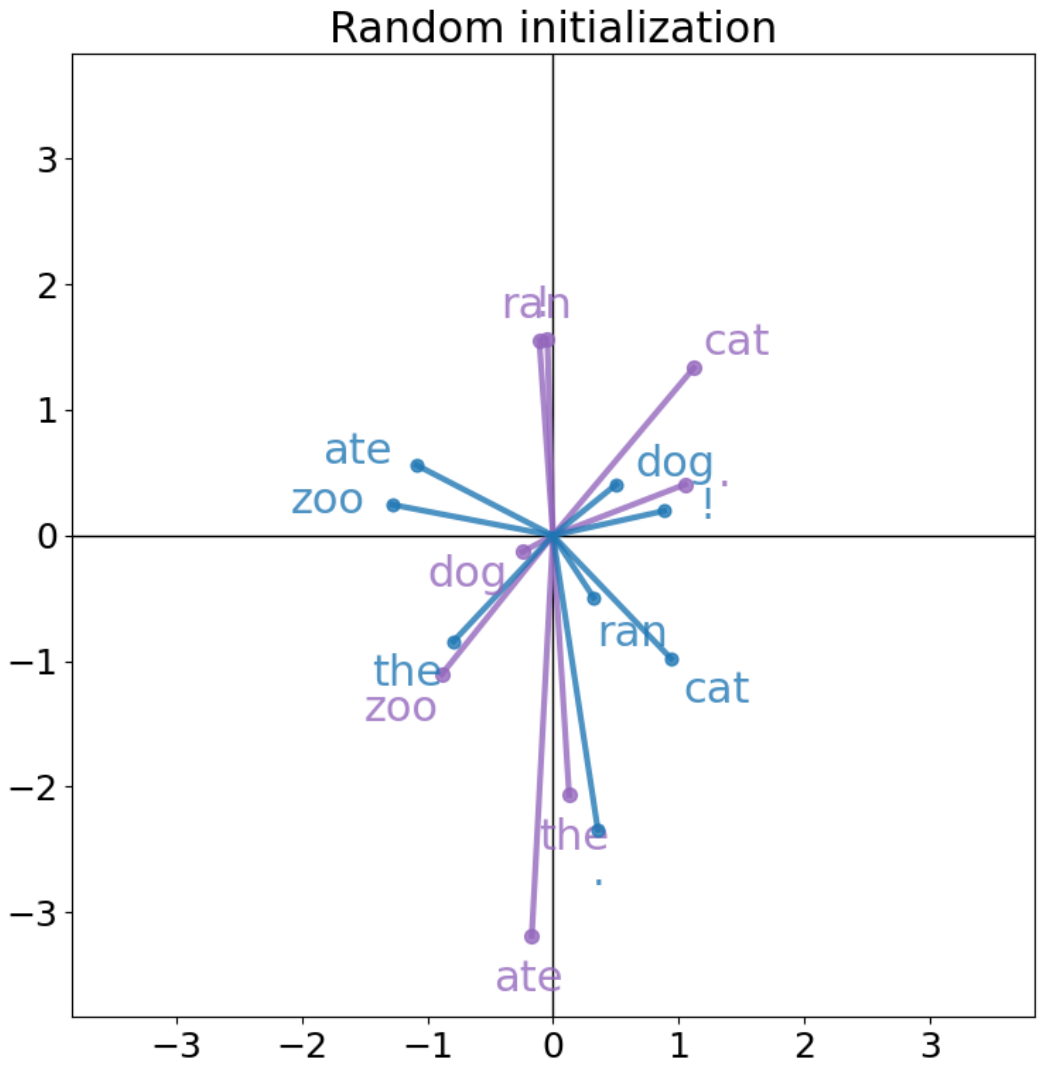
(embedding data in a lower/different dimensional space)

Word Embeddings

Vector representation for each token in vocabulary (initially random)

V: Previous

V:		
!:	0.884,	0.196
.:	0.358,	-2.343
ate:	-1.085,	0.560
cat:	0.939,	-0.978
dog:	0.503,	0.406
ran:	0.323,	-0.493
the:	-0.792,	-0.842
zoo:	-1.280,	0.246

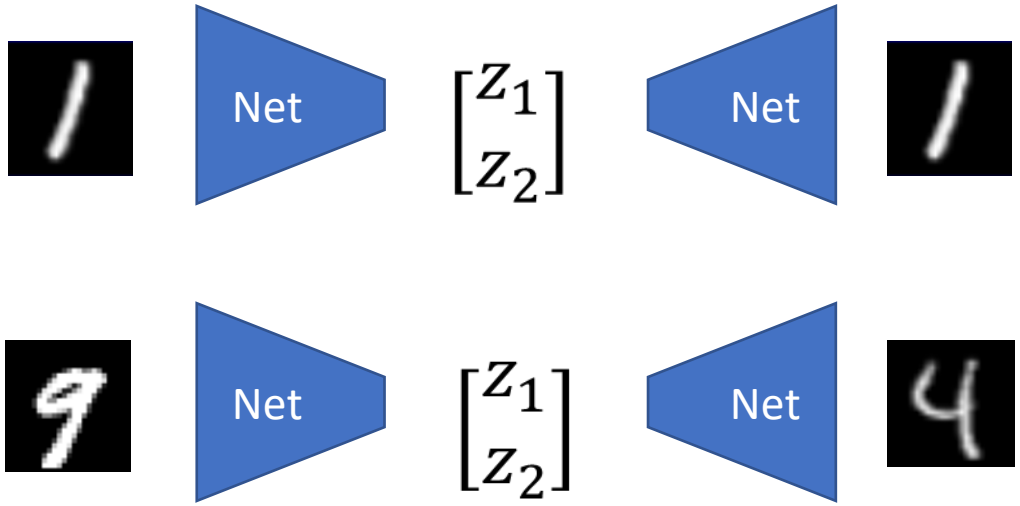


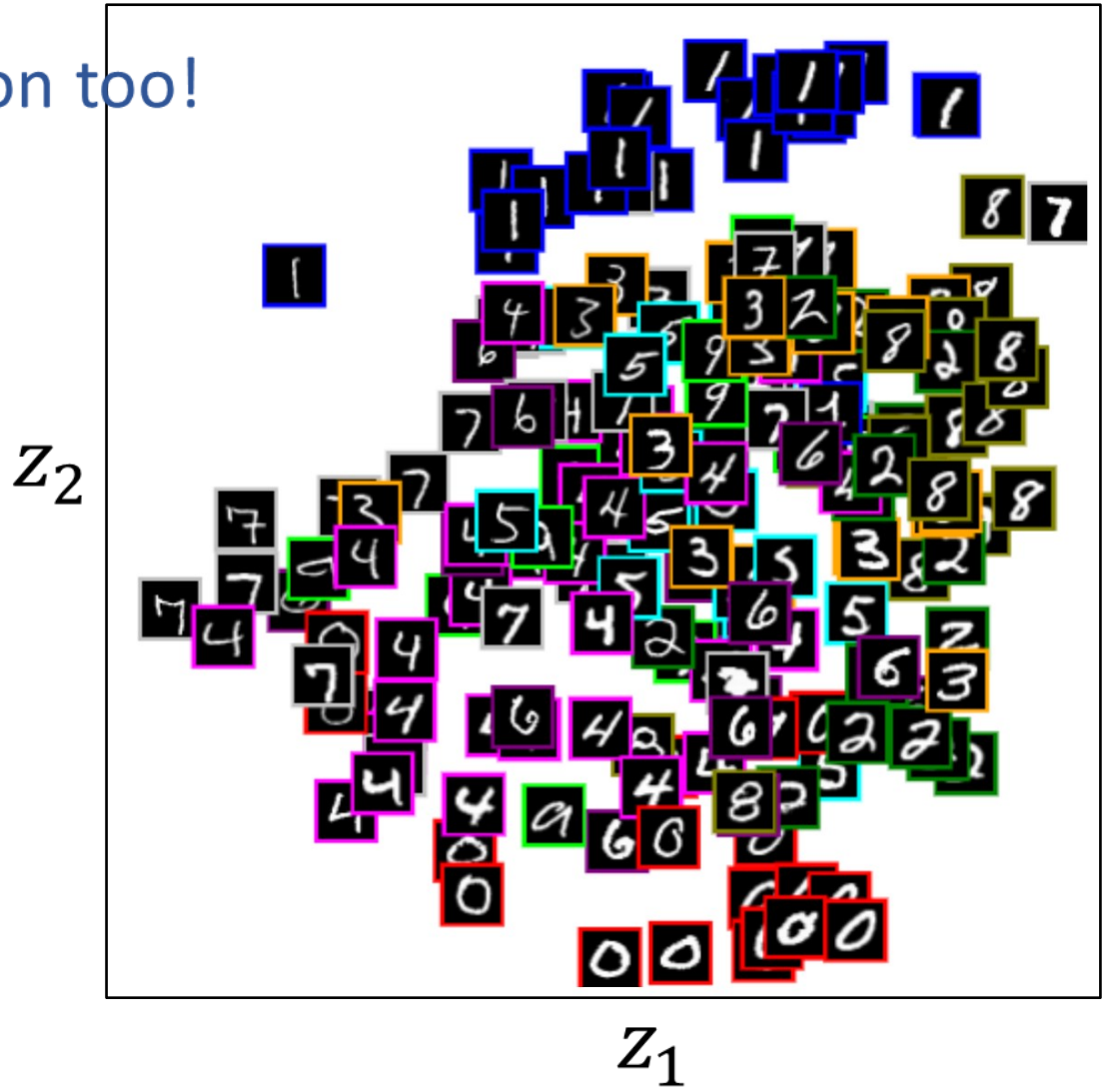
U: Next

U:		
!:	-0.044,	1.568
.:	1.051,	0.406
ate:	-0.169,	-3.190
cat:	1.120,	1.333
dog:	-0.243,	-0.130
ran:	-0.109,	1.556
the:	0.129,	-2.067
zoo:	-0.885,	-1.105

Learning to Organize Data

Neural networks can learn to organization too!

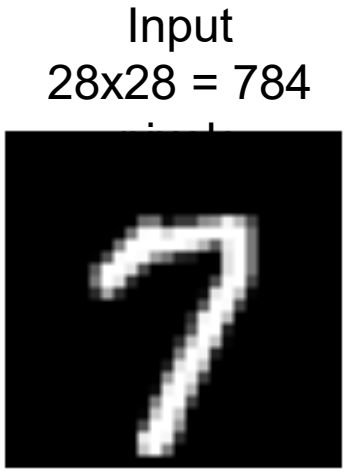


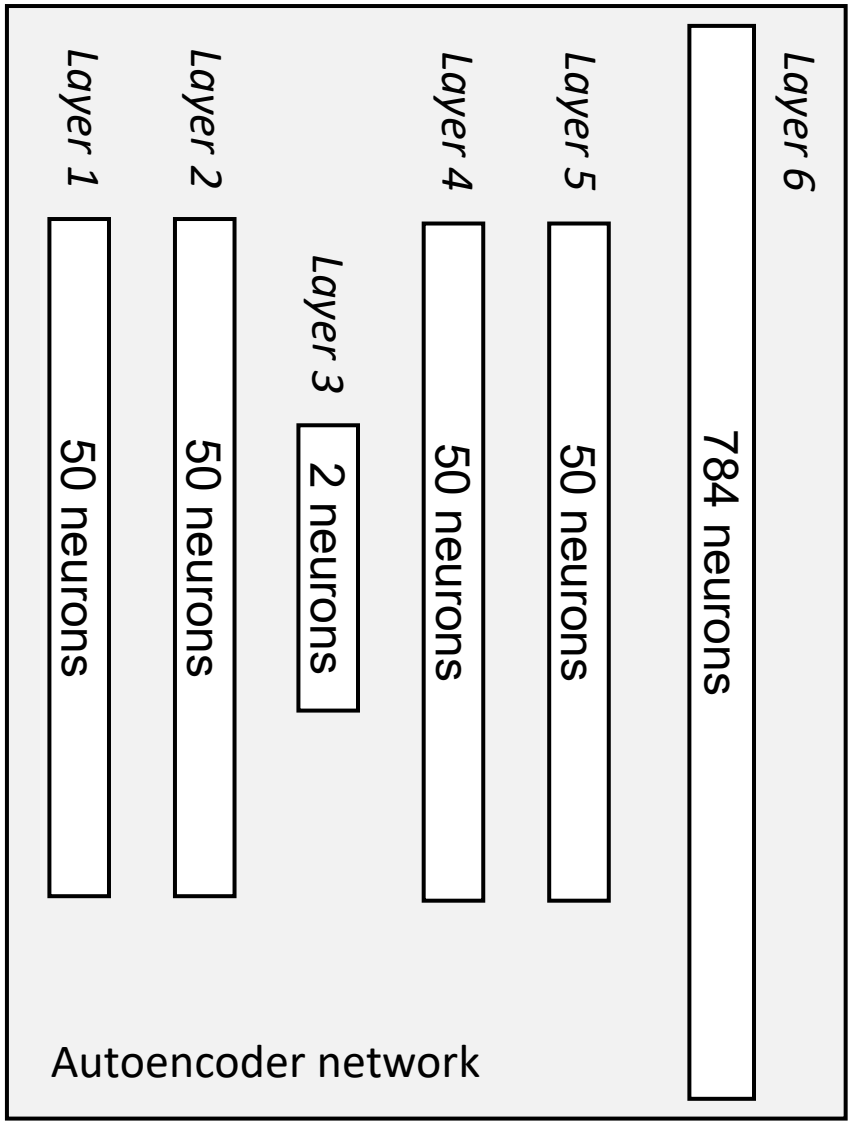


https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html

Digit Autoencoder

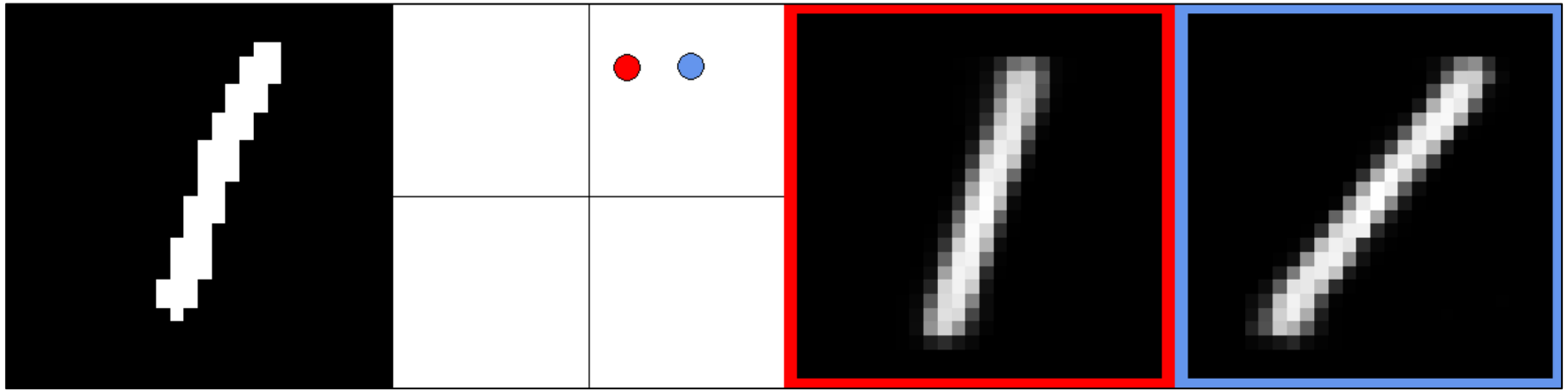
https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html





Digit Autoencoder

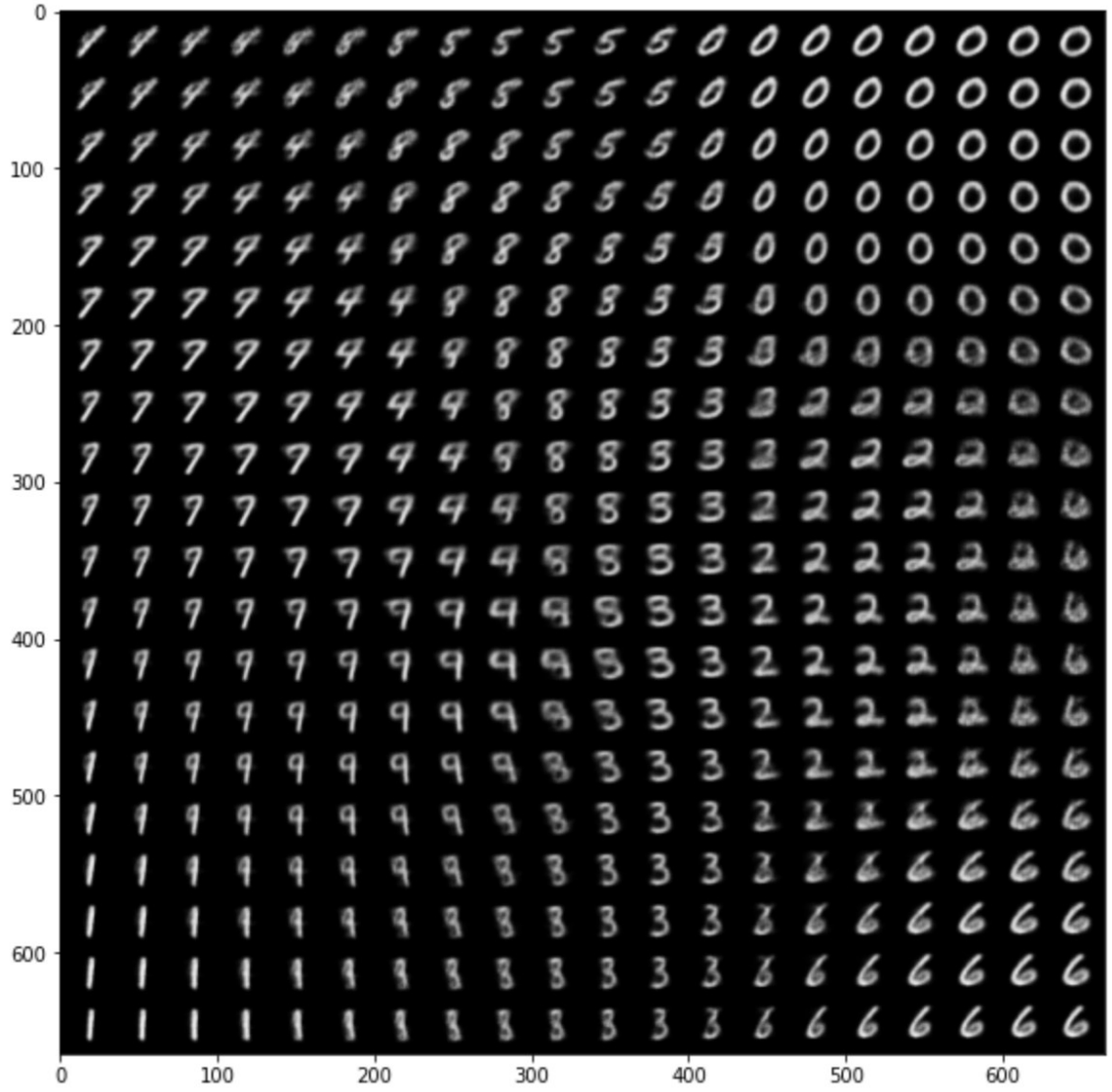
Demo: Using a learned feature space



Variational Autoencoder Demo

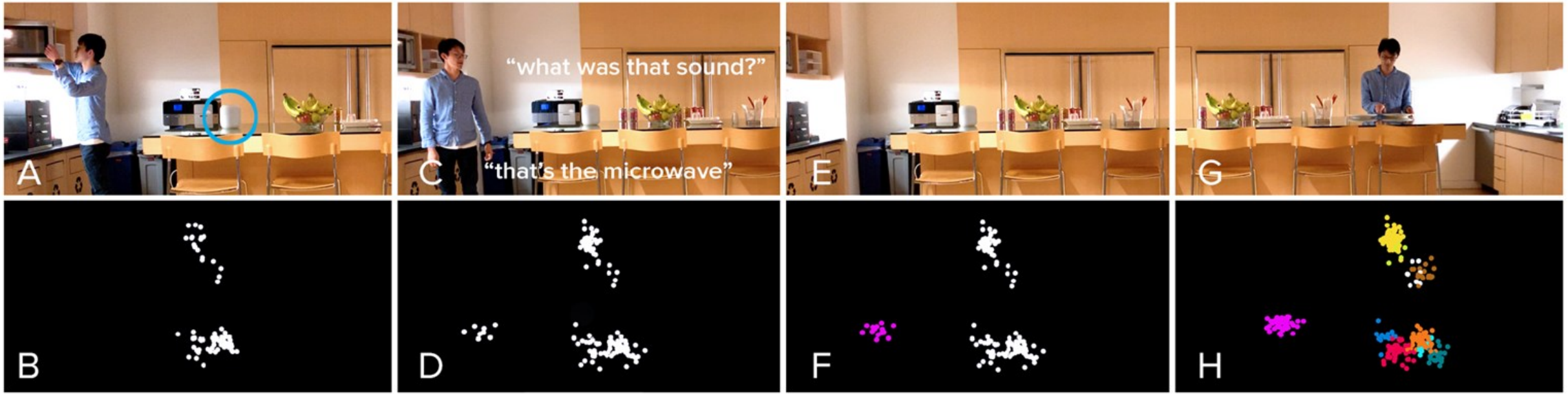
Zhuoyue Lyu, Safinah Ali, and Cynthia Breazeal. EAAI 2022.

https://colab.research.google.com/gist/ZhuoyueLyu/5046225a9ae3675cf633c1df5f63be06/digits-interpolation-notebook-eaai.ipynb



Feature Learning

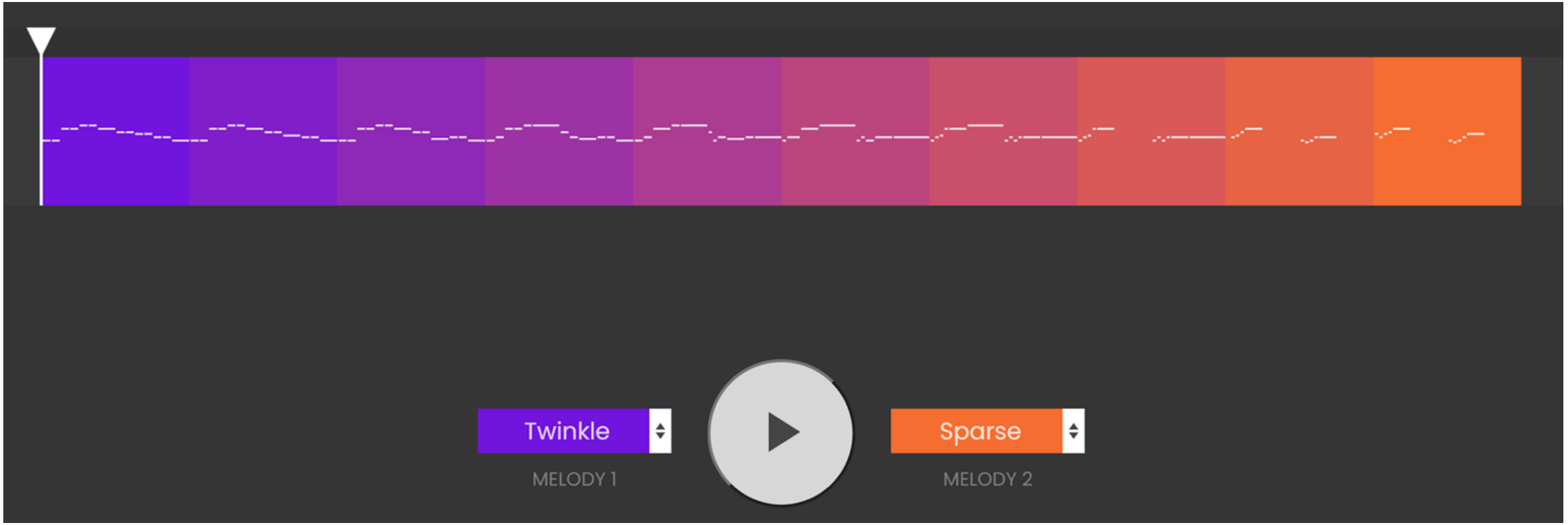
Listen Learner



https://chrisharrison.net/index.php/Research/ListenLearner

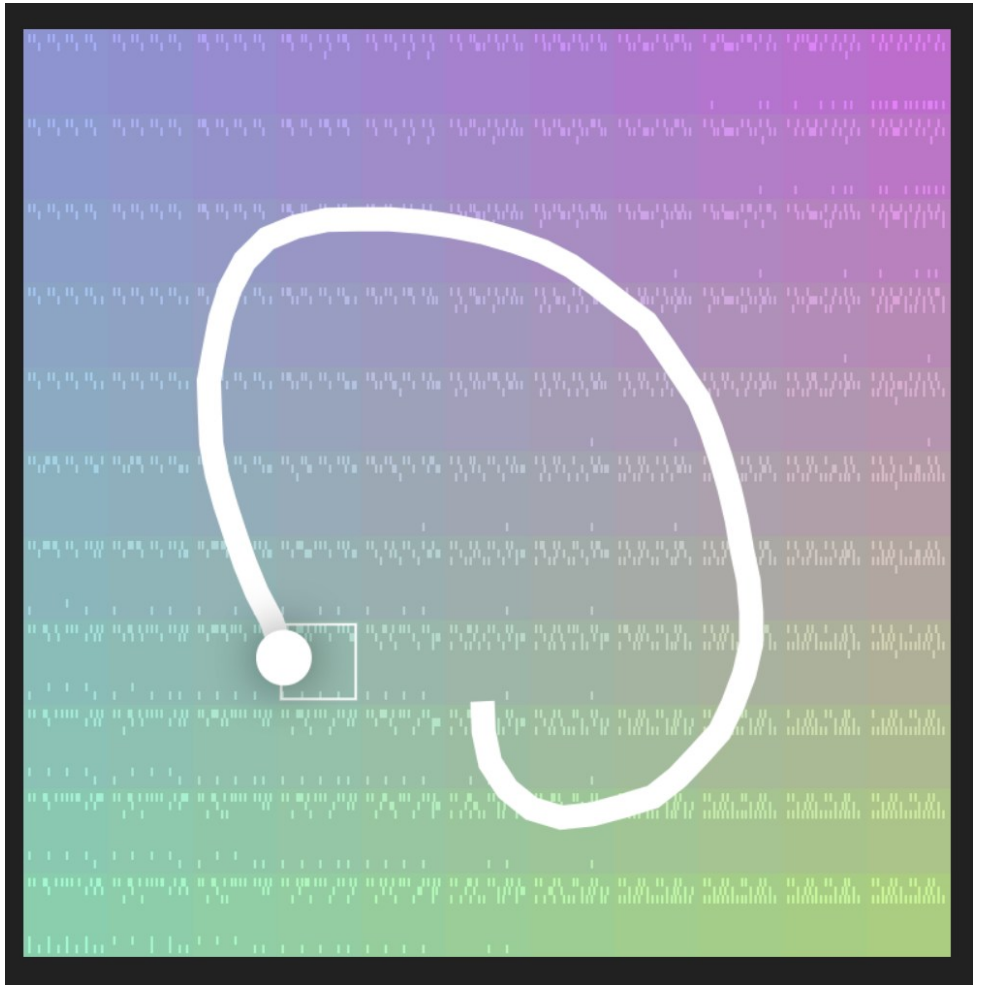
Exploring Feature Space

https://experiments.withgoogle.com/ai/melody-mixer/view/



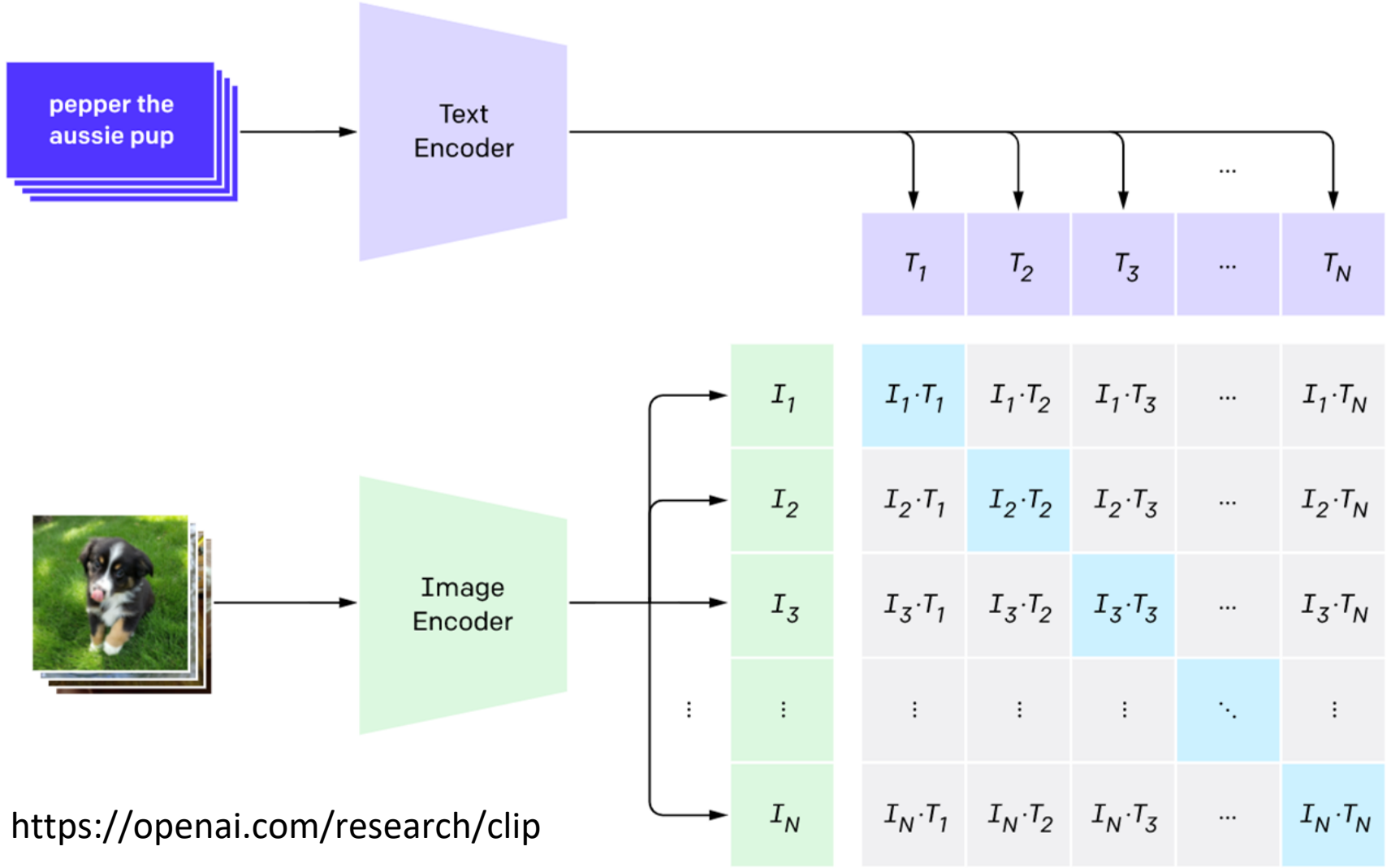
Exploring Feature Space

https://experiments.withgoogle.com/ai/beat-blender/view/



Feature Learning

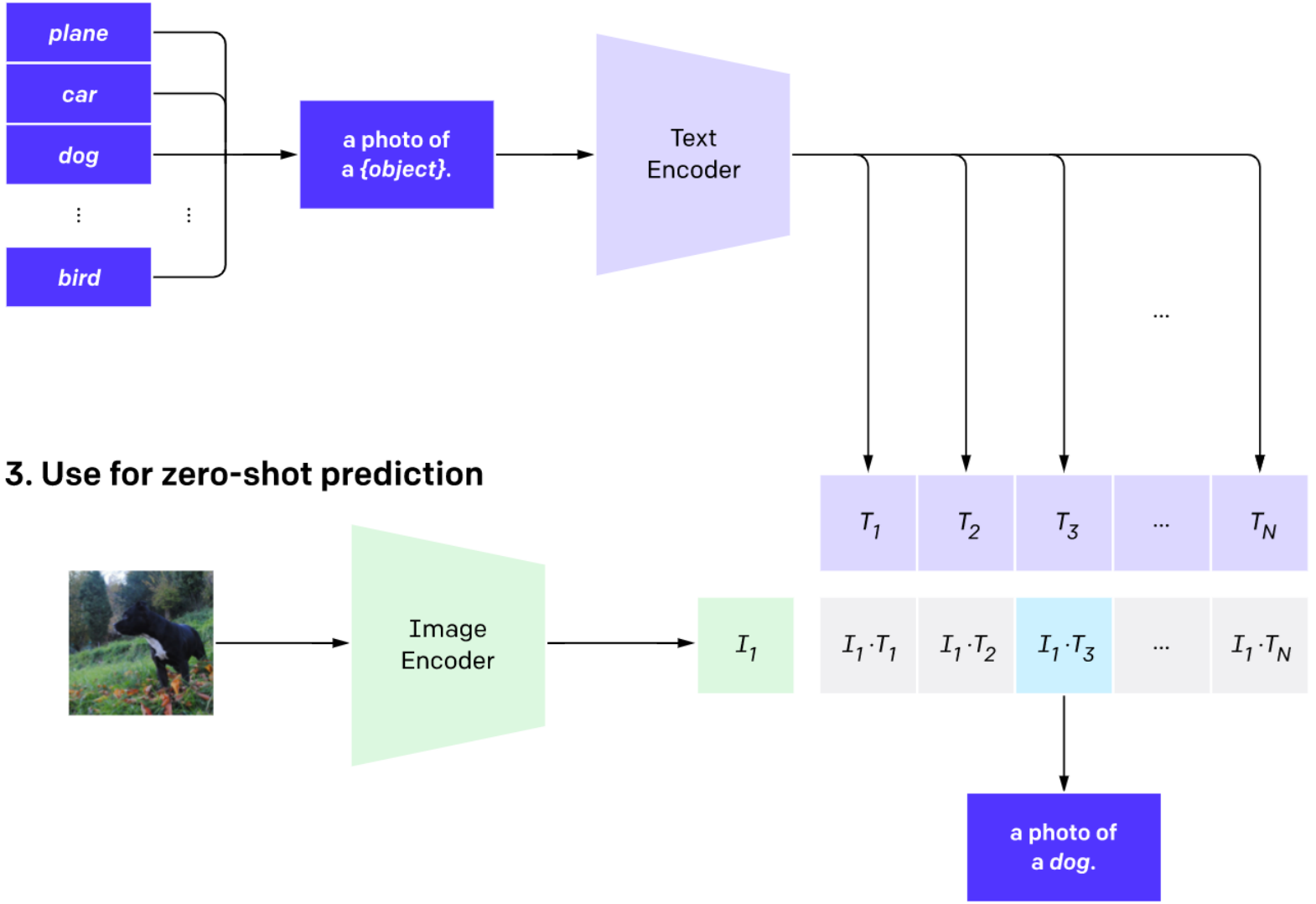
CLIP: Connecting text and images



https://openai.com/research/clip

Feature Learning

CLIP: Connecting text and images



https://openai.com/research/clip