Deep Learning

Day 10: Sequential Data and Language Modeling

CSCI 1470

Eric Ewing

Tuesday, 10/7/25

Final Project Logistics

DL Day (Poster Session) will take place on Thursday, Dec 11th

If you're capstoning, you don't need to fill out a form (for me)

Groups of 3-4 students, assigned to a mentor

Project Options

Core to all: You need to train a neural network (i.e., using an LLMfor something new does not count if no training occurs)

1. Re-implement a paper, extend to different dataset, run new experiments, etc.
2. Try something new (i.e., ask a research question)
3. AI enabled: Use LLM coding tools to create a DL project. (higher expectations in terms of production and scope of project)

Project Options

Core to all: You need to train a neural network (i.e., using an LLMfor something new does not count if no training occurs)

1. Re-implement a paper, extend to different dataset, run new experiments, etc.
2. Try something new (i.e., ask a research question)
3. AI enabled: Use LLM coding tools to create a DL project. (higher expectations in terms of production and scope of project)

Written details will come soon!
Step 1: what interests you?

Recap

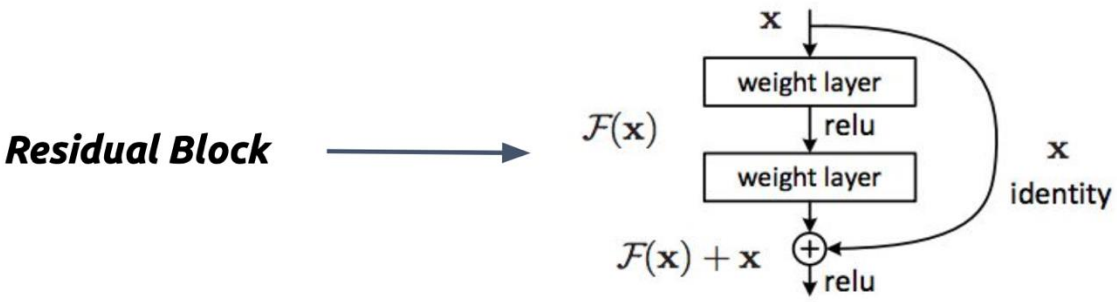
Convolutions provide spatial reasoning capabilities by **breaking symmetries** between inputs (not all pixels are connected to all hidden units).

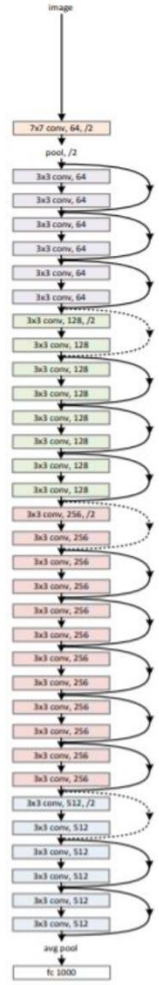
This means that the **spatial layout** of the original image **matters** for convolutional neural networks

More Complicated Networks

ResNet:

Lots of layers, tons of learnable parameters
Avoids Vanishing Gradient problem





K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

Recap

Convolutions provide spatial reasoning capabilities by **breaking symmetries** between inputs (not all pixels are connected to all hidden units).

This means that the **spatial layout** of the original image **matters** for convolutional neural networks

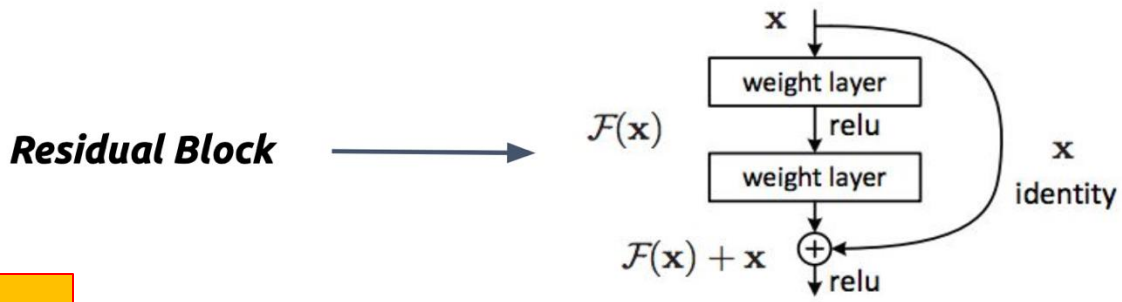
For a new data type (images), we introduced a new network architecture (convolutions) to fix an issue with MLPs.

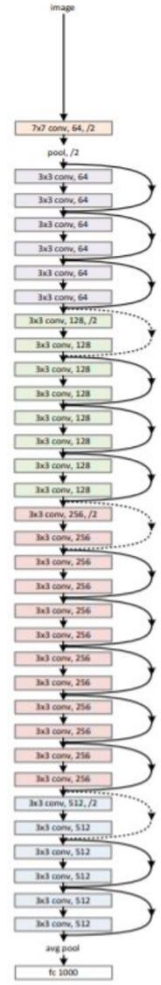
More Complicated Networks

ResNet:

Lots of layers, tons of learnable parameters

Avoids Vanishing Gradient problem





K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

New data type: sequences

- Audio

- DNA

- Stock market

- Weather

New data type: sequences

- Audio

- DNA

- Stock market

- Weather

What is the data property here that we could leverage?

Natural Language

“language that has developed naturally in use”

Natural Language

“language that has developed naturally in use”

Compare to *constructed* or *formal* language

- code: `for i in range(50):`
- math: `52 + 94 = 147`
- logic: `A ^ B -> C` (if A and B, then C)

Natural Language

In this class: **sequence of *words***

"They went to the grocery store and bought bread, peanut butter, and jam."

Natural Language: Prediction tasks?

Input: X

I do not want sour cream in my burrito

Function: f

Output: Y

No quiero crema agrea en mi burrito

Natural Language: Prediction tasks?

Example of prediction?

Input: X

I do not want sour cream in my burrito

Function: f

Output: Y

No quiero crema agrea en mi burrito

Natural Language: Prediction tasks?

Example of classification?

Input: X

"The story telling was erratic and, at times, slow"

"Loved the diverse cast of this movie"

Function: f

Output: Y

"Good review?"

Natural Language: Prediction tasks?

Example of prediction?

"They went to the grocery store and bought... **bread?**

milk?

rock?

Generating artificial sentences: Here each word is a discrete unit; predicting the next part of the sequence means predicting words

Language models

Definition: Probability distribution over strings in a language.

Exponentially-many strings means each string has very low probability

Relative probabilities are meaningful:

P(“*they went to the store*”) >> P(“*butter dancing rock*”)

Language models logic: leverage sentence structure

P(any sequence) is determined by **P(the words in the sequence).**

Language models logic: leverage sentence structure

P(any sequence) is determined by **P(the words in the sequence).**

Said differently, we can represent a sequence as $w_1, w_2, \ldots w_n$, and

$$P(w_1, w_2, \ldots w_n) = P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2) \ * \cdots P(w_n|w_1 \ldots w_{n-1})$$

Language models logic: leverage sentence structure

P(any sequence) is determined by **P(the words in the sequence).**

Said differently, we can represent a sequence as $w_1, w_2, \ldots w_n$, and

$$P(w_1, w_2, \ldots w_n) = P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2) \ * \cdots P(w_n|w_1 \ldots w_{n-1})$$

P(“*they went to the store*”) = P(“*they*”)*P(“*went*”|“*they*”)*P(“*to*”|“*they went*”)* ..

“The probability of a sentence is the product of the probabilities of each word given the previous words”

This is an application of the ***chain rule for probabilities***

Language models: weird & cool!

Model trained on the King James Bible, Structure and Interpretation of Computer Programs, and some of Eric S. Raymond's writings:

- ***The righteous shall inherit the land, and leave it for an inheritance unto the children of Gad according to the number of steps that is linear in b.***
- ***And this I pray, that your love may abound yet more and more like a controlled use of shared memory.***

(King James Programming)

https://kingjamesprogramming.tumblr.com/

Discriminative vs Generative Models

Discriminative vs Generative Models

Discriminitive models learn conditional probabilities $P(Y|X = x)$

Discriminative vs Generative Models

Discriminitive models learn conditional probabilities $P(Y|X = x)$

Given some features, what is the probability of a label?

Discriminative vs Generative Models

Discriminitive models learn conditional probabilities $P(Y|X = x)$

Given some features, what is the probability of a label?

Generative models learn joint probabilities $P(X, Y)$

Discriminative vs Generative Models

Discriminitive models learn conditional probabilities $P(Y|X = x)$

Given some features, what is the probability of a label?

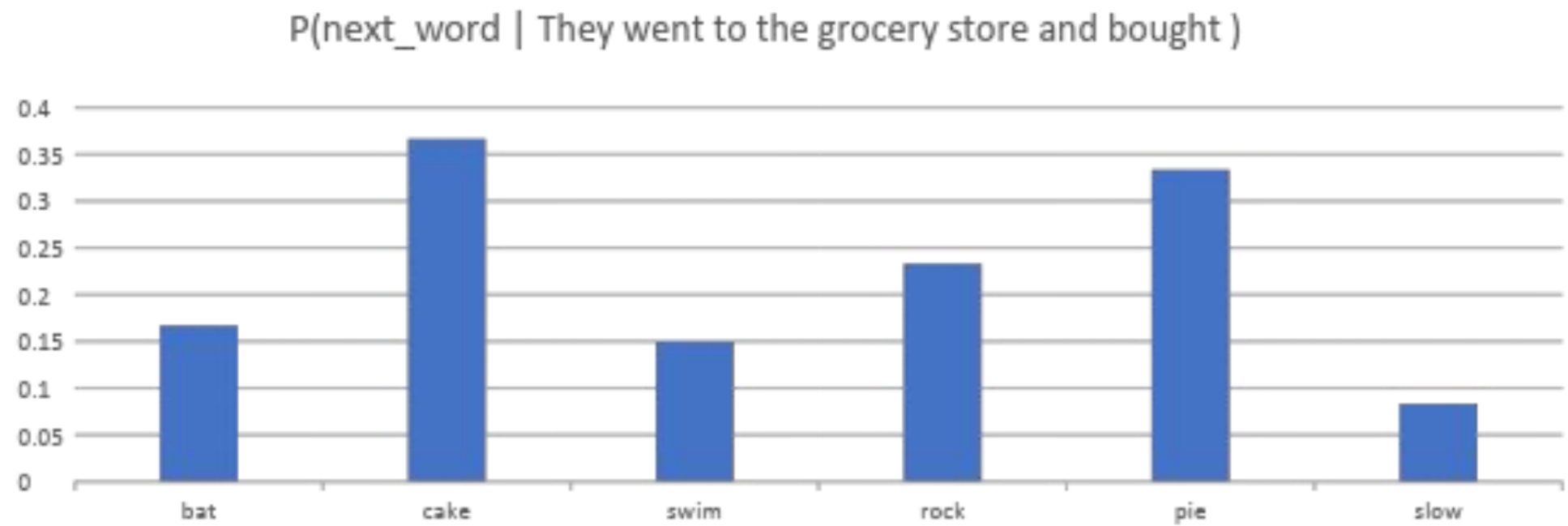
Generative models learn joint probabilities $P(X, Y)$

models how a signal was generated

Language models: the math

At each step, we look at a probability distribution for what the *next* word might be.

They went to the grocery store and bought ..

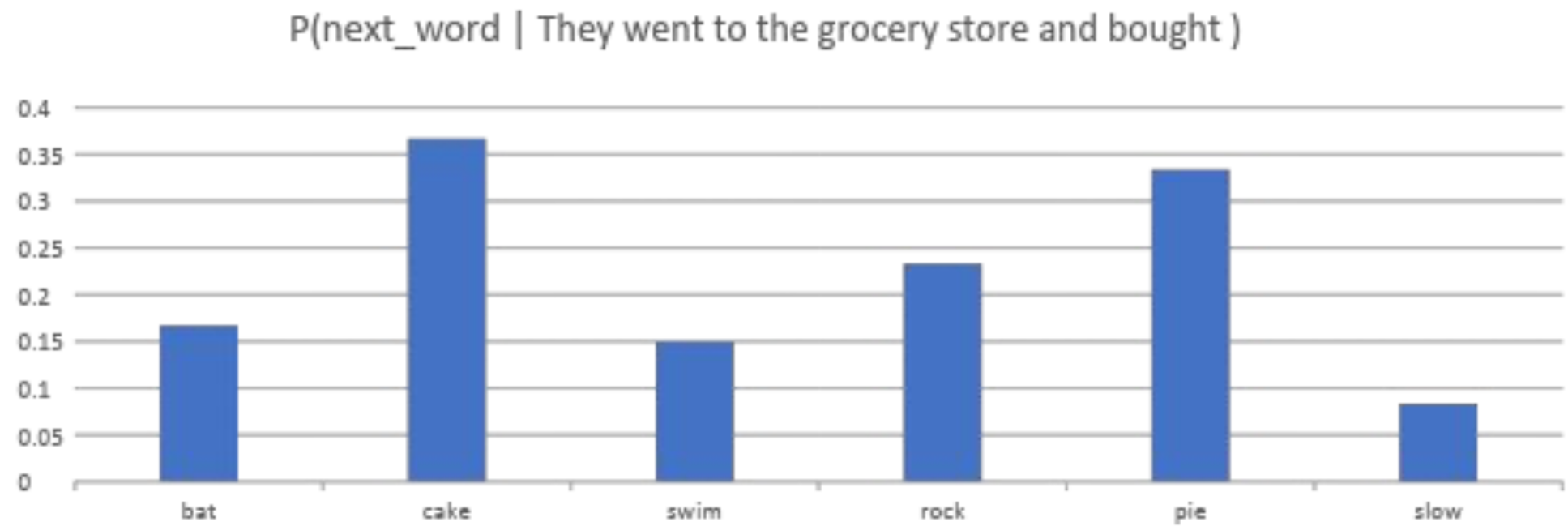


Language models: the math

But first, how do we represent sentence?

At each step, we look at a probability distribution for what the *next* word might be.

They went to the grocery store and bought ..



Natural language: tokenization

“They went to the grocery store and bought bread, peanut butter, and jam.”

```
[“they”, “went”, “to”, “the”,
“grocery”, “store”, “and”,
“bought”, “bread”, “peanut”,
“butter”, “and”, “jam”]
```

Natural language: tokenization

"They went to the grocery store and bought bread, peanut butter, and jam."

- Consistent casing
- Strip punctuation
- One word is one token
- Split on spaces

```
["they", "went", "to", "the",
"grocery", "store", "and",
"bought", "bread", "peanut",
"butter", "and", "jam"]
```

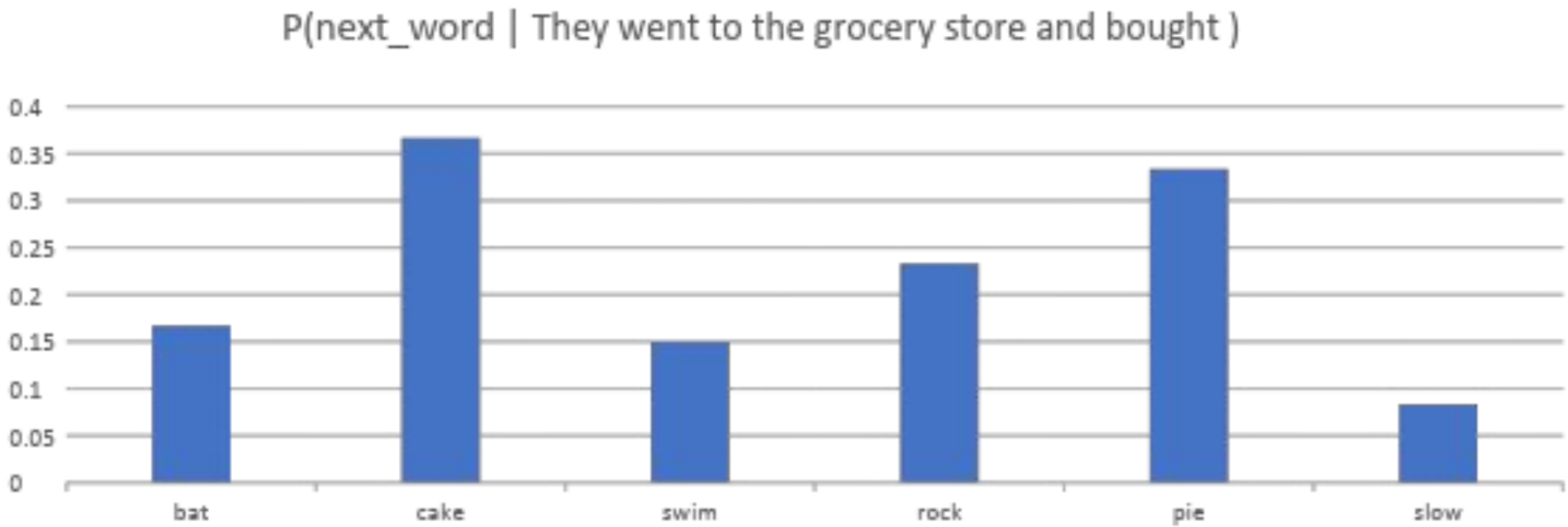
Aside: Tokenization itself can be challenging...

- A lot easier in English than other languages (e.g. Chinese)
  - Chinese is character-based; words & phrases have different character lengths
  - No spaces

Language models: the math

At each step, we look at a probability distribution for what the *next* word might be.

They went to the grocery store and bought ..

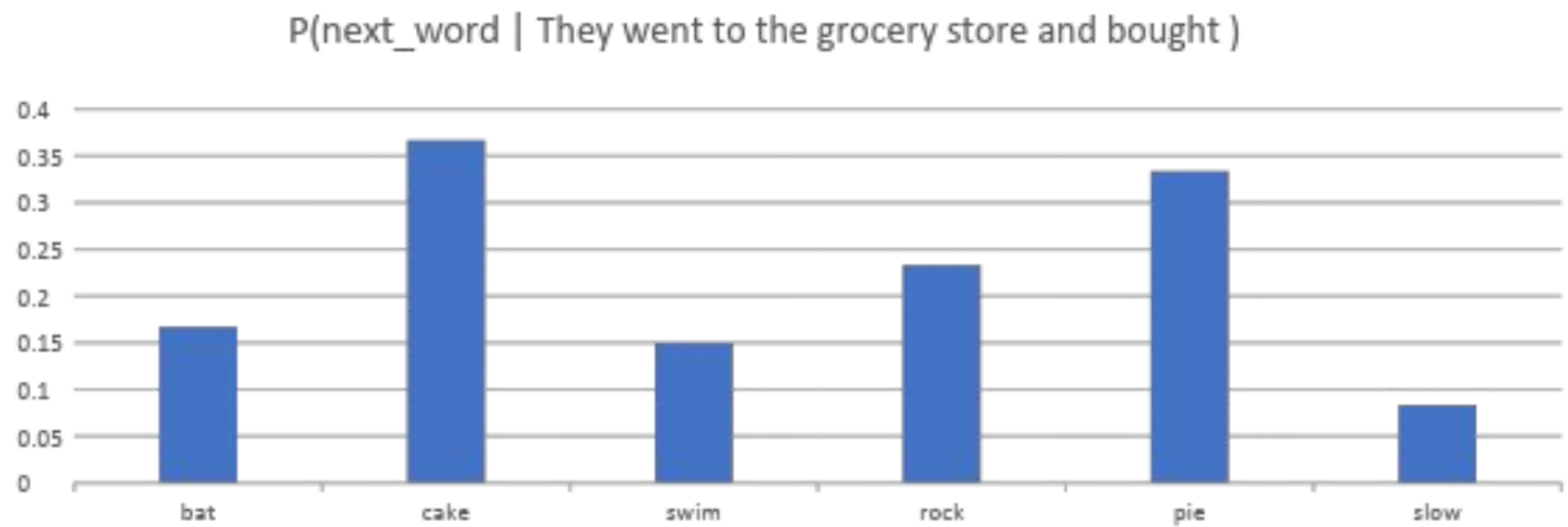


Language models: the math

How do we know which words to calculate probabilities for?

At each step, we look at a probability distribution for what the *next* word might be.

They went to the grocery store and bought ..



Vocabularies: Defining a finite set of words

Vocabularies: the set of all words “known” to the model

Why?

- We need a finite set of words in order to define a discrete distribution over it.

Vocabularies: Defining a finite set of words

Vocabularies: the set of all words “known” to the model

Why?

- We need a finite set of words in order to define a discrete distribution over it.

How?

- Choose a hyperparameter **vocab_size** for how many words the model should know
- Keep only the **vocab_size** with most frequent words – replace everything else with “**UNK**”

Vocabularies: how

- Original sentence:
  - ***"They galloped to the Ratty for dinner, and ate exactly seventy-three waffle fries and chocolate peamilk."***

Vocabularies: how

- Original sentence:
  - *“They galloped to the Ratty for dinner, and ate exactly seventy-three waffle fries and chocolate peamilk.”*
- Tokenized:
  - `[“they”, “galloped”, “to”, “the”, “ratty”, “for”, “dinner”, “and”, “ate”, “exactly”, “seventy-three”, “waffle”, “fries”, “and”, “chocolate”, “peamilk”]`

Vocabularies: how

- Original sentence:
  - *“They galloped to the Ratty for dinner, and ate exactly seventy-three waffle fries and chocolate peamilk.”*
- Tokenized:
  - `[“they”, “galloped”, “to”, “the”, “ratty”, “for”, “dinner”, “and”, “ate”, “exactly”, “seventy-three”, “waffle”, “fries”, “and”, “chocolate”, “peamilk”]`

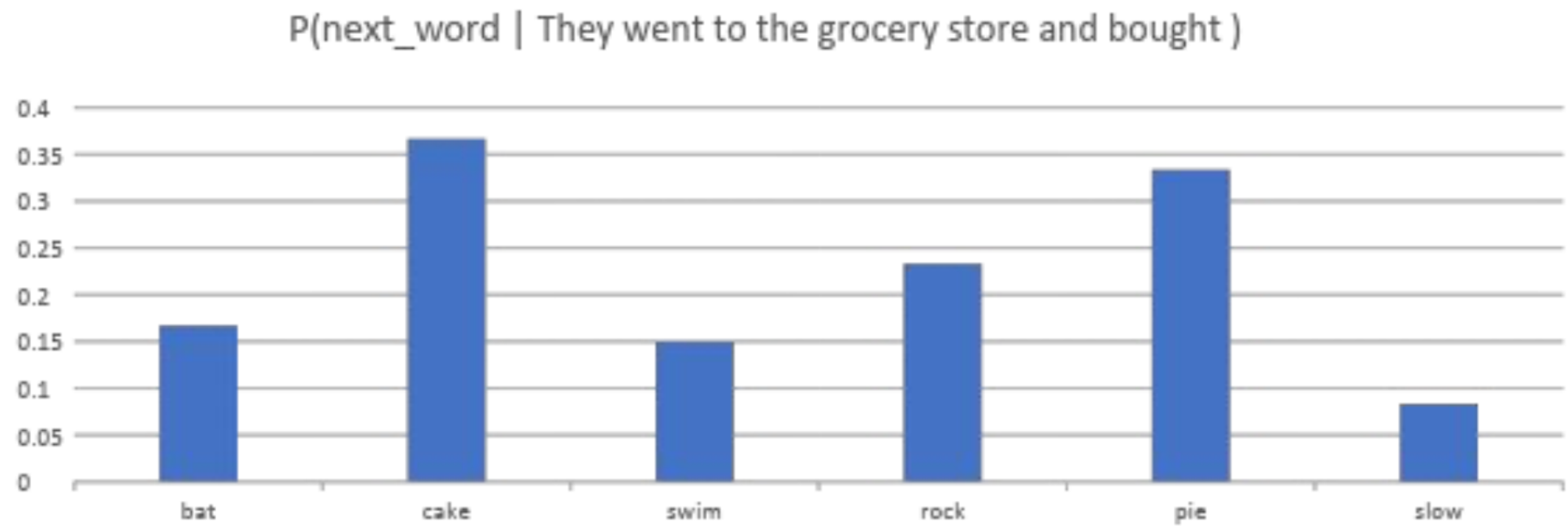
Vocabularies: how

- Original sentence:
  - *"They galloped to the Ratty for dinner, and ate exactly seventy-three waffle fries and chocolate peamilk."*
- Tokenized:
  - `["they", "galloped", "to", "the", "ratty", "for", "dinner", "and", "ate", "exactly", "seventy-three", "waffle", "fries", "and", "chocolate", "peamilk"]`
- UNKed:
  - `["they", "UNK", "to", "the", "UNK", "for", "dinner", "and", "ate", "exactly", "UNK", "waffle", "fries", "and", "chocolate", "UNK"]`

Language models: the math

At each step, we look at a probability distribution for what the *next* word might be.

They went to the grocery store and bought ..

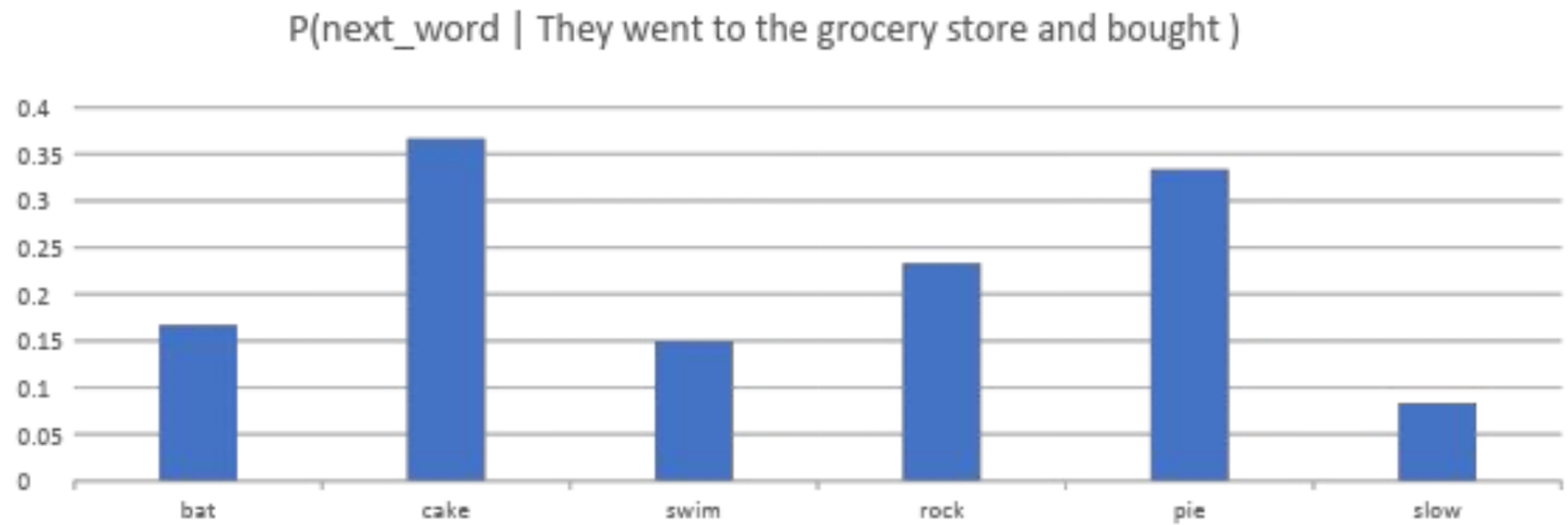


Language models: the math

How to calculate the probability for words in our vocabulary?

At each step, we look at a probability distribution for what the *next* word might be.

They went to the grocery store and bought ..



LM implementation: counting

- Goal: predict next word given a preceding sequence
  - $P(\boldsymbol{word_n}|\, word_1, word_2, \ldots word_{n-1}) = \frac{Count(word_1, word_2, \ldots word_{n-1}, \boldsymbol{word_n})}{Count(word_1, word_2, \ldots word_{n-1})}$

LM implementation: counting

- Goal: predict next word given a preceding sequence
  - $P(\boldsymbol{word_n}|\, word_1, word_2, \ldots word_{n-1}) = \frac{Count(word_1, word_2, \ldots word_{n-1}, \boldsymbol{word_n})}{Count(word_1, word_2, \ldots word_{n-1})}$
- Example task: predict the next word
  - ***he danced*** ___

LM implementation: counting

- Goal: predict next word given a preceding sequence
  - $P(\boldsymbol{word_n}|\, word_1, word_2, \ldots word_{n-1}) = \frac{Count(word_1, word_2, \ldots word_{n-1}, \boldsymbol{word_n})}{Count(word_1, word_2, \ldots word_{n-1})}$
- Example task: predict the next word
  - ***he danced*** ___
- Strategy: iterate through all words in vocabulary, and calculate

$$\frac{Count(he\ danced\ <word>)}{Count(he\ danced)}$$ for each word

LM implementation: counting

- Our training sentences were:
  - *"She danced happily"*
  - *"They sang beautifully"*
  - *"He danced energetically"*
  - *"He sang happily"*
  - *"She danced gracefully"*

- *"He danced _ _ _"*

- *"He danced happily"*

$$\frac{Count(he\ danced\ <word>)}{Count(he\ danced)}$$

LM implementation: counting

- Our training sentences were:
  - *"She danced happily"*
  - *"They sang beautifully"*
  - *"He danced energetically"*
  - *"He sang happily"*
  - *"She danced gracefully"*

$$\frac{Count(he\ danced\ <word>)}{Count(he\ danced)}$$

- *"He danced _ _ _"*

- *"He danced happily"* Has 0 probability

LM implementation: counting

- Our training sentences were:
  - *"She danced happily"*
  - *"They sang beautifully"*
  - *"He danced energetically"*
  - *"He sang happily"*
  - *"She danced gracefully"*
- *"He danced _ _ _"*
- *"He danced happily"* Has 0 probability

$$\frac{Count(he\ danced\ <word>)}{Count(he\ danced)}$$

Why doesn't this work?

LM implementation: counting

- Our training sentences were:
  - *"She danced happily"*
  - *"They sang beautifully"*
  - *"He danced energetically"*
  - *"He sang happily"*
  - *"She danced gracefully"*
- *"He danced _ _ _"*
- *"He danced happily"* Has 0 probability

$$\frac{Count(he\ danced\ <word>)}{Count(he\ danced)}$$

Why doesn't this work?

This strategy depends on having instances of sentence prefixes.

LM implementation: N-gram counting

Improvement: **N-gram** model – only look at **N** words at a time

LM implementation: N-gram counting

Improvement: **N-gram** model – only look at **N** words at a time

(in this case, **bi**grams look at **2** words at a time)

- *"She danced happily"*
- *"They sang beautifully"*
- *"He danced energetically"*
- *"He sang happily"*
- *"She danced gracefully"*

LM implementation: N-gram counting

Improvement: **N-gram** model – only look at **N** words at a time

(in this case, **bi**grams look at **2** words at a time)

- *"danced happily"*
- *"sang beautifully"*
- *"danced energetically"*
- *"sang happily"*
- *"danced gracefully"*

"He danced happily" now has 1/3 probability!

But what if the answer was *"He danced beautifully"* ?

LM implementation

Problem: it's impossible for the training set to have *every possible valid* sequence of words!

Let's try to learn a better **numerical** representation

LM implementation

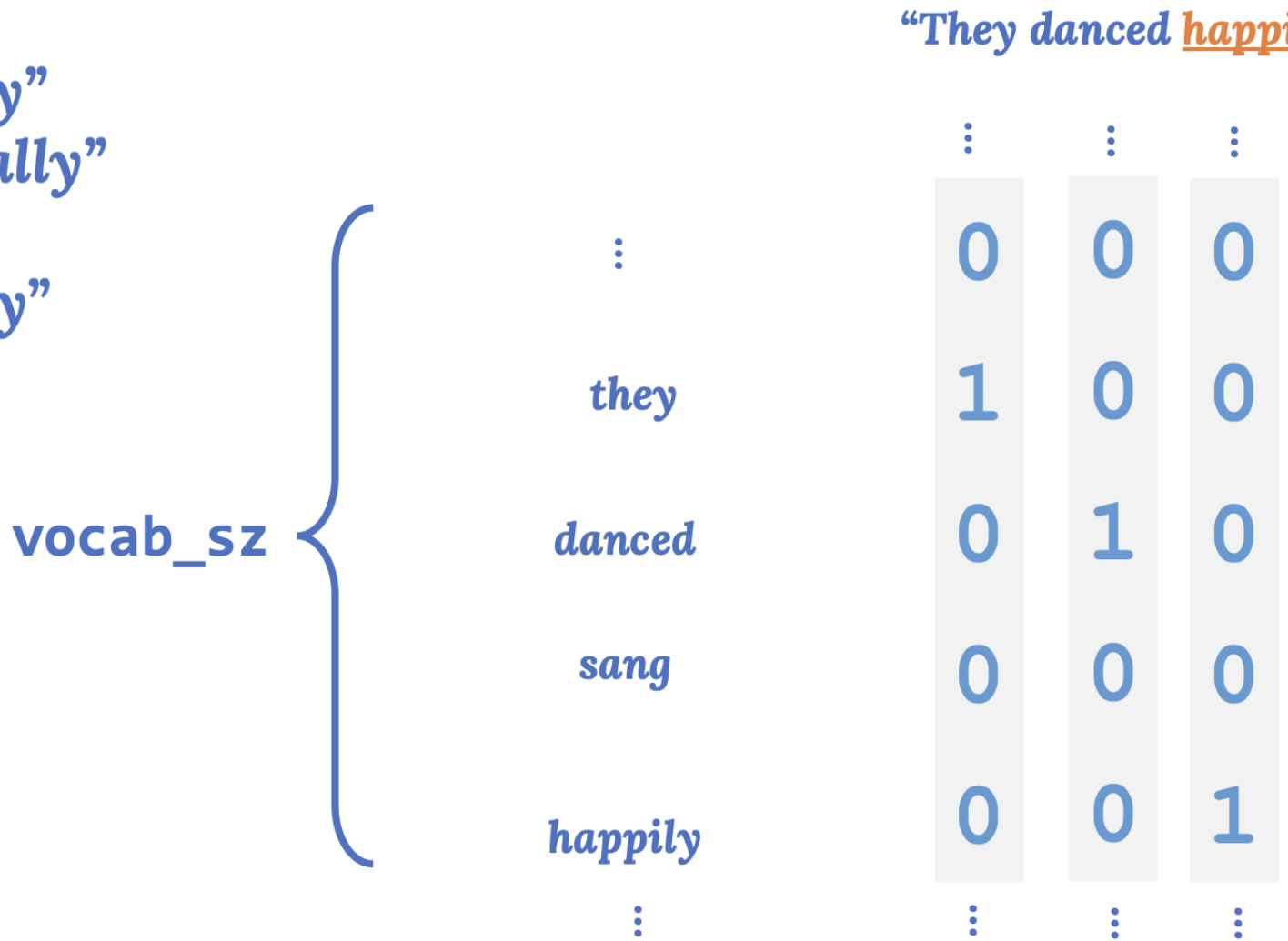
Problem: it's impossible for the training set to have *every possible valid* sequence of words!

Let's try to learn a better **numerical** representation

What is the simplest thing you can think of?

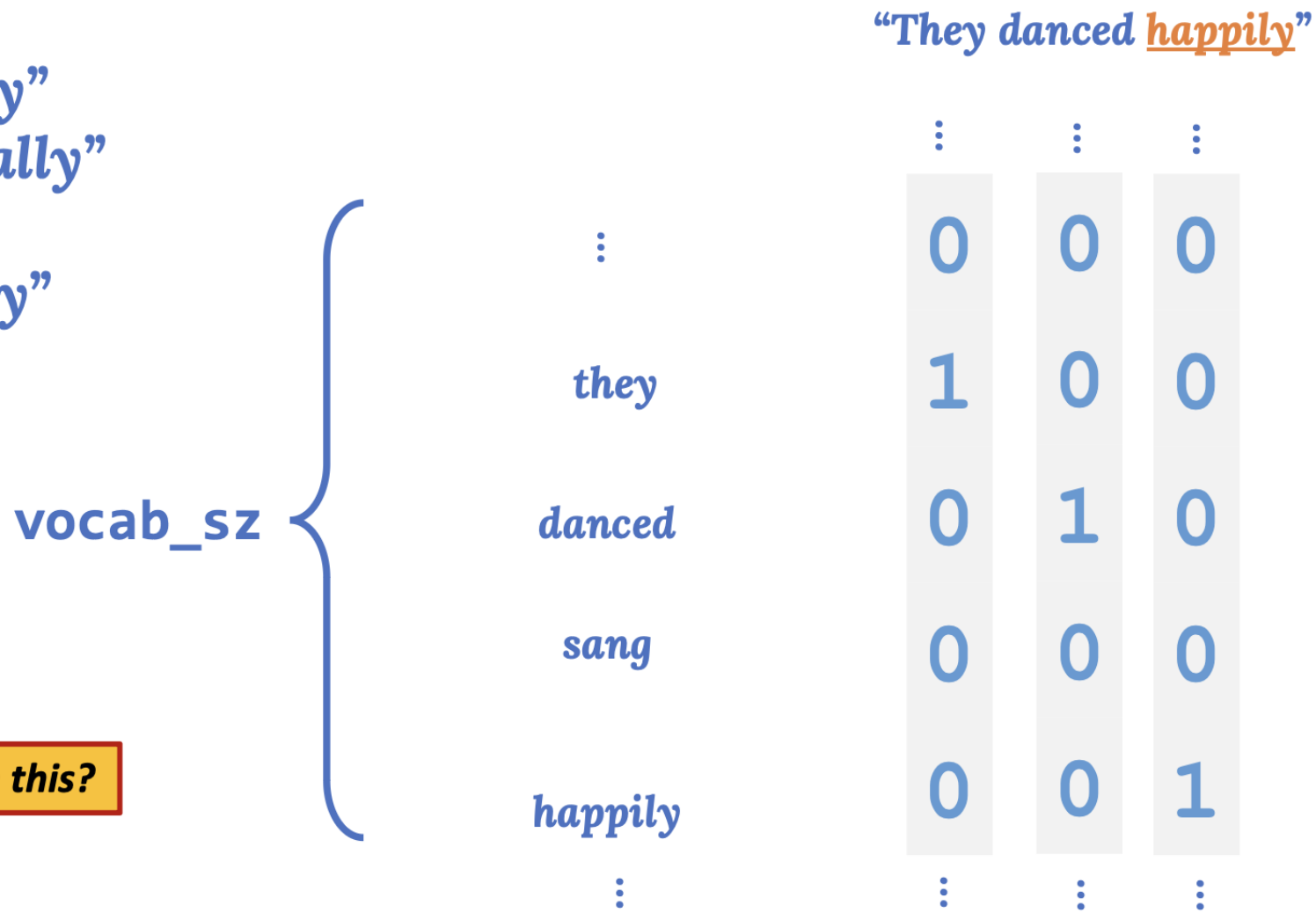
LM implementation: Simple approach

- *"She danced happily"*
- *"They sang beautifully"*
- *"He danced energetically"*
- *"He sang happily"*
- *"She danced gracefully"*



LM implementation: Simple approach

- *"She danced happily"*
- *"They sang beautifully"*
- *"He danced energetically"*
- *"He sang happily"*
- *"She danced gracefully"*



Any potential issues with this?

LM implementation

Problem: it's impossible for the training set to have *every possible valid* sequence of words!

Can we learn a better numerical representation **which associates *related* words with one another**?

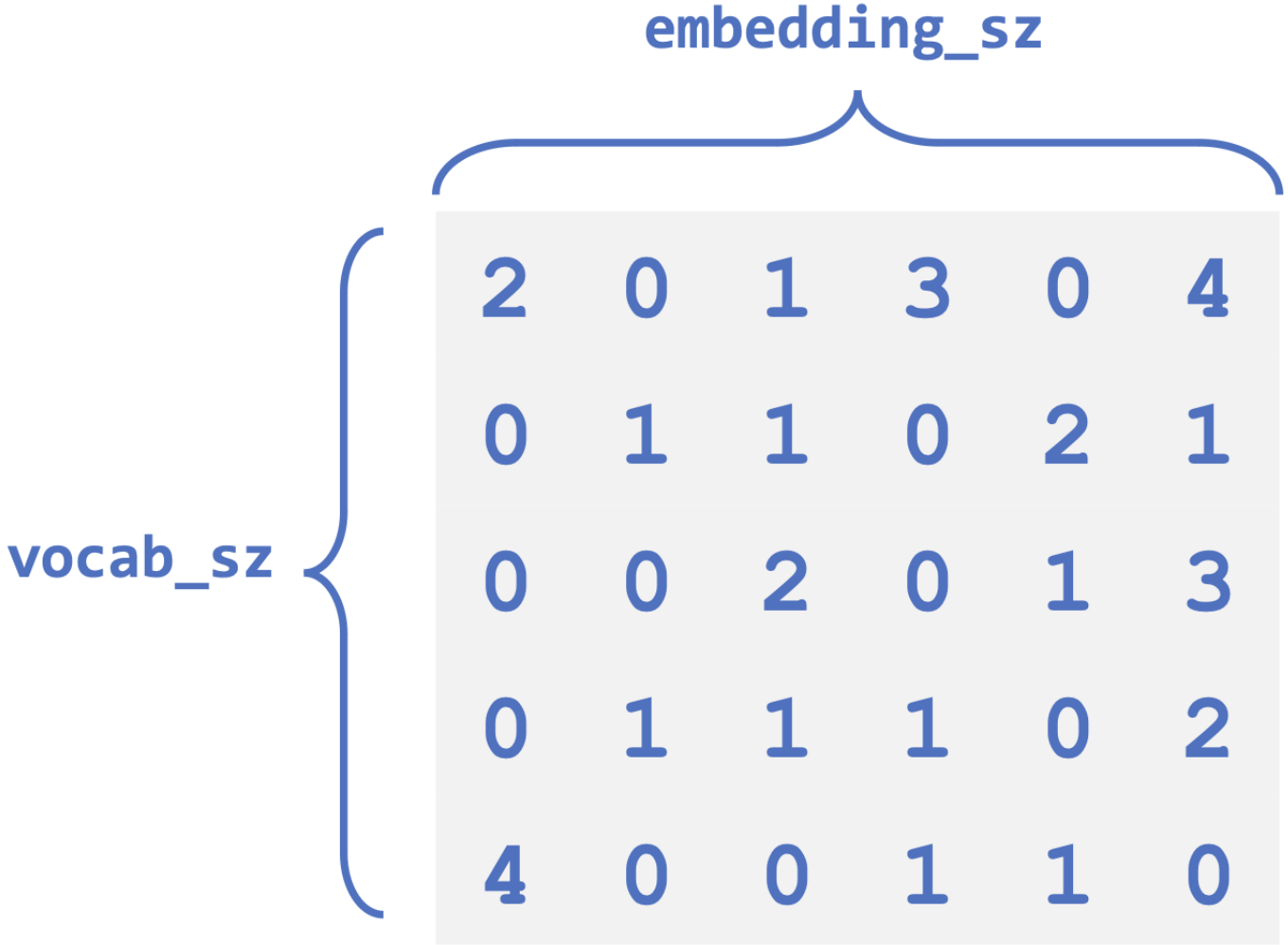
Embedding matrix

Embedding matrix

vocab_sz

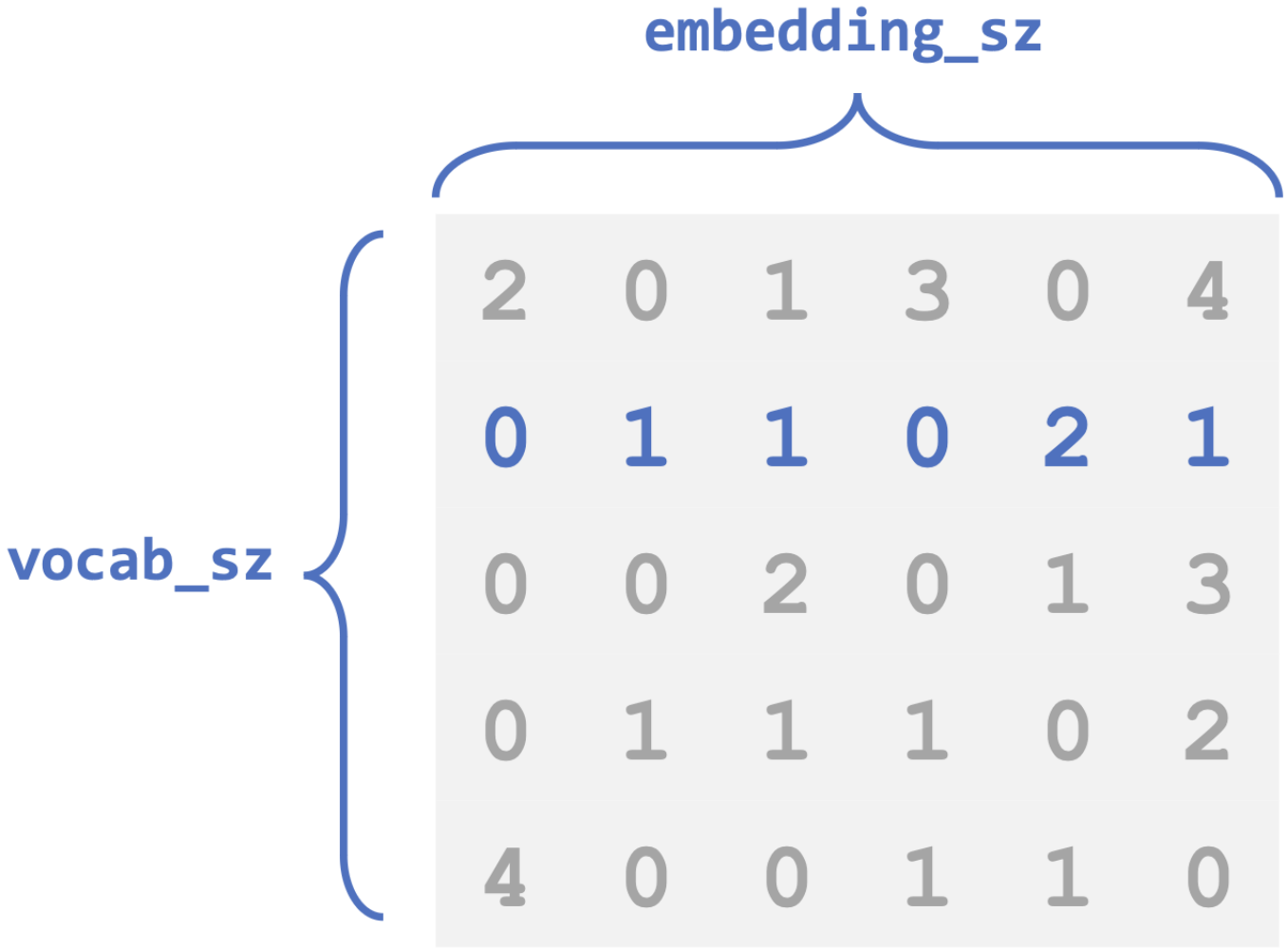
⋮						
they	2	0	1	3	0	4
danced	0	1	1	0	2	1
sang	0	0	2	0	1	3
happily	0	1	1	1	0	2
gleefully	4	0	0	1	1	0
⋮						

Embedding matrix



- 2d matrix: **vocab_sz x embedding_sz**

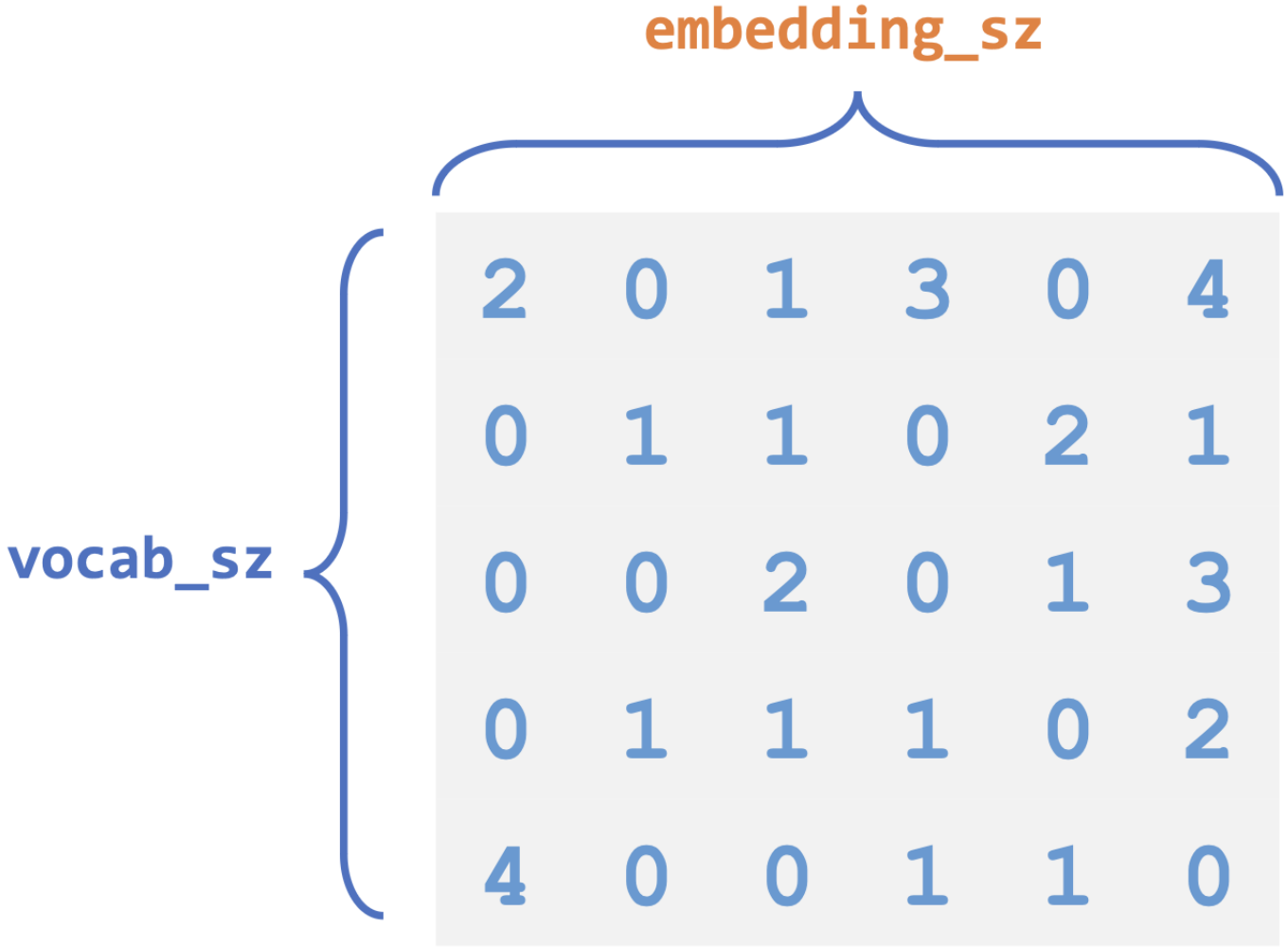
Embedding matrix



- 2d matrix: **vocab_sz x embedding_sz**
- each word corresponds to an index, or word ID – hence the **vocab_sz** dimension

Embedding matrix

How to build this embedding matrix?



- 2d matrix: **vocab_sz x embedding_sz**
- each word corresponds to an index, or word ID – hence the **vocab_sz** dimension
- **embedding_sz** is a hyperparameter

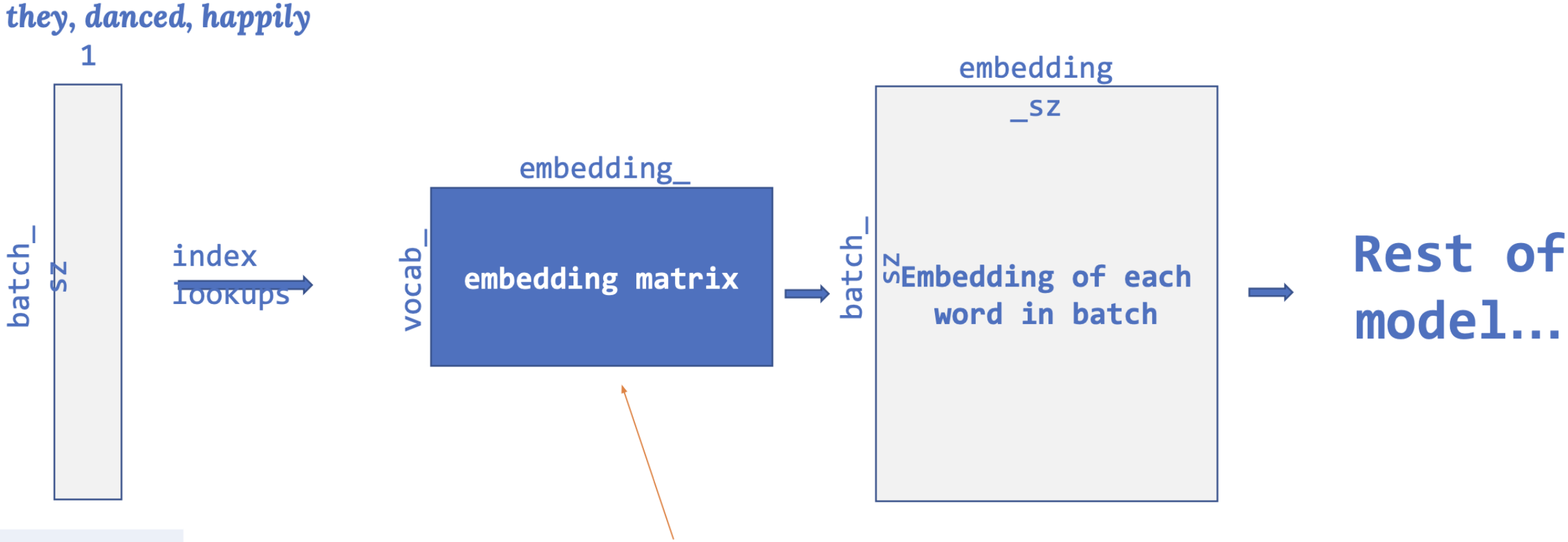
LM implementation: deep learning

Deep learning helps solve this!

We can learn an ***embedding matrix*** that associates *related* words with one another for solving a prediction task.

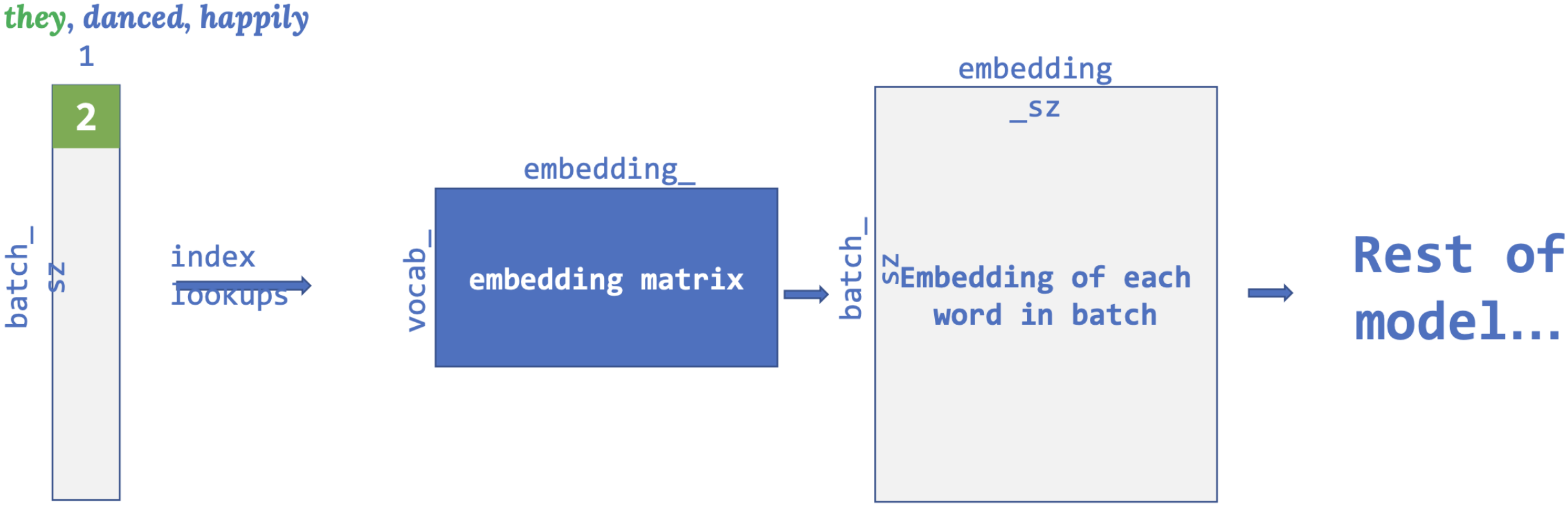
Using the Embedding Matrix in a Network

If you want to input a [batch of] words into a neural net, this is how:



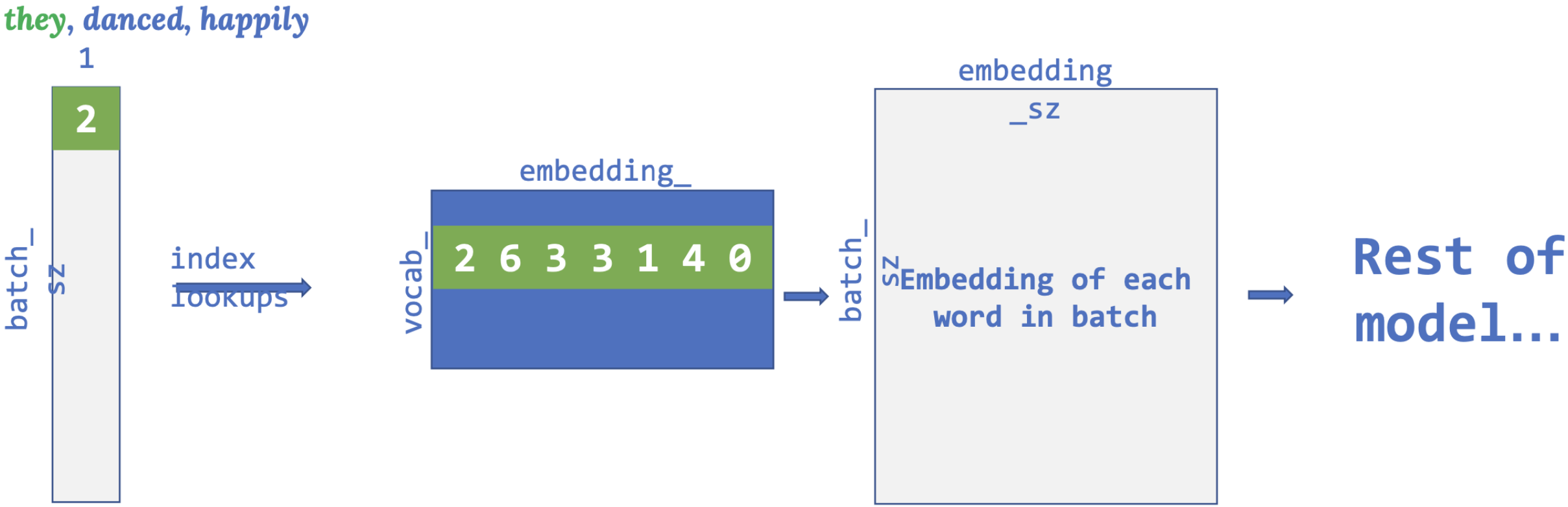
Using the Embedding Matrix in a Network

Let's look at the 0th word in this batch; its ID in the vocab is 2.



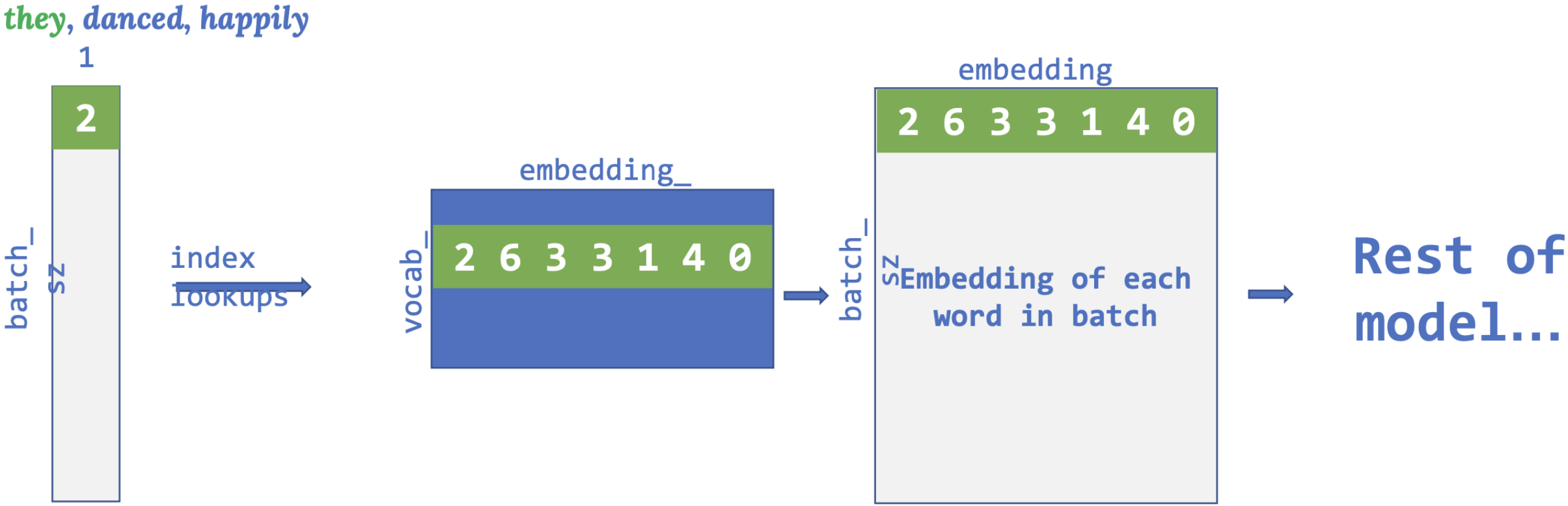
Using the Embedding Matrix in a Network

So we look at row 2 of the embedding matrix.



Using the Embedding Matrix in a Network

We can then pull out this embedding so we can use it in the rest of the model!

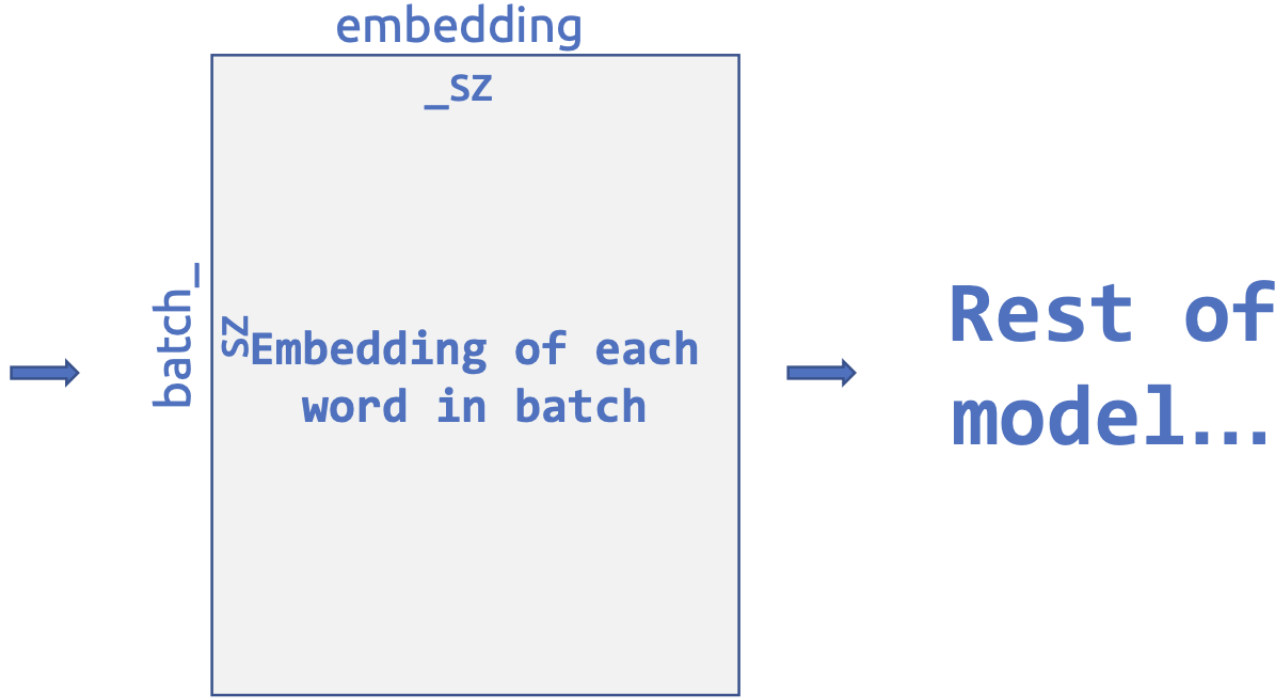


Using the Embedding Matrix in a Network

In tensorflow, we can use

`tf.nn.embedding_lookup`

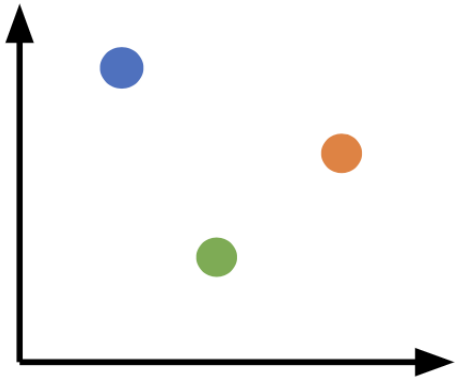
which takes in an embedding matrix and a list of indices, and returns the embedding corresponding to each index.



What does the embedding matrix represent?

- Each row in the matrix can be viewed as a vector in vector space

Example 2-D vector space:



Vocab size: 3

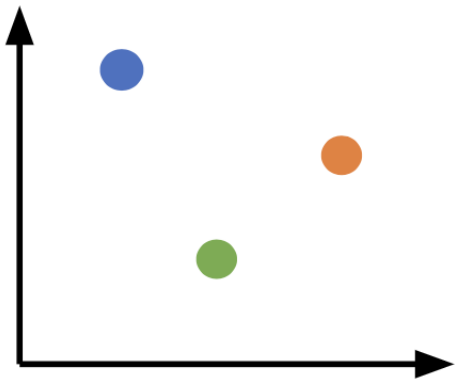
Embed size: 2

1	3
2	1
3	2

What does the embedding matrix represent?

- Each row in the matrix can be viewed as a vector in vector space
- "Embedding": We're **embedding** a non-Euclidian entity [a word] into Euclidian space

Example 2-D vector space:



Vocab size: 3

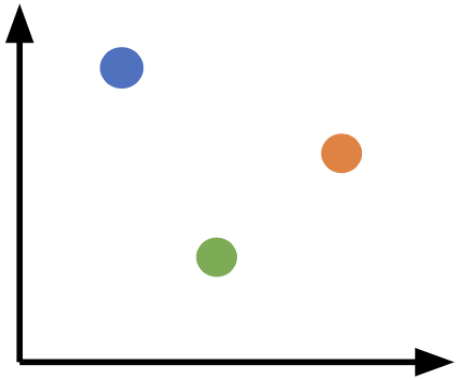
Embed size: 2

1	3
2	1
3	2

What does the embedding matrix represent?

- Each row in the matrix can be viewed as a vector in vector space
- "Embedding": We're **embedding** a non-Euclidian entity [a word] into Euclidian space
- Each row represents the "embedding" for a single word

Example 2-D vector space:



Vocab size: 3

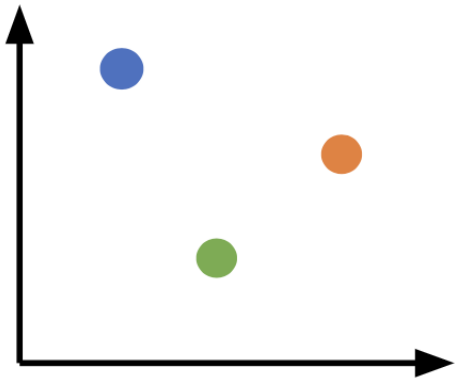
Embed size: 2

1	3
2	1
3	2

What does the embedding matrix represent?

- Each row in the matrix can be viewed as a vector in vector space
- "Embedding": We're **embedding** a non-Euclidian entity [a word] into Euclidian space
- Each row represents the "embedding" for a single word
- This has pretty remarkable properties!

Example 2-D vector space:

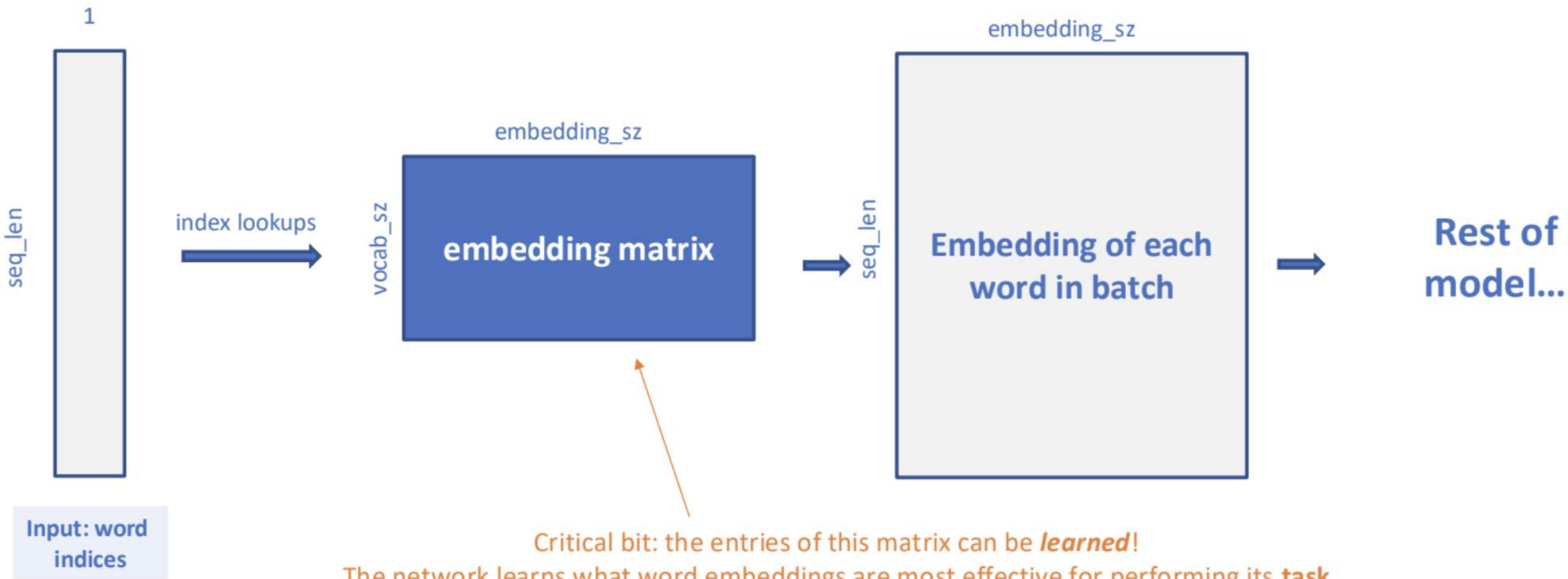


Vocab size: 3

Embed size: 2

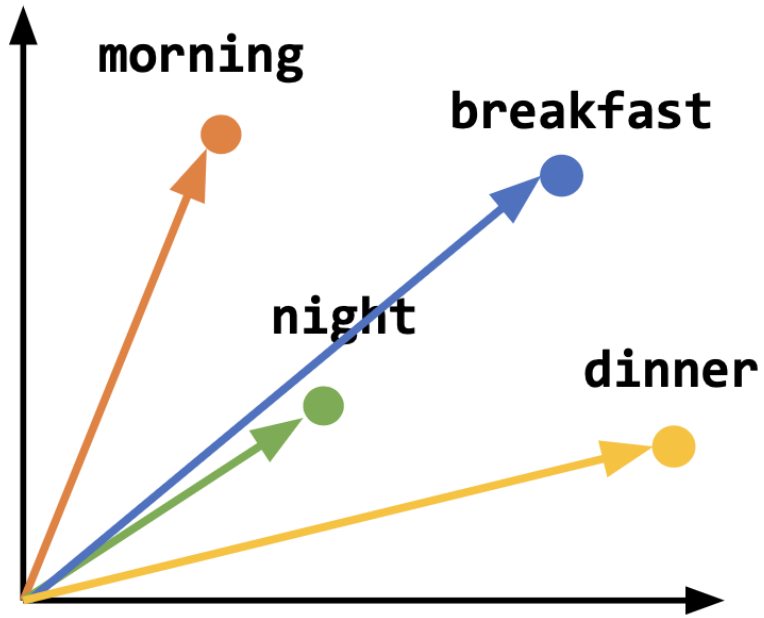
1	3
2	1
3	2

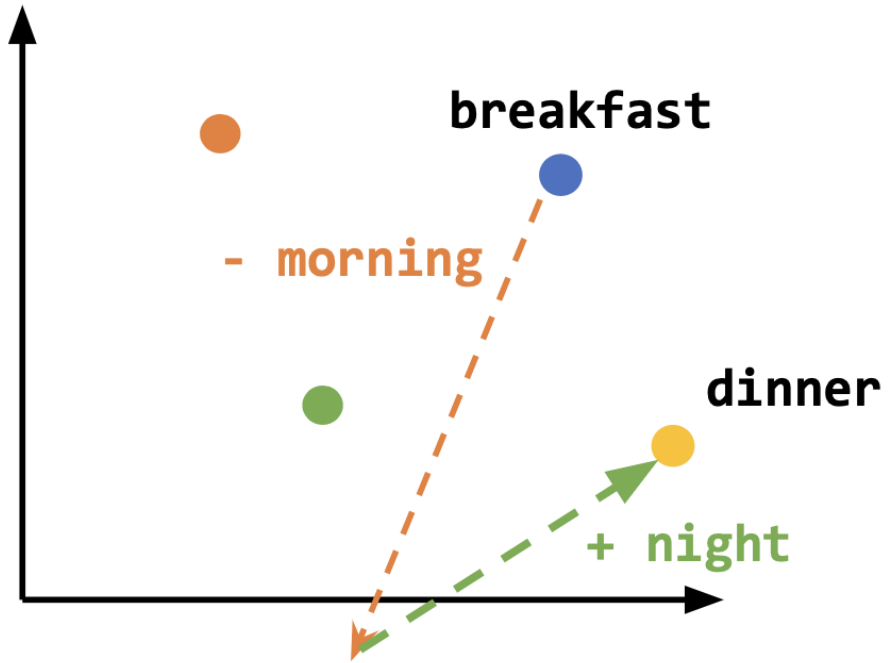
Using the Embedding Matrix in a Network



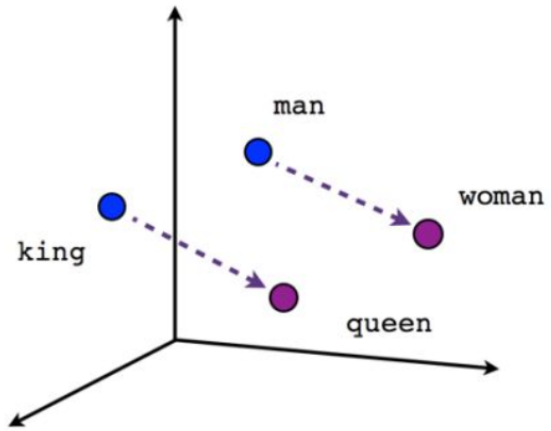
Vector arithmetic in the embedding matrix

Demo here: https://turbomaze.github.io/word2vecjson/





More ‘semantic directions’ in embedding space

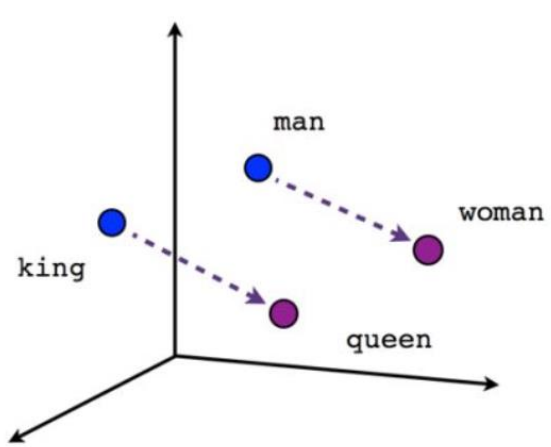


Male-Female

E(queen) – E(king) ≈
E(woman) – E(man)

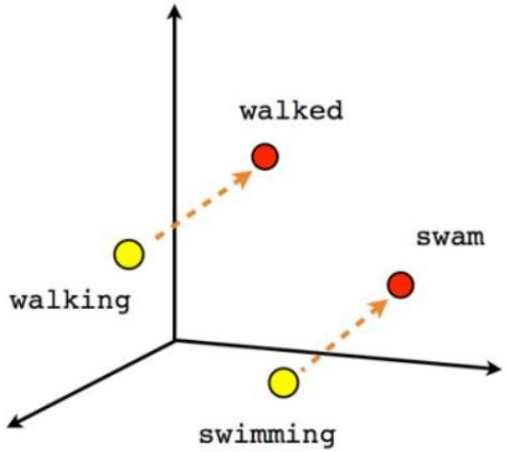
Semantic: relating to meaning in language

More ‘semantic directions’ in embedding space



Male-Female

E(queen) – E(king) ≈
E(woman) – E(man)



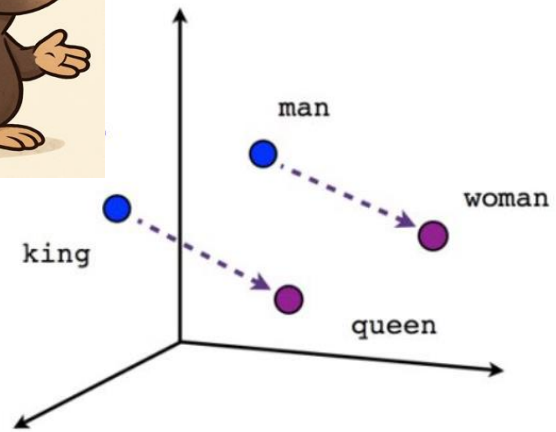
Verb tense

E(walked) – E(walking) ≈
E(swam) – E(swimming)

Semantic: relating to meaning in language

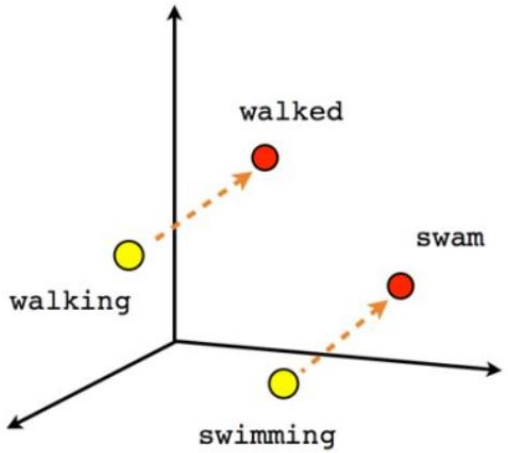
More ‘semantic directions’ in embedding space

Any questions?



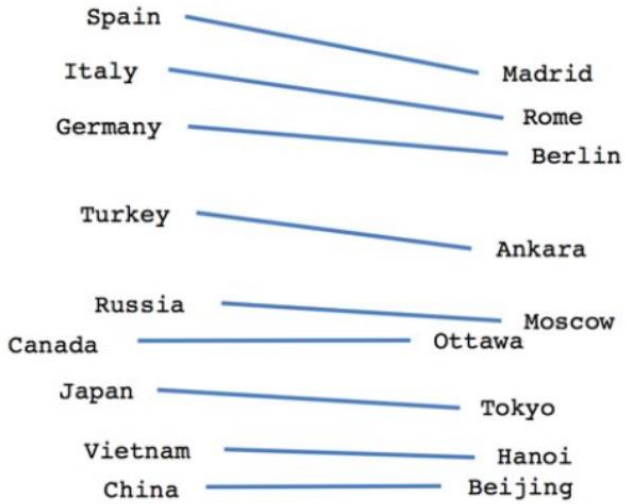
Male-Female

E(queen) – E(king) ≈
E(woman) – E(man)



Verb tense

E(walked) – E(walking) ≈
E(swam) – E(swimming)

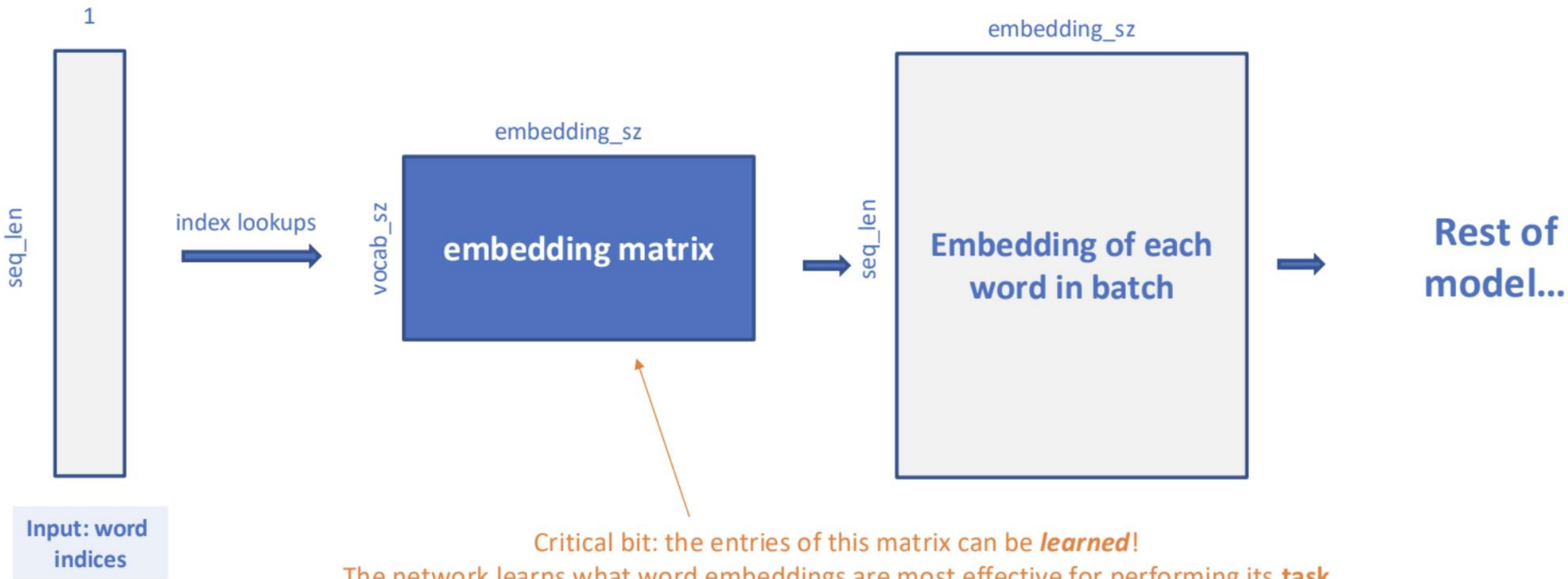


Country-Capital

E(Spain) – E(Madrid) ≈
E(Vietnam) – E(Hanoi)

Semantic: relating to meaning in language

Using the Embedding Matrix in a Network



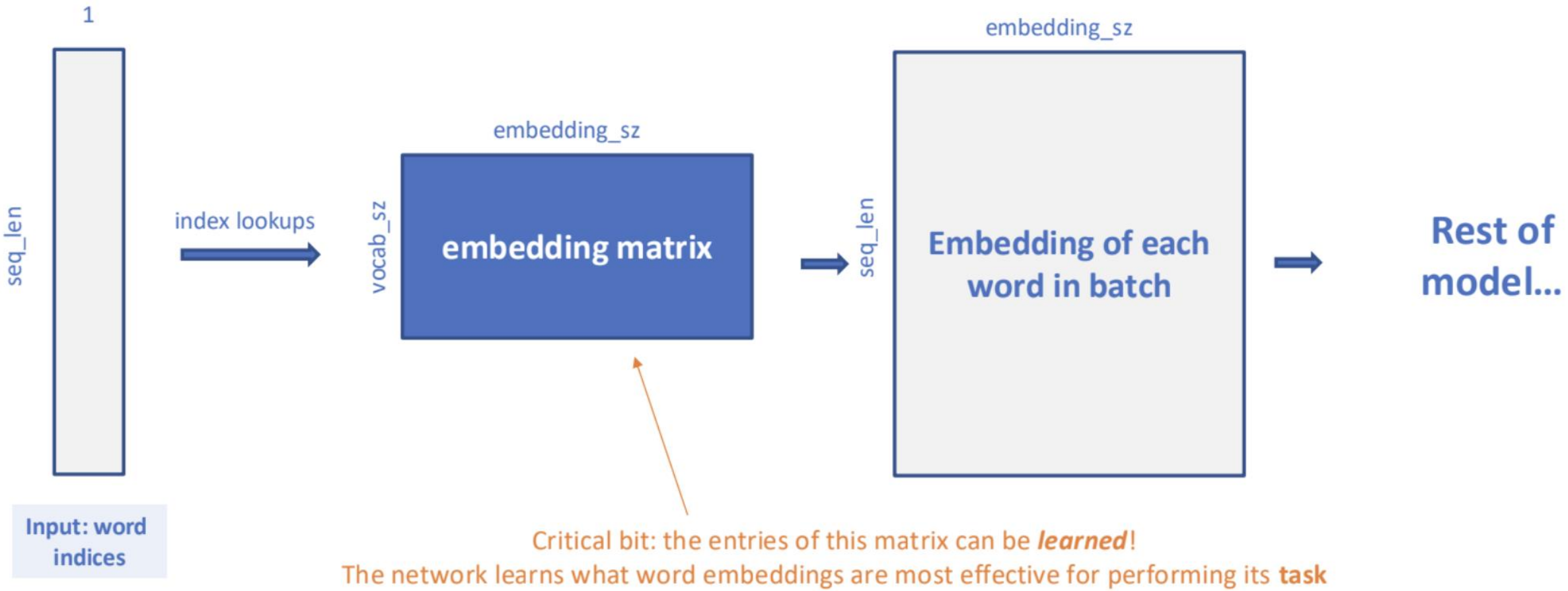
Using the Embedding Matrix in a Network

Say in the middle of training, the model sees:

$P(\text{“}\boldsymbol{happily}\text{”}|\text{”}\textbf{They Danced”}) = \textbf{High}$

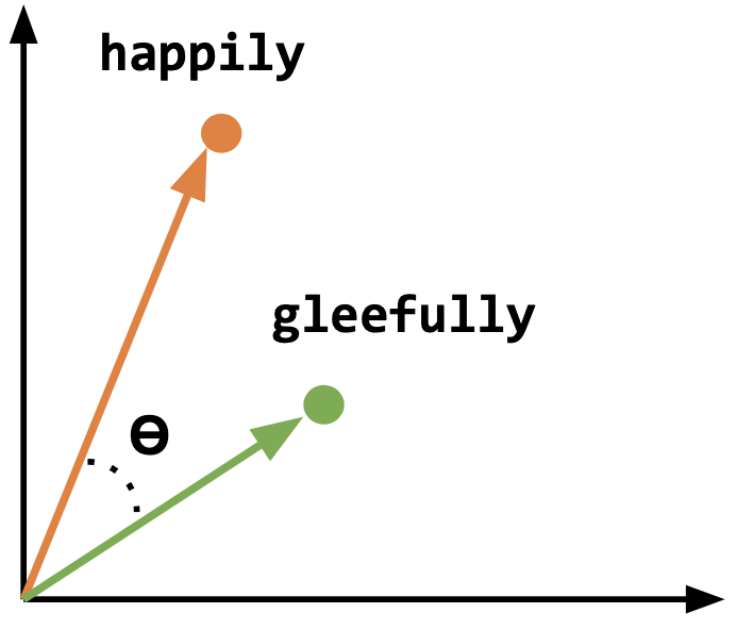
$P(\text{“}\boldsymbol{gleefully}\text{”}|\text{”}\textbf{They Danced”}) = \textbf{Low}$

Then the model sees a lot of “danced gleefully”



Quantifying “similarity”

- $$cosine\ similarity = \cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$



$\cos(0°) = 1$

$\cos(90°) = -0.448$

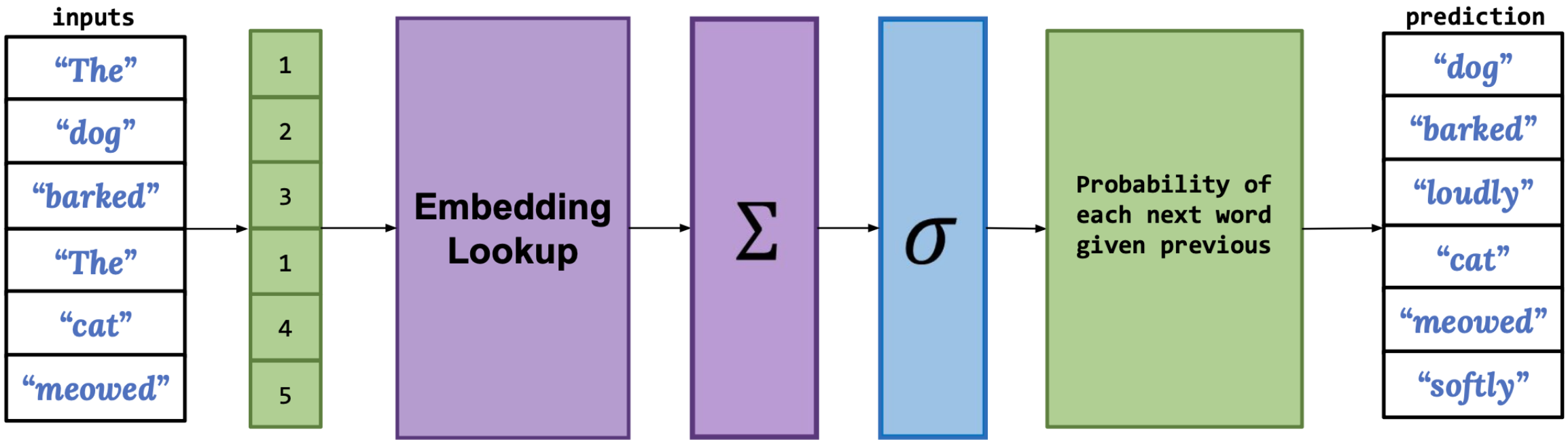
$\cos(180°) = -0.598$

Limitations of the N-gram model

What issues do we run into using feed-forward N-gram models?

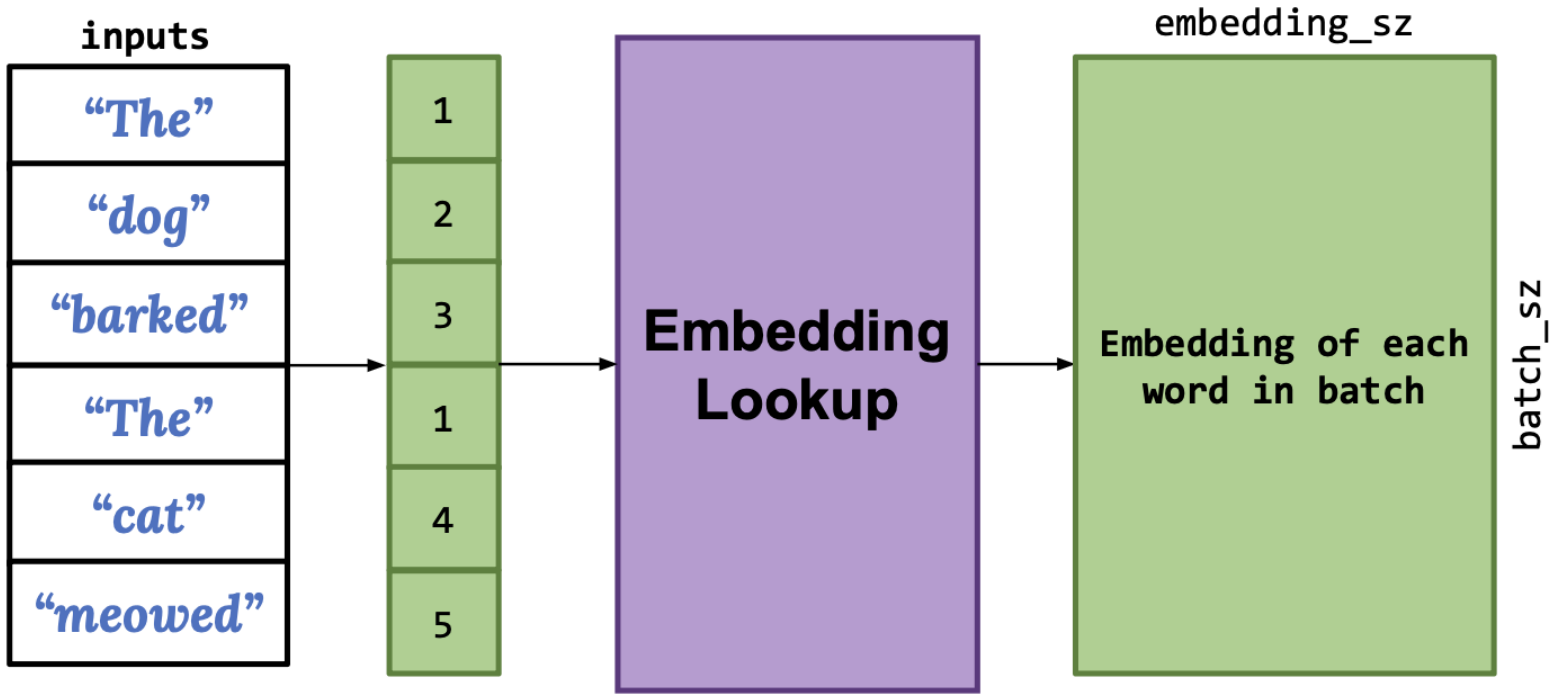
Size of Feed Forward bigram Model

Let's look at bigram model and count the number of weights.



Size of Feed Forward bigram Model

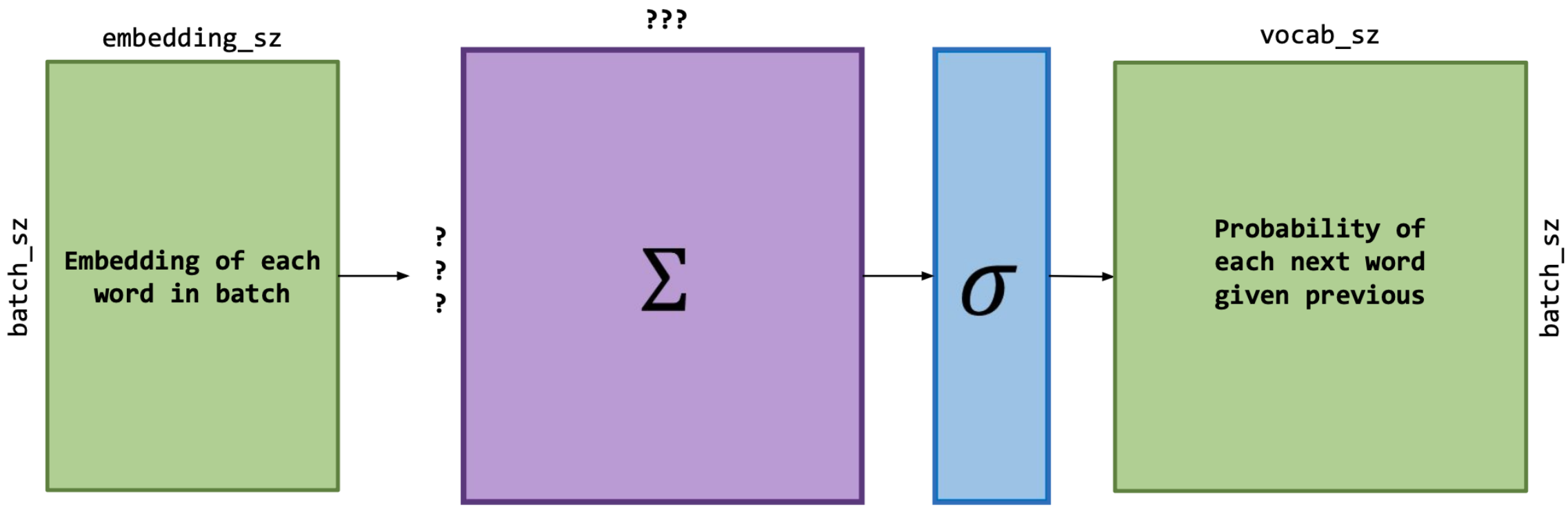
To preform embedding lookup on our entire batch, we just need one embedding matrix of size: `(vocab_sz, embedding_sz)`



Size of Feed Forward bigram Model

What size do we need the linear layer to be in order to map:

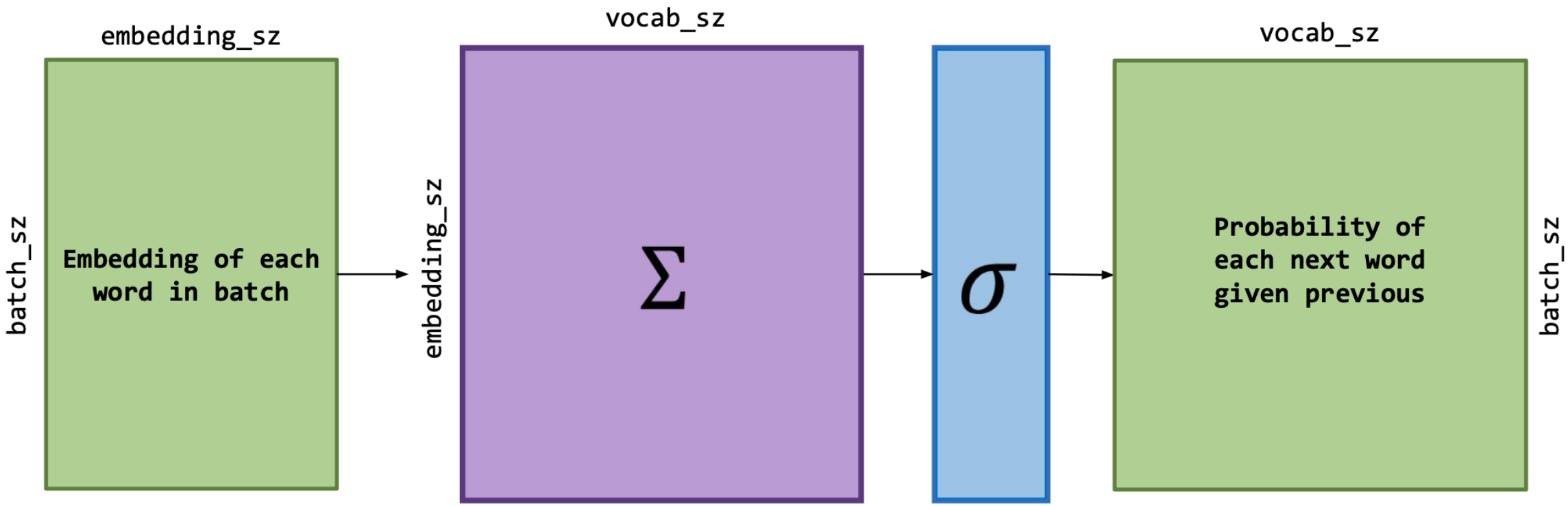
(batch_sz, embedding_sz) × (???, ???) → (batch_sz, vocab_sz)



Size of Feed Forward bigram Model

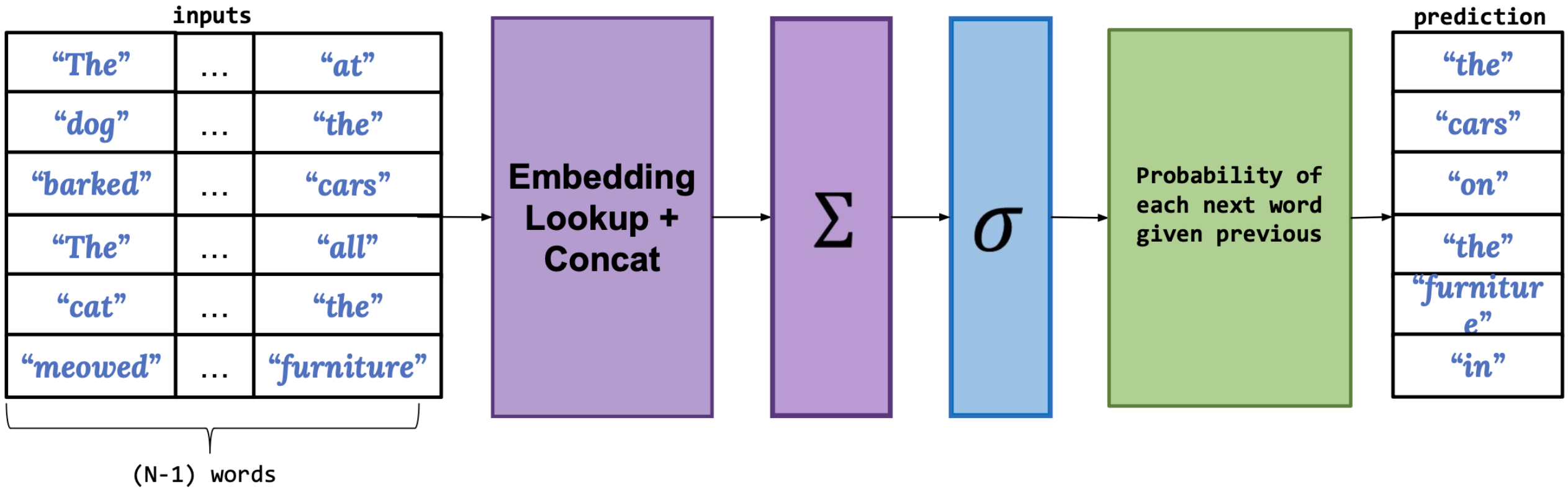
What size do we need the linear layer to be in order to map:

(batch_sz, embedding_sz) × (???, ???) → (batch_sz, vocab_sz)



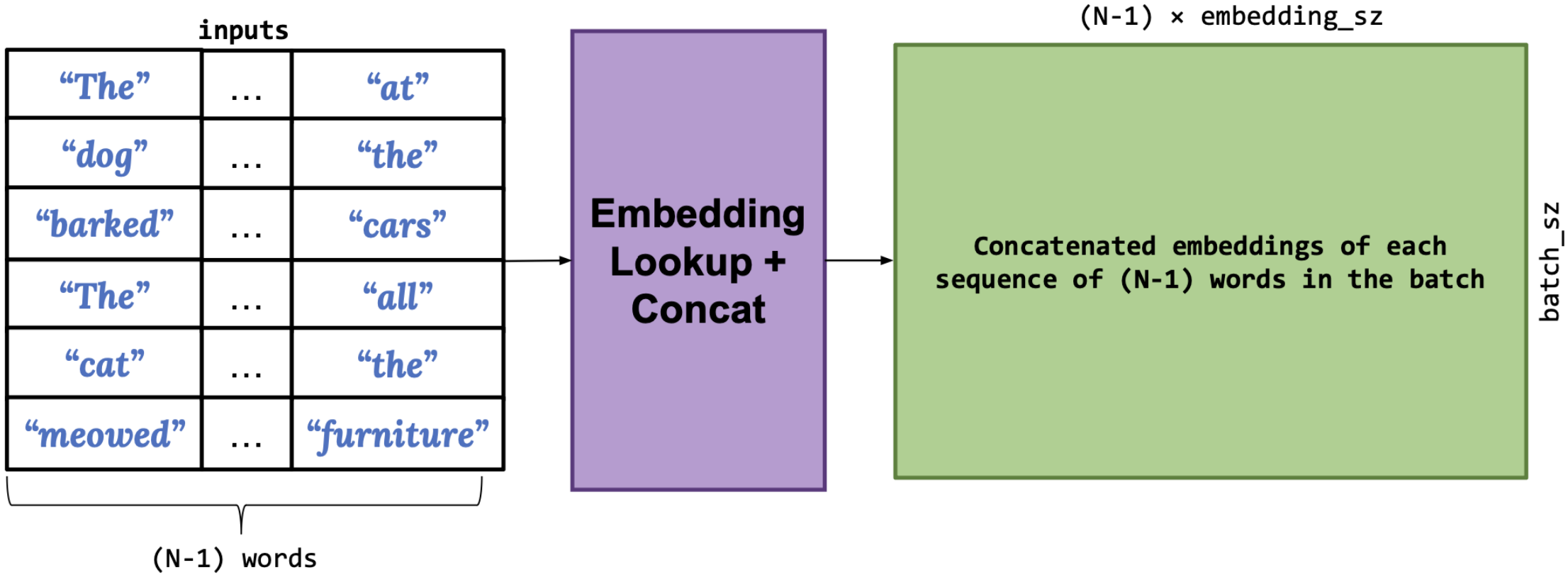
Size of Feed Forward N-gram Model

So what happens in the N-gram case?



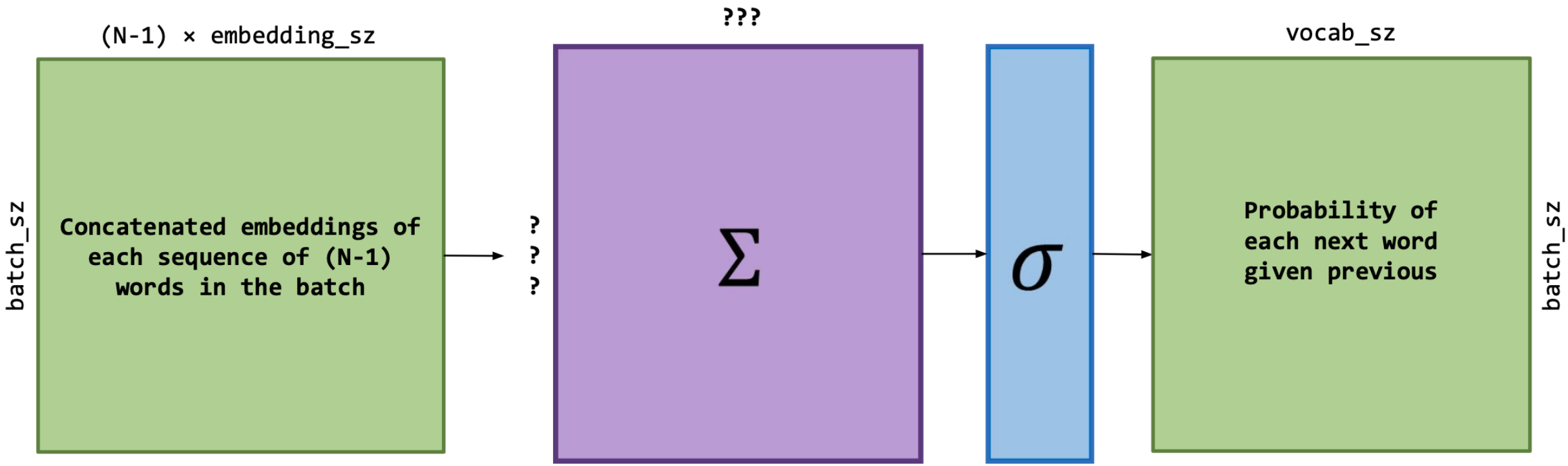
Size of Feed Forward N-gram Model

Embedding lookup + Concatenation still requires only one embedding matrix of size: `(vocab_sz, embedding_sz)`



Size of Feed Forward N-gram Model

But what happens to our feed forward layer?



Limitations of the N-gram model

- What issues do we run into using feed-forward N-gram models?
  - As the size of **N** increases, the number of weights needed for the linear layer becomes far too large.

Limitations of the N-gram model

- What issues do we run into using feed-forward N-gram models?
  - As the size of **N** increases, the number of weights needed for the linear layer becomes far too large.
  - Using a fixed **N** creates problems with the flexibility of our model.

Lack of Flexibility with N-grams

We would like for our language model **to be more aware of context** when deciding on how many words in the past to consider as "relevant".

For example, we can see that at some parts of the sentence below, smaller N-gram models should be sufficient to make predictions:

"The dog barked at one of the cats."

("The", "dog") → "barked"

Lack of Flexibility with N-grams

We would like for our language model to be more aware of context when deciding on how many words in the past to consider as “relevant”.

But when we look at other portions, common phrases and sequences of words may make it impossible to have any idea what should come next.

“The dog barked at one of the cats.”

(“at”, “one”, “of”, “the”) →
???

Lack of Flexibility with N-grams

We would like for our language model to be more aware of context when deciding on how many words in the past to consider as “relevant”.

But when we look at other portions, common phrases and sequences of words may make it impossible to have any idea what should come next.

“The dog barked at one of the cats.”

We want our model to recognize these patterns and dynamically adapt how it makes a prediction based on context.

Any questions?

Limitations of the N-gram model

What problems do we run into using Feed Forward N-gram models?

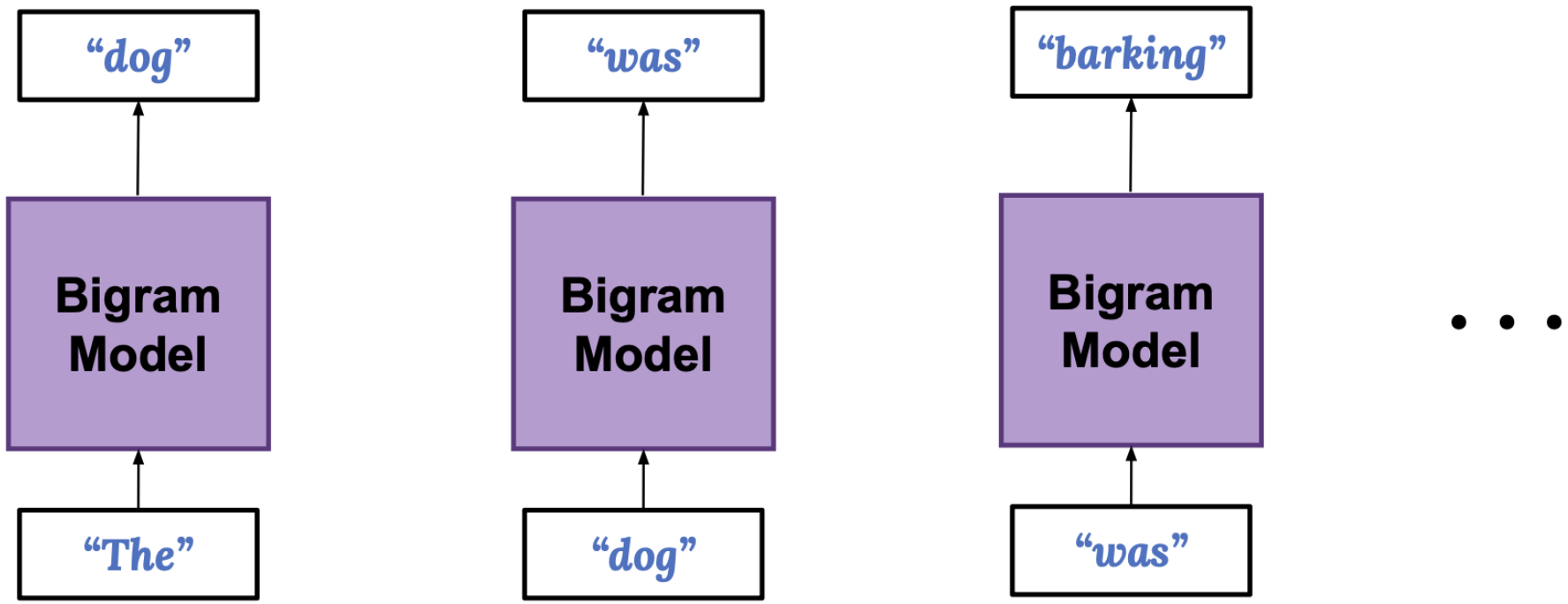
1. As the size of **N** increases, the number of weights needed for the linear layer becomes far too large.

2. Using a fixed **N** creates problems with the flexibility of our model.

We need a solution that is both computationally cheap and more dynamic in terms of its memory of previously seen words.

New Approach

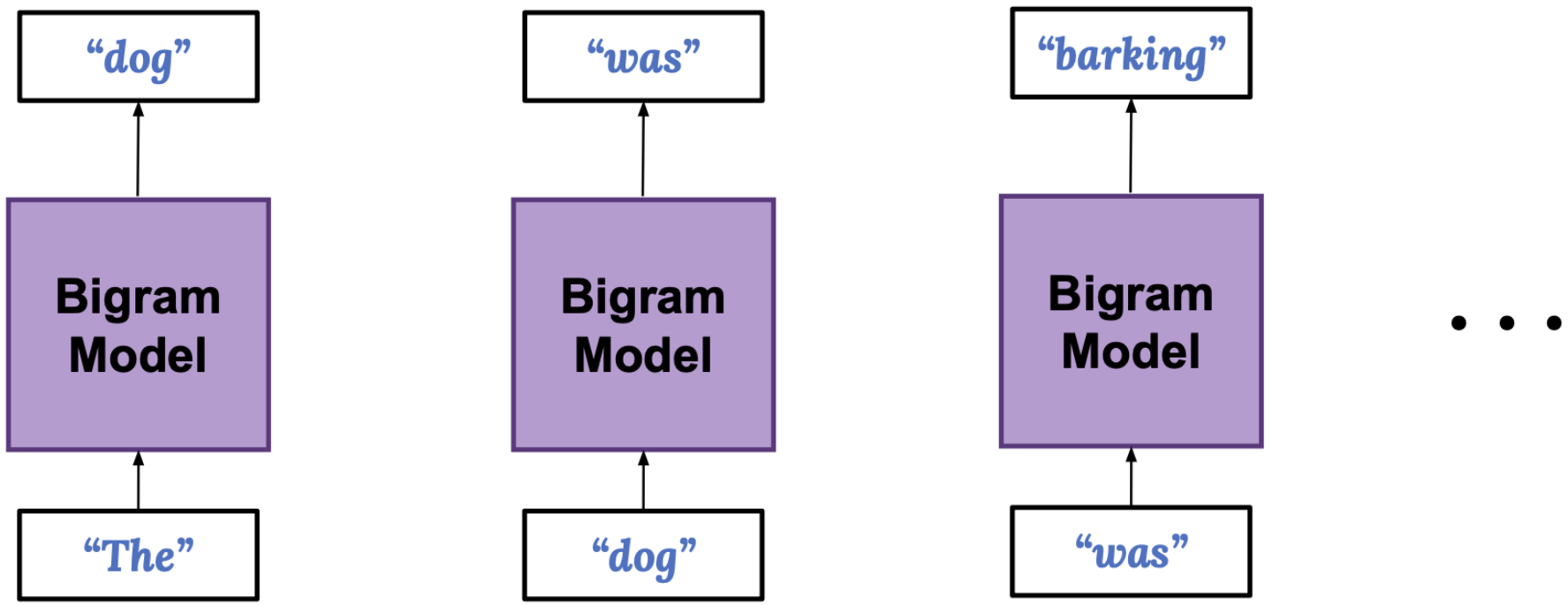
Let's revisit the bigram model and see several iterations of prediction using a bigram model:



New Approach

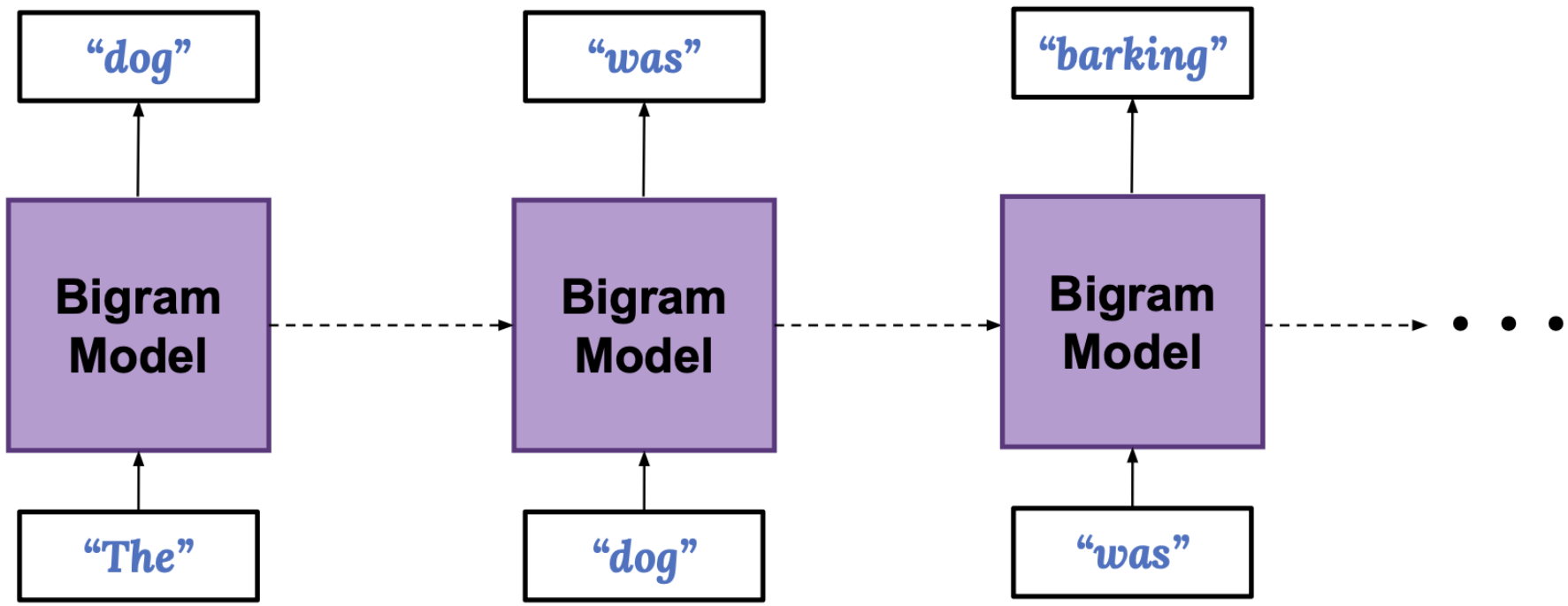
Any ideas?

Ideally, we would like to be able to keep “memory” of what words occurred in the past.



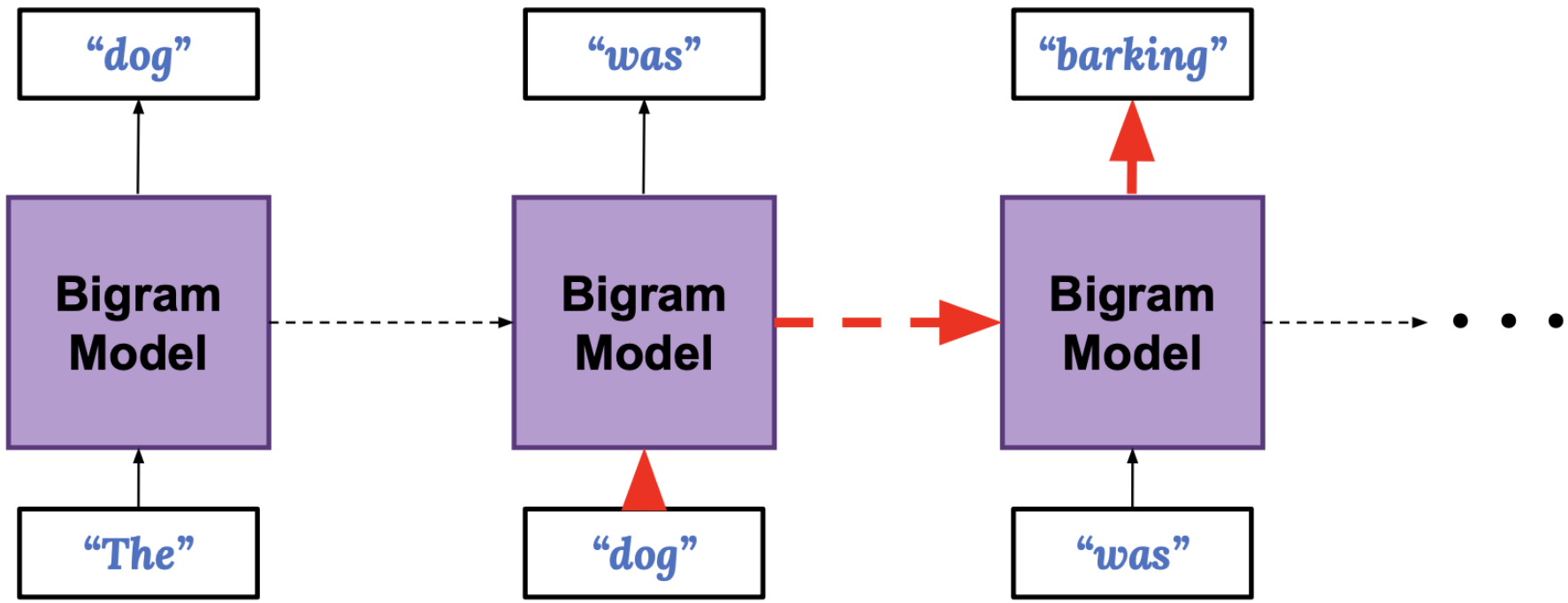
New Approach

What if we sequentially passed information from our previous bigram block into our next block?



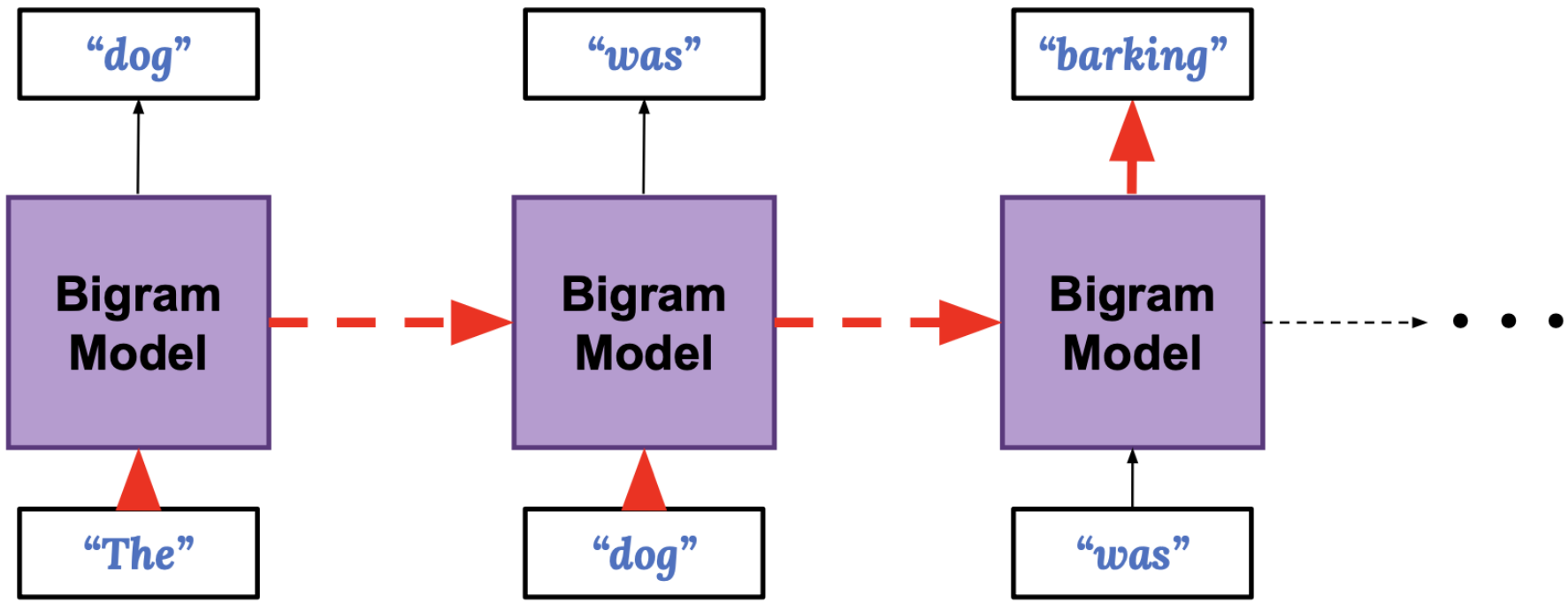
New Approach

If we follow the information flow, we see that when predicting “barking”, we have some way of knowing that “dog” was previously observed:



New Approach

In fact, we even have a way of knowing that “The” was observed!

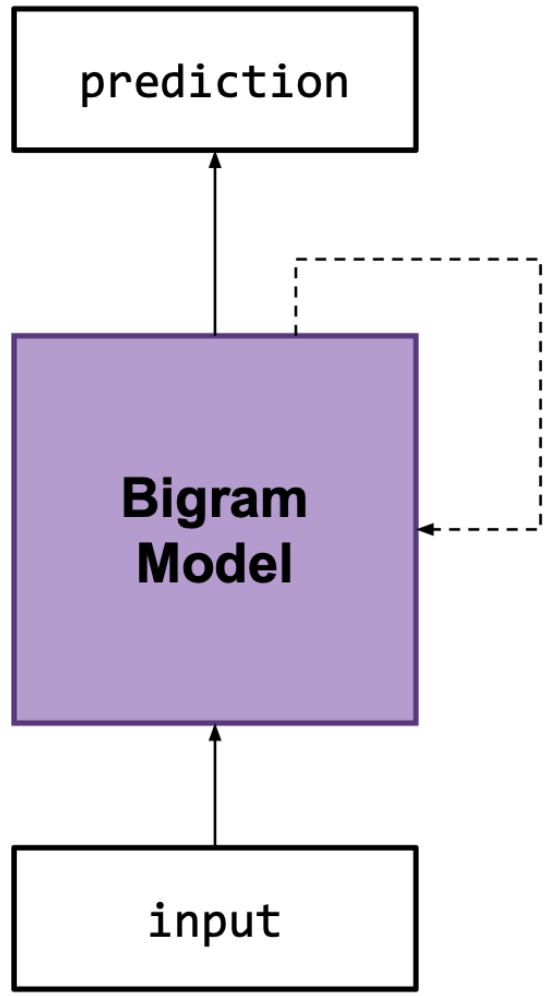


New Approach

We can represent this relationship using only one bigram block and connection that feeds from the output of the model back into the input.

We call this connection a *recurrent* connection.

We call the previous representation the “unrolled” representation.



Recurrent Neural Network (RNN)

Recurrent Neural Networks are networks in the form of a directed ***cyclic*** graph.

They pass previous *state* information from previous computations to the next.

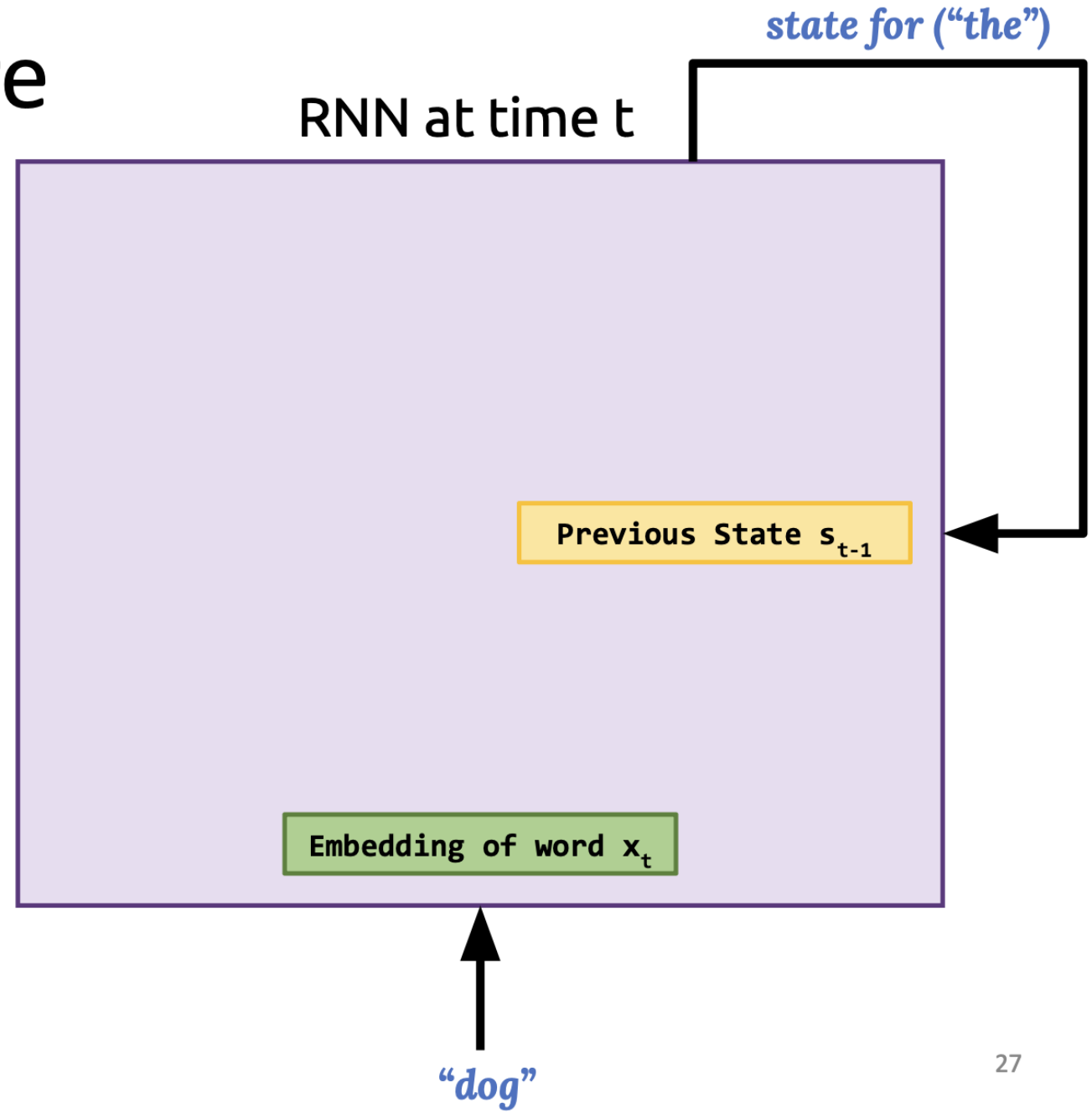
They can be used to process sequence data with relatively low model complexity when compared to feed forward models.

The block of computation that feeds its own output into its input is called the *RNN cell.*

Let's see how we can build one!

RNN Cell Architecture

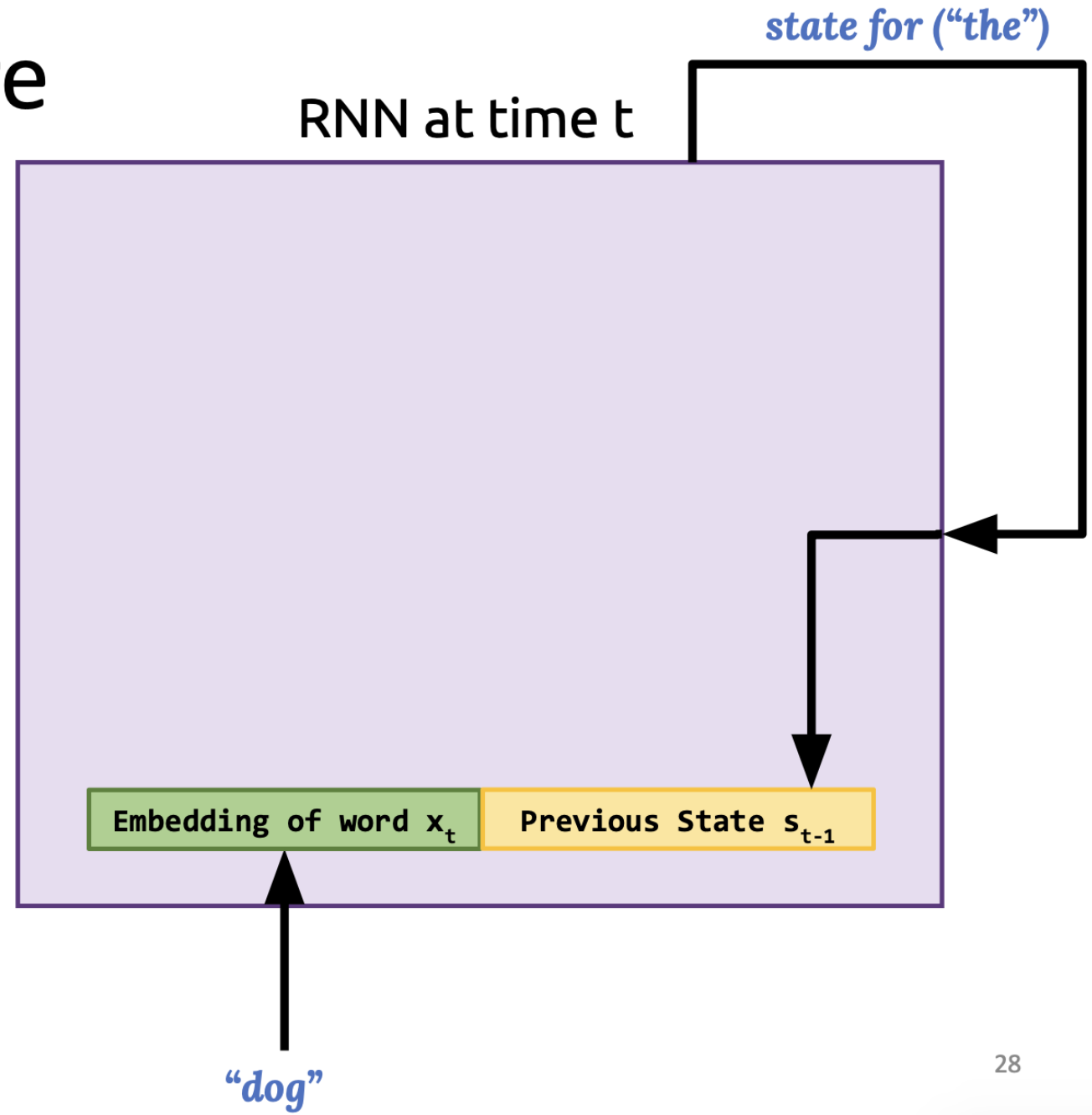
At each step of our RNN, we will get an input word, and a state vector from the previous cell.



RNN Cell Architecture

At each step of our RNN, we will get an input word, and a state vector from the previous cell.

We then concatenate the embedding and state vectors.

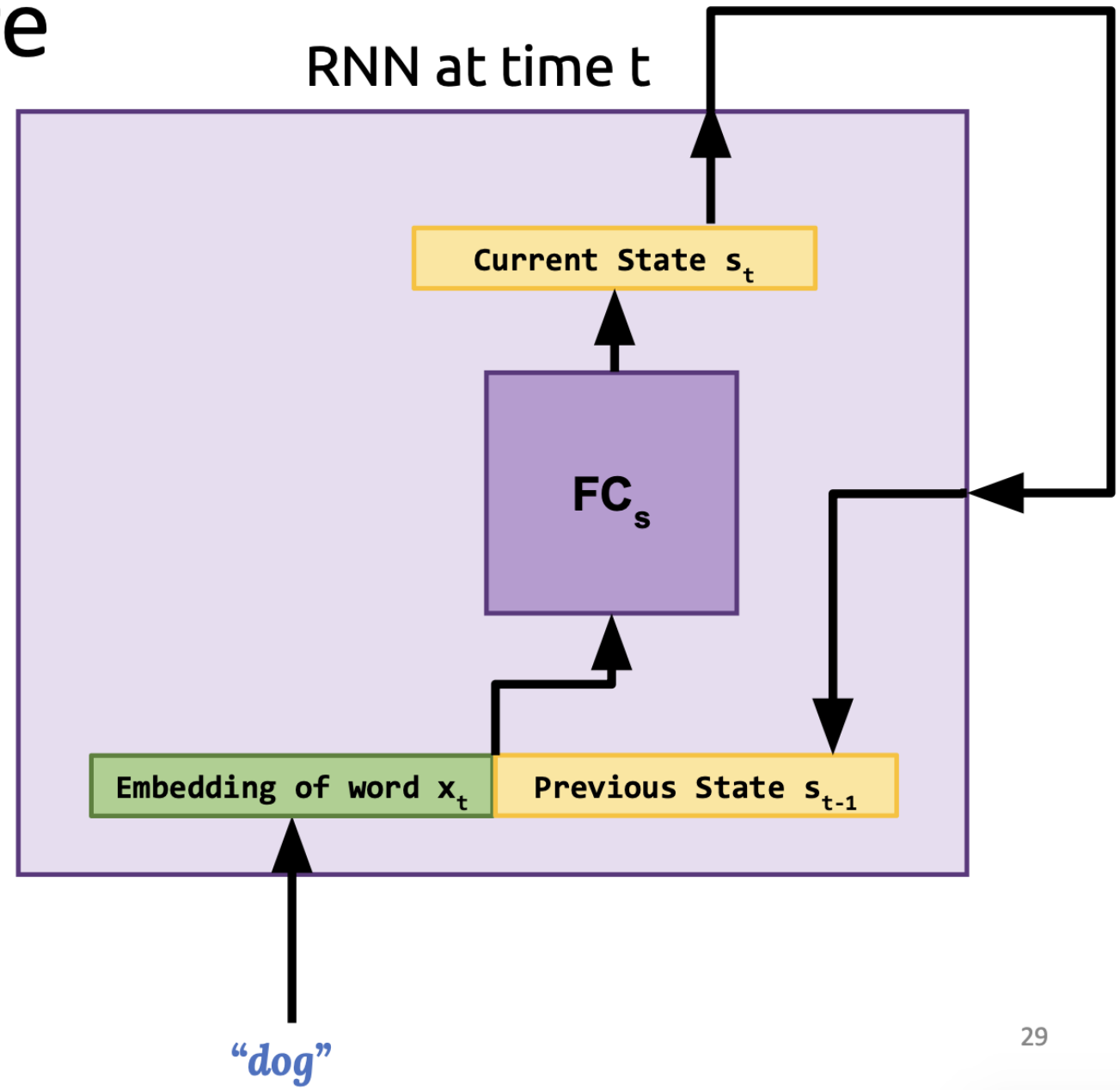


RNN Cell Architecture

At each step of our RNN, we will get an input word, and a state vector from the previous cell.

We then concatenate the embedding and state vectors.

We use a fully connected layer to compute the next state



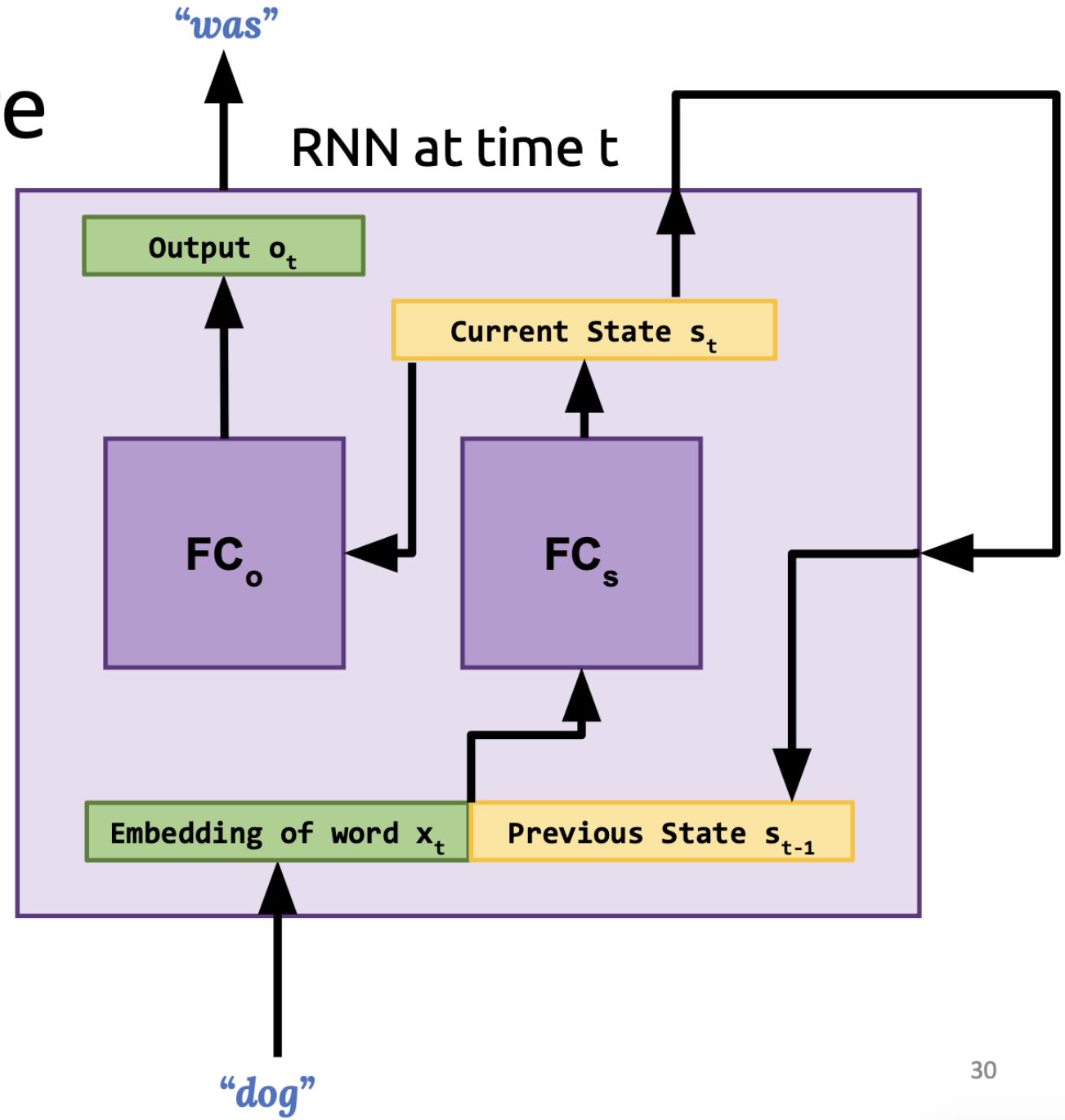
RNN Cell Architecture

At each step of our RNN, we will get an input word, and a state vector from the previous cell.

We then concatenate the embedding and state vectors.

We use a fully connected layer to compute the next state

We use another connected layer to get the output.

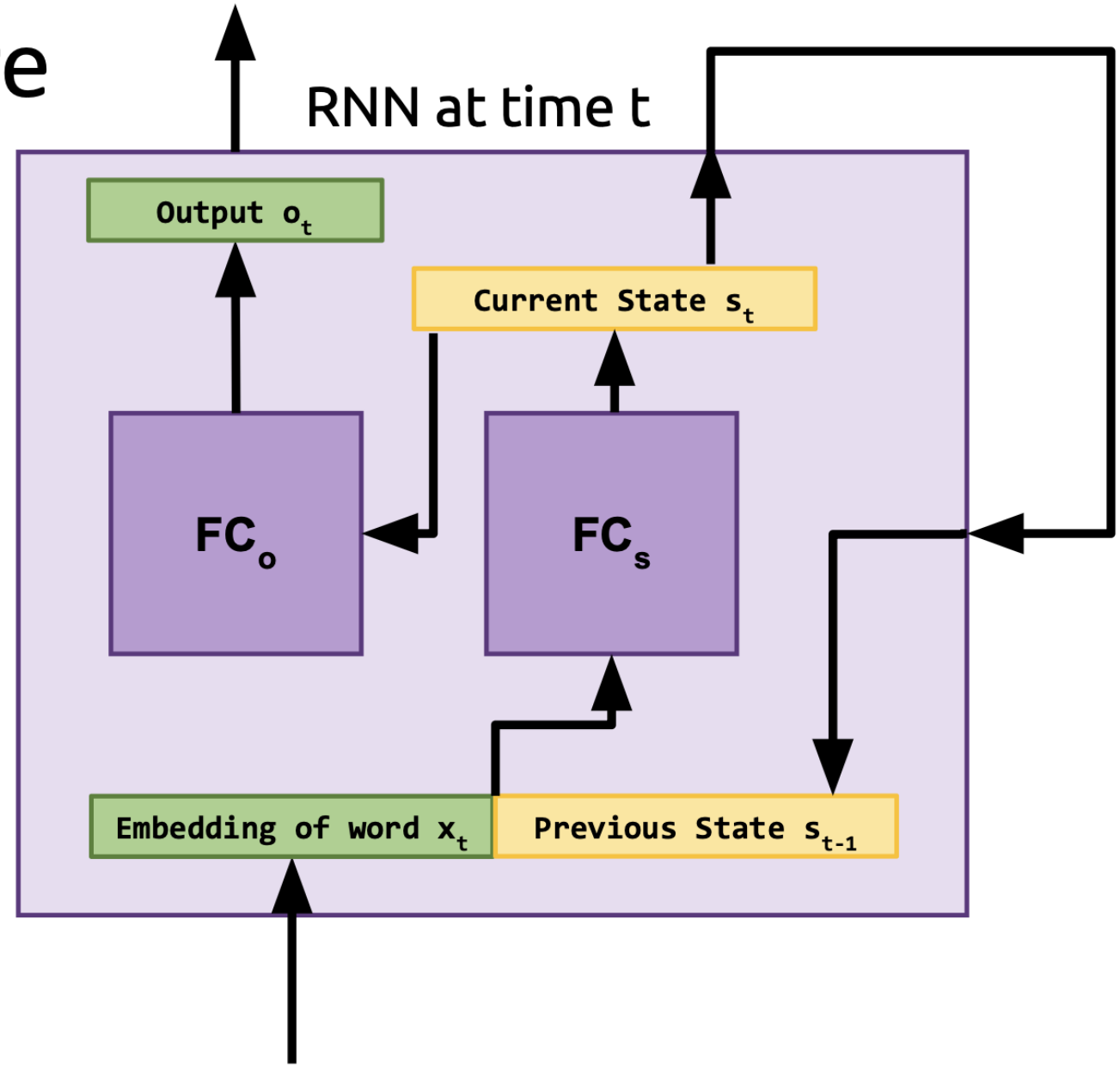


RNN Cell Architecture

We can represent the RNN in with the following equations:

$$s_t = \rho\big((e_t, s_{t-1})W_r + b_r\big)$$

$$o_t = \sigma(s_t W_o + b_o)$$



RNN Cell Architecture

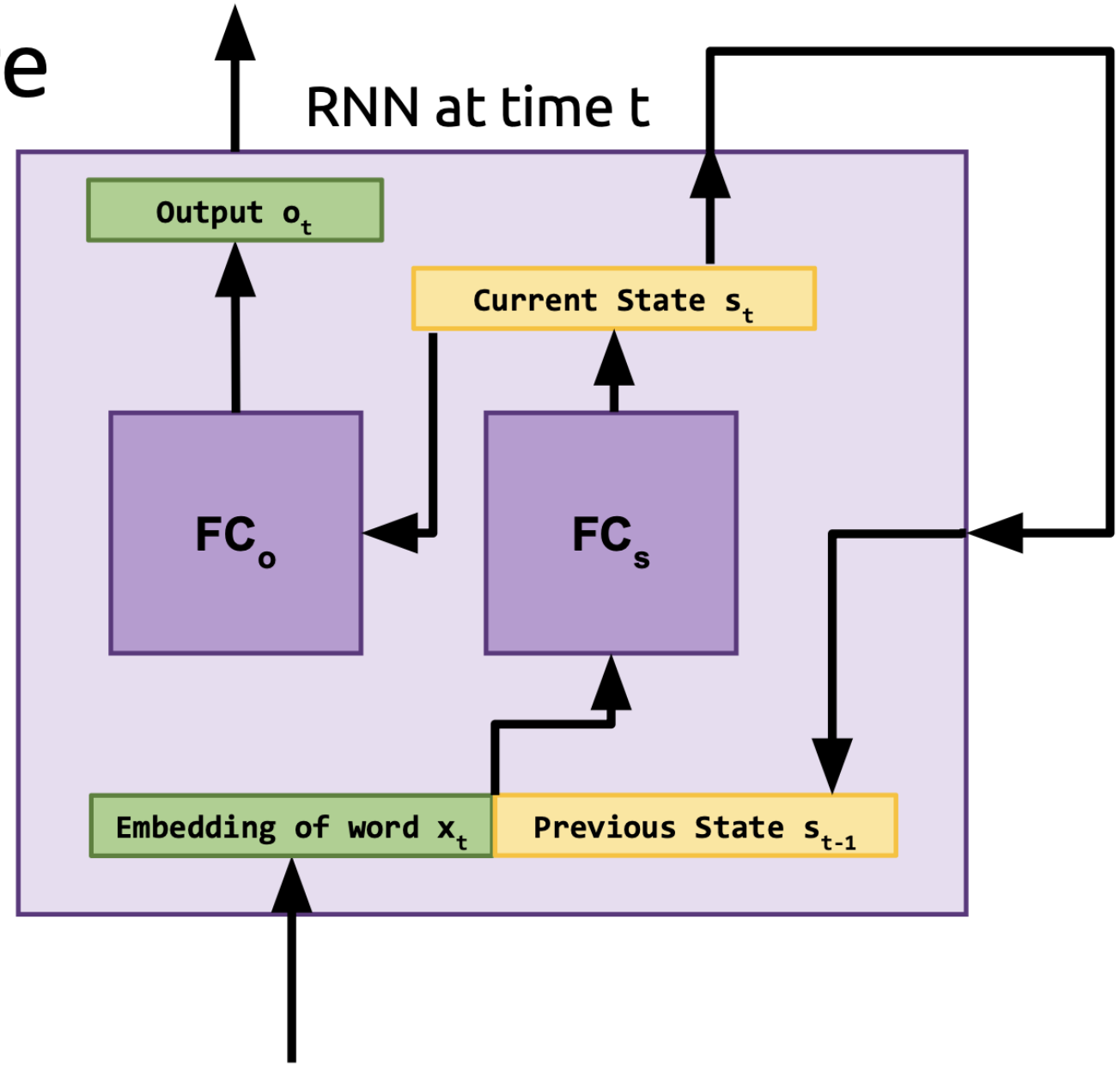
We can represent the RNN in with the following equations:

$$s_t = \rho\big((e_t, s_{t-1})W_r + b_r\big)$$

$$o_t = \sigma(s_t W_o + b_o)$$

Nonlinear activations (e.g. sigmoid, tanh)

Any questions?



RNN Cell Architecture

We can represent the RNN in with the following equations:

$$s_t = \rho\big((e_t, s_{t-1})W_r + b_r\big)$$

$$o_t = \sigma(s_t W_o + b_o)$$

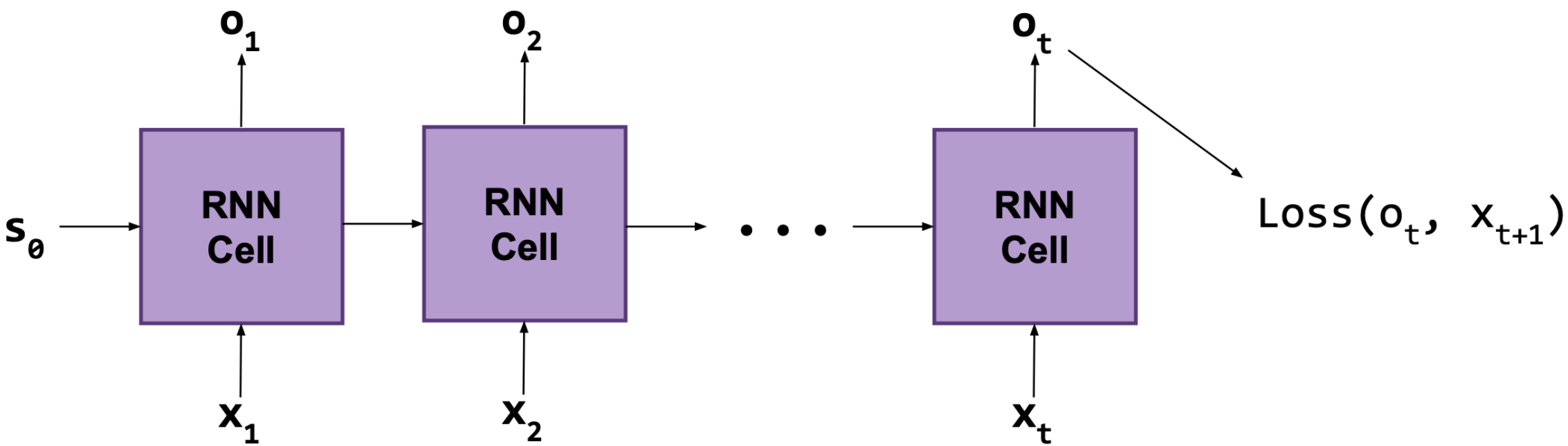
This brings up an immediate question: **what is s_0?**

Typically, we initialize s_0 to be a vector of zeros

(i.e. "initially, there is no memory of any previous words")

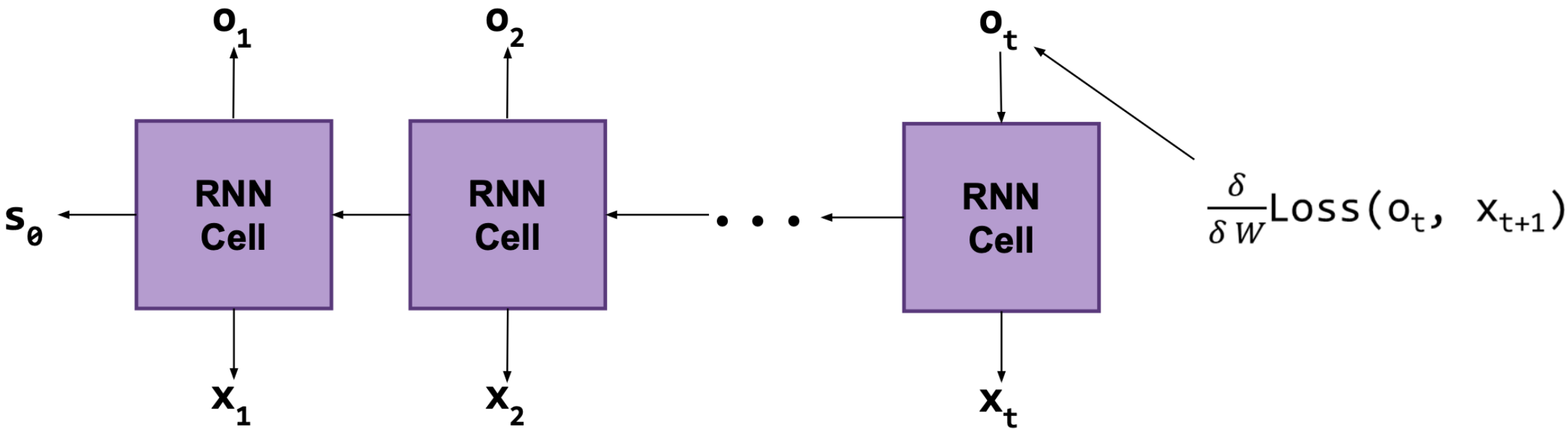
Training RNNs

We can calculate the cross entropy loss just as before since for any sequence of input words (x_1, x_2, …, x_t), we know the true next word x_{t+1}



Training RNNs

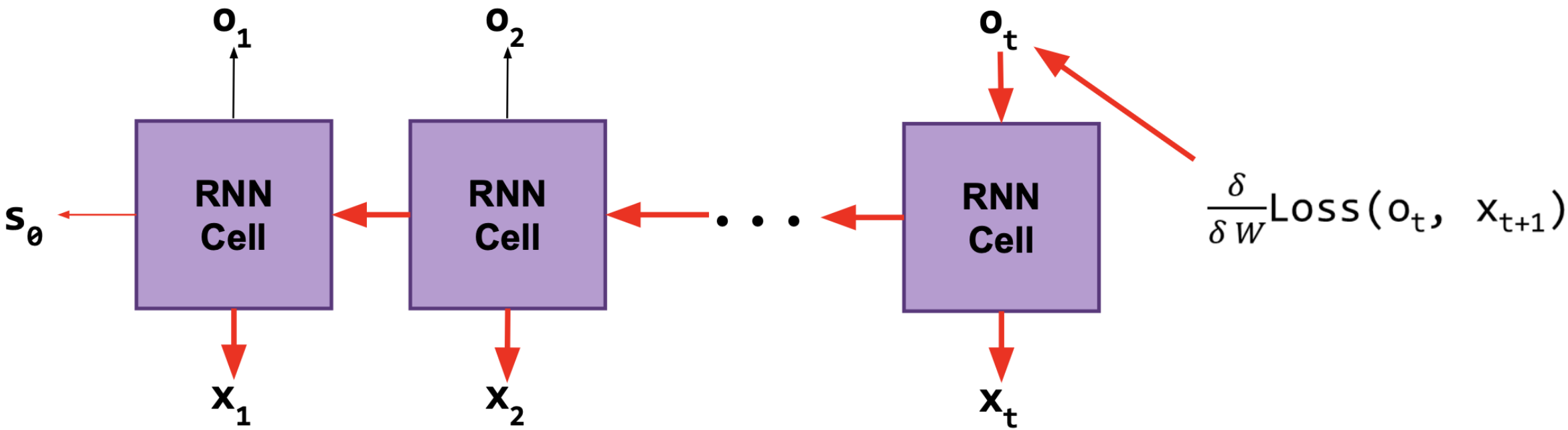
But what happens when we differentiate the loss and backpropagate?



Training RNNs

Not only do our gradients for o_t depend on x_t, but also on all of the previous inputs.

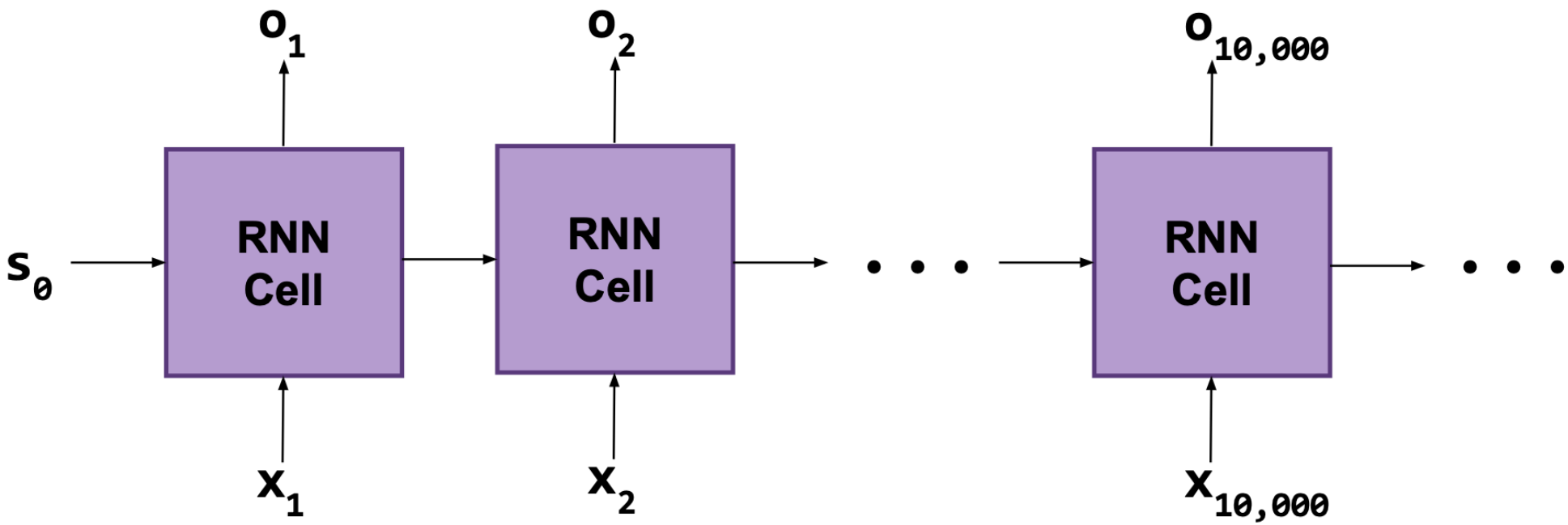
We call this *backpropagation through time.*



Training RNNs

But at what point do we stop and calculate the loss/update?

With this architecture, we can run the RNN cell for as many steps as we want, constantly accumulating memory in the state vector.



Training RNNs

Solution: We define a new hyperparameter called `window_sz.`

We now chop our corpus into sequences of words of size `window_sz`

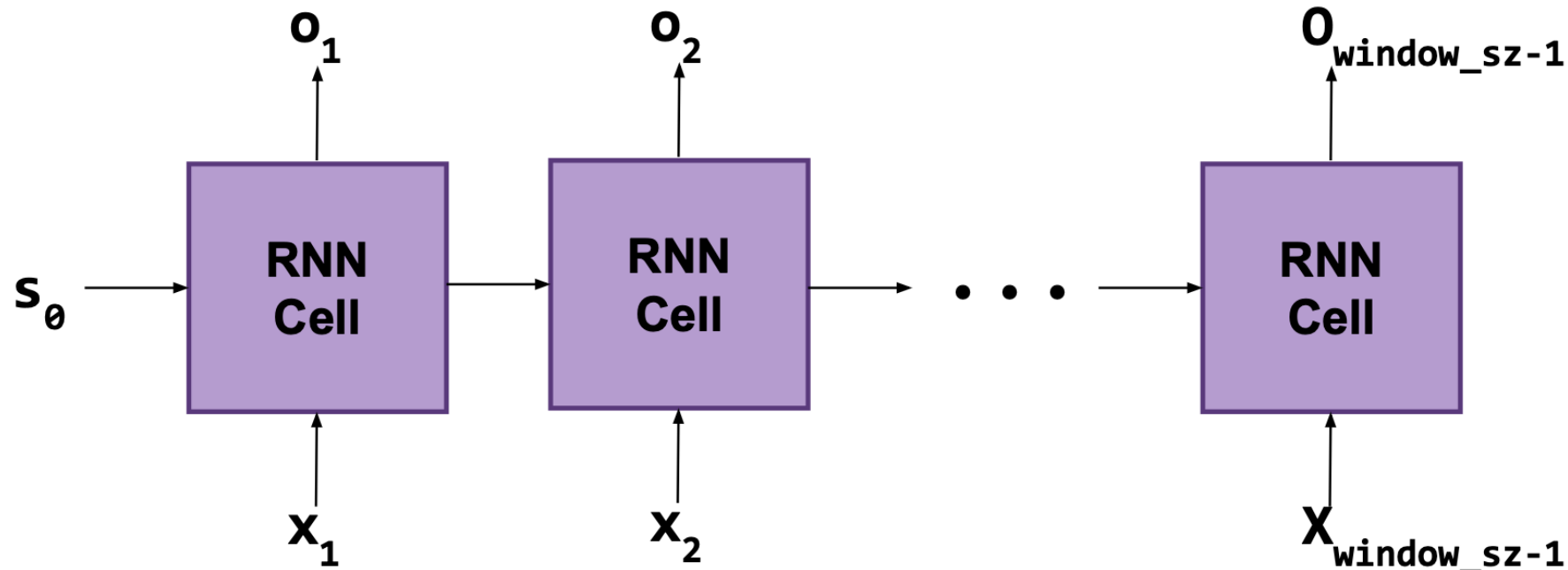
The new shape of our data should be:

`(batch_sz, window_sz, embedding_sz)`

Each example in our batch is a "window" of `window_sz` many words. Since each word is represented as an `embedding_sz`, that is the last dimension of the data.

Training RNNs

Now that every example is a window or words, we can run the RNN till the end of that window, and compute the loss for that specific window and update our weights

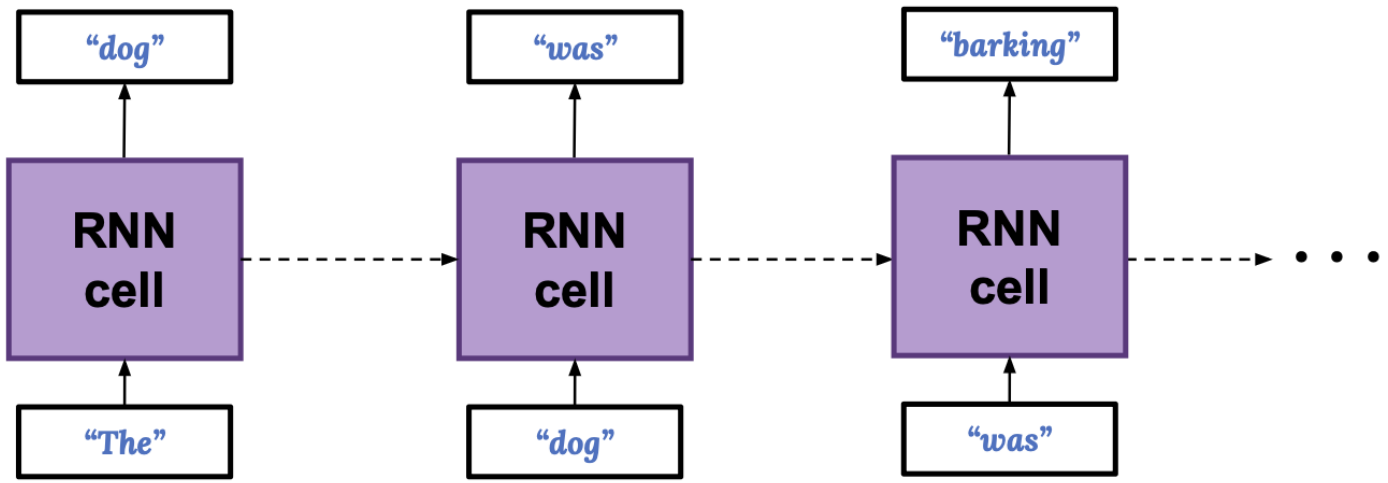


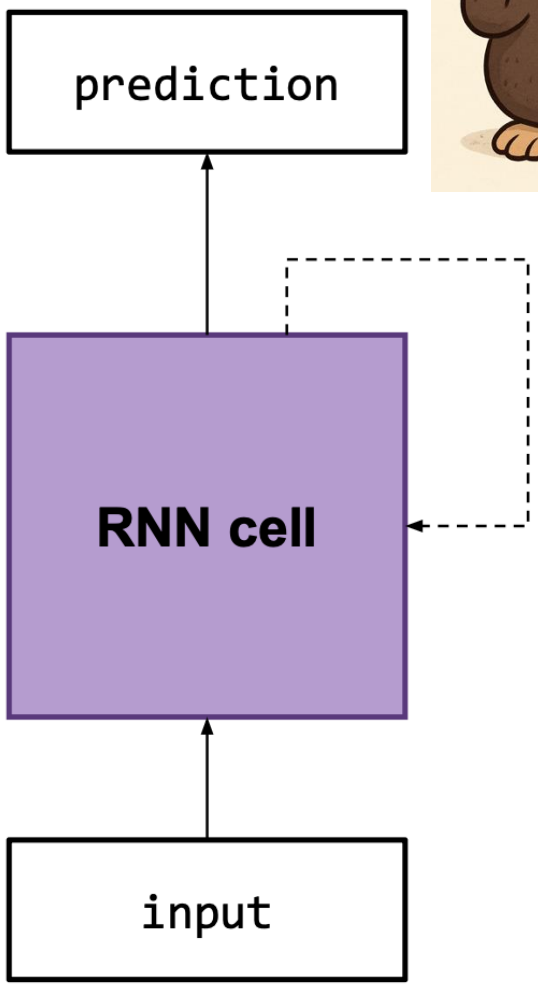
Any questions?

Does RNN fix the limitations of the N-gram model?

1. Number of of weights not dependent on N
2. State gives flexibility to choose context from near or far

"The dog was barking at one of the cats."





RNNs in Tensorflow

RNNs can be built from scratch using Python `for` loops:

```
prev_state = Zero vector
for i from 0 to window_sz:
   state_and_input = concat(inputs[i], prev_state)
   current_state = fc_state(state_and_input)
   outputs[i] = fc_output(current_state)
   prev_state = current_state
return outputs
```

RNNs in Tensorflow

RNNs can be built from scratch using Python `for loops`.

There's also a handy built-in Keras recurrent layer:

```
tf.keras.layers.SimpleRNN(units, activation, return_sequences)
```

RNNs in Tensorflow

RNNs can be built from scratch using Python `for loops`.

There's also a handy built-in Keras recurrent layer:

```
tf.keras.layers.SimpleRNN(units, activation, return_sequences)
```

The size of our output vectors

RNNs in Tensorflow

RNNs can be built from scratch using Python `for loops`.

There's also a handy built-in Keras recurrent layer:

```
tf.keras.layers.SimpleRNN(units, activation, return_sequences)
```

The activation function to be used in the FC layers inside of the RNN Cell

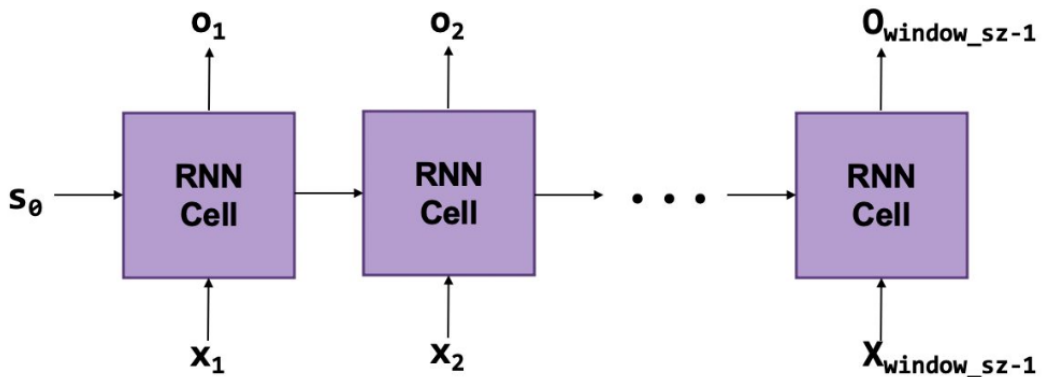
RNNs in Tensorflow

Any intuition why we would want return_sequences to be TRUE?

RNNs can be built from scratch using Python `for loops`.

There's also a handy built-in Keras recurrent layer:

```
tf.keras.layers.SimpleRNN(units, activation, return_sequences)
```



- If **True**: calling the RNN on an input sequence returns the whole sequence of outputs + final state output
- If **False**: calling the RNN on an input sequence returns just the final state output (Default)

RNNs in Tensorflow

RNNs can be built from scratch using Python `for loops`.

There's also a handy built-in Keras recurrent layer:

```
tf.keras.layers.SimpleRNN(units, activation, return_sequences)
```

Usage:

```
RNN = SimpleRNN(10) # RNN with 10-dimensional output vectors
Final_output = RNN(inputs) # inputs: a [batch_sz, seq_length, embedding_sz] tensor
```