CSCI 1470

Deep Learning

Eric Ewing

Day 18: Introduction to RNNs

Wenesday,
3/5/25

Updates

- I’m not holding office hours today.
- No weekly quiz this week.
- I’ll sort out consequences for people who have come forward by tomorrow.
    - I’ve responded to some people, but not everyone. If you’ve reached out acknowledging your use of AI and haven’t sent your chat transcripts, please send those. You’ll save me an email.

Collaboration

- Collaboration is *encouraged* and is great for learning
- Lots of people get help at hours and might have similar code if they were discussing issues or getting the same help from TAs, this is fine.
- But also… be aware that some students brought code to office hours generated by AI that they couldn't debug on their own…
  - Becomes a little unclear what to do in cases of fruit from a poisonous tree

Final Project

Groups of 3 or 4

Project: Go do some Deep Learning!

Option #1: Reimplement recent research paper

Option #2: Do something new (required for capstoning students)

Full project description, dates, and intermediate checkpoints:

https://hackmd.io/@BDLS25/ryoDyDmiJg

Final Project

Step 1: Form groups!

Groups of 3 or 4

Project: Go do some Deep Learning!

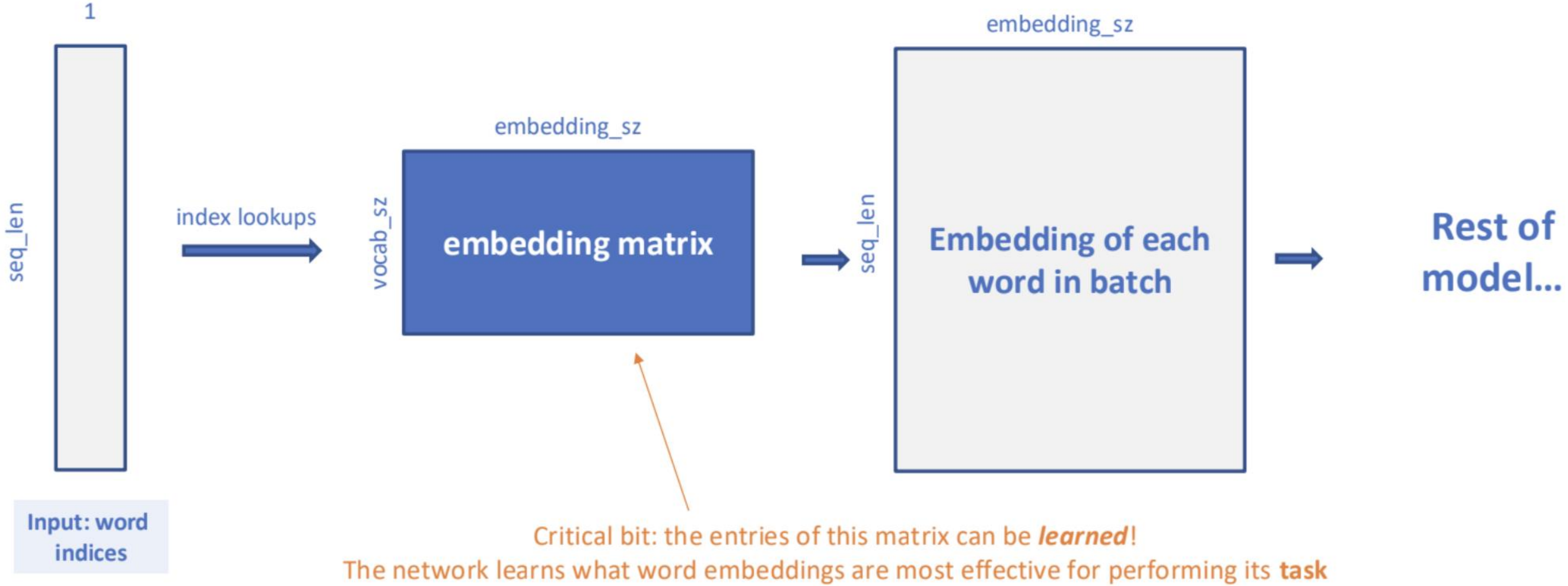
Option #1: Reimplement recent research paper

Option #2: Do something new (required for capstoning students)

Full project description, dates, and intermediate checkpoints:

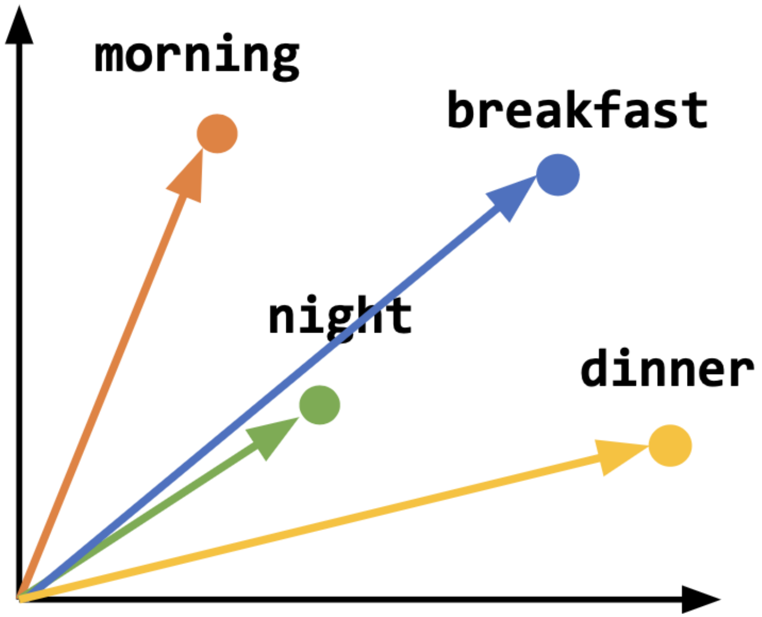
https://hackmd.io/@BDLS25/ryoDyDmiJg

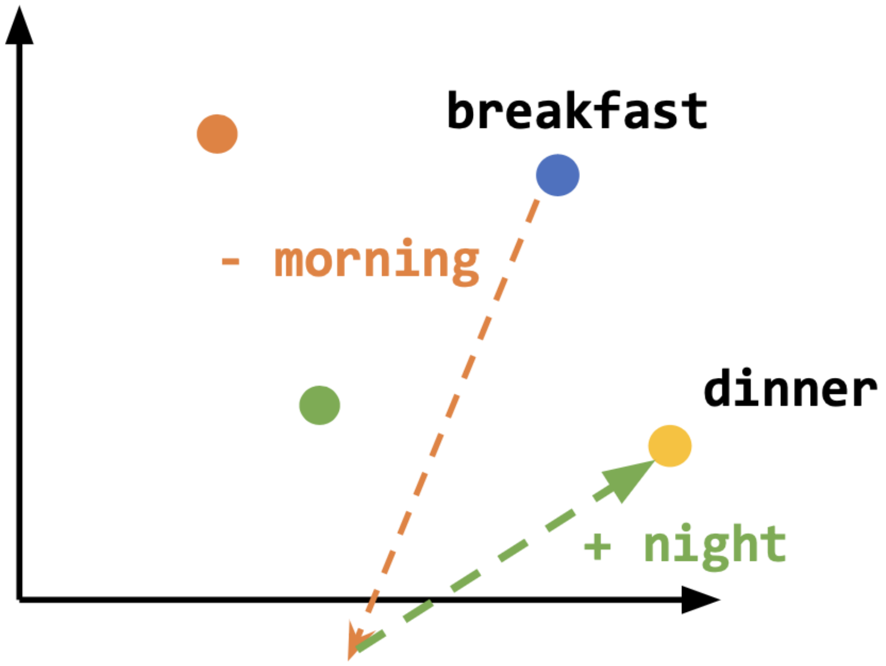
Using the Embedding Matrix in a Network



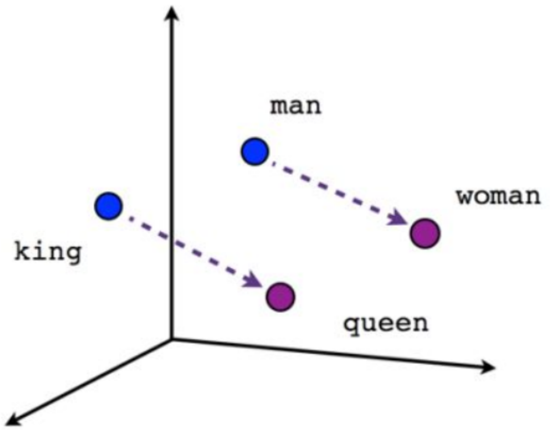
Vector arithmetic in the embedding matrix

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





More ‘semantic directions’ in embedding space

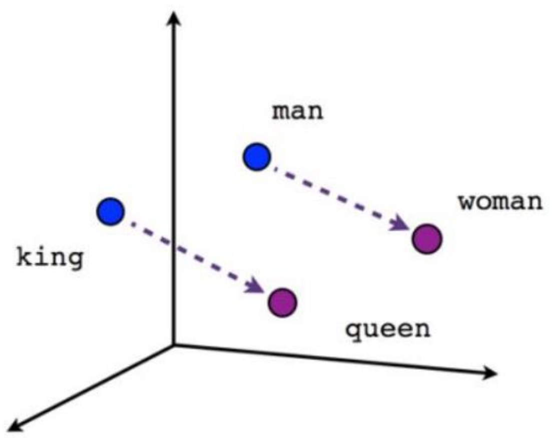


Male-Female

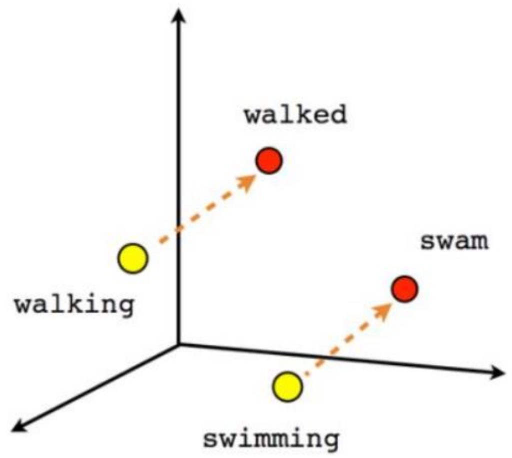
E(queen) – E(king) ≈
E(woman) – E(man)

Semantic: relating to meaning in language

More ‘semantic directions’ in embedding space



Male-Female



Verb tense

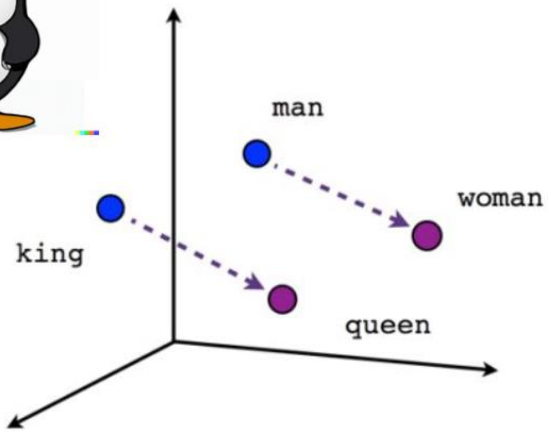
E(queen) – E(king) ≈ E(woman) – E(man)

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

Semantic: relating to meaning in language

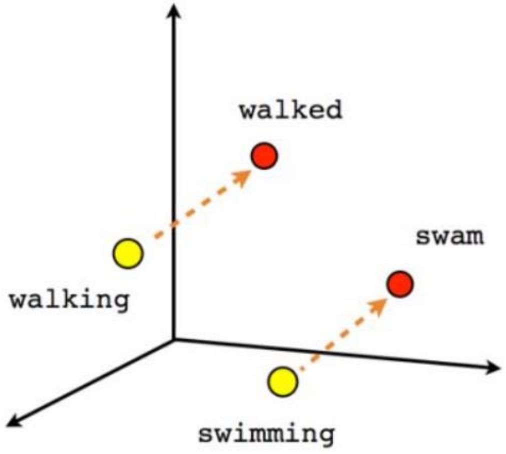
More ‘semantic directions’ in embedding space

Any questions?



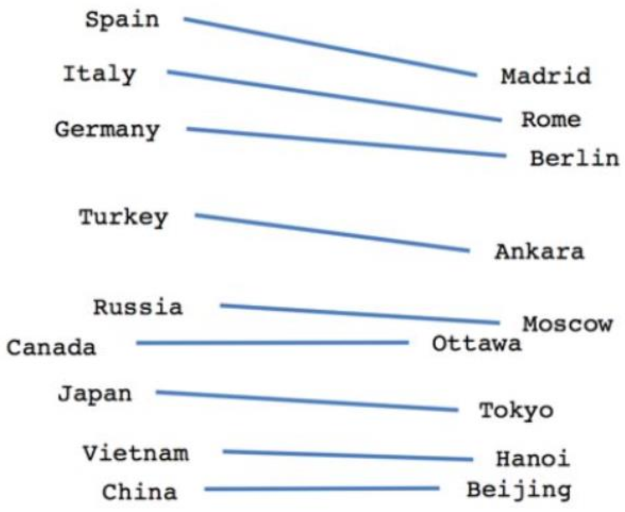
Male-Female

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



Verb tense

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

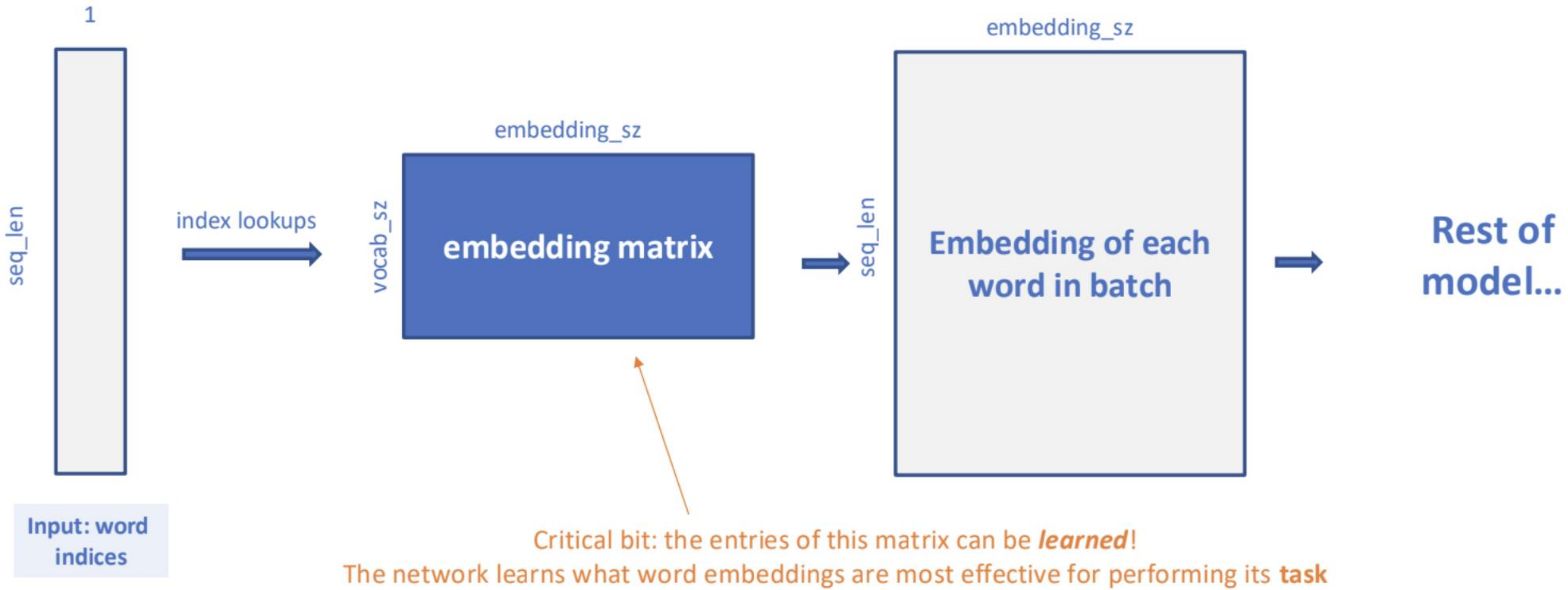


Country-Capital

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

Semantic: relating to meaning in language

Using the Embedding Matrix in a Network



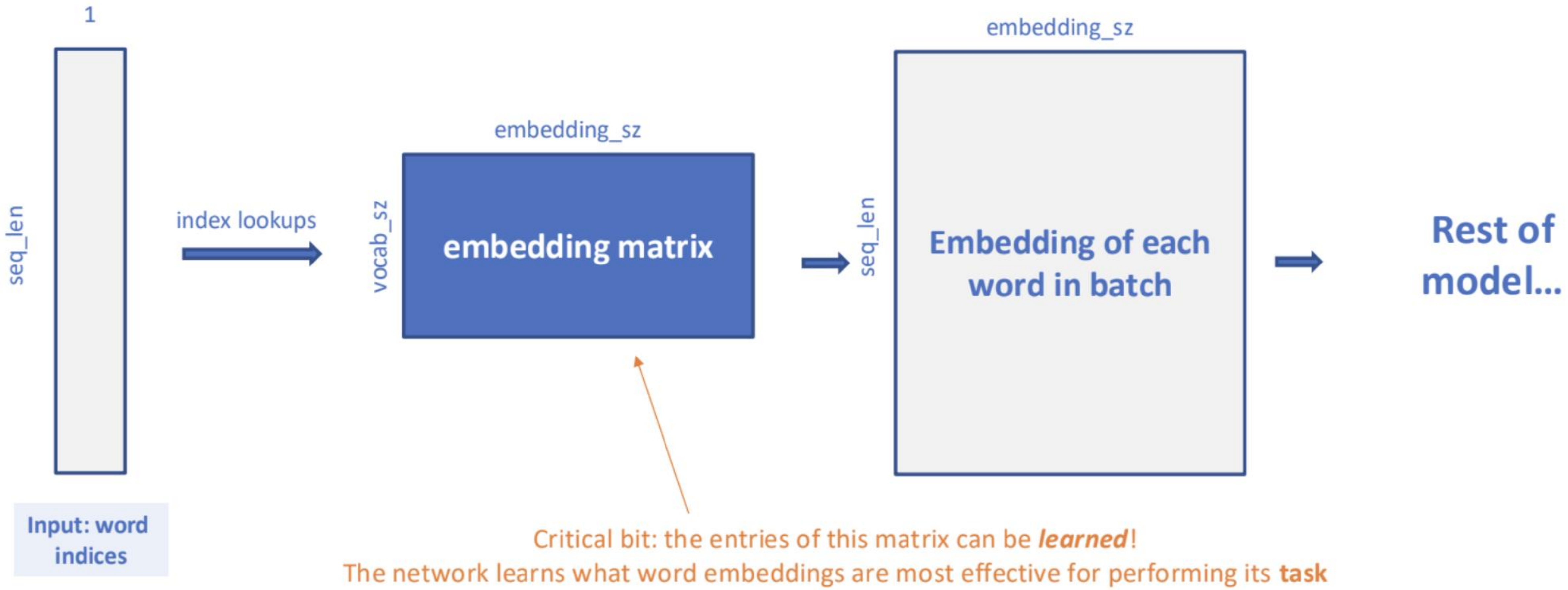
Using the Embedding Matrix in a Network

Say in the middle of training, the model sees:

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

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

Then the model sees a lot of “danced gleefully”



Using the Embedding Matrix in a Network

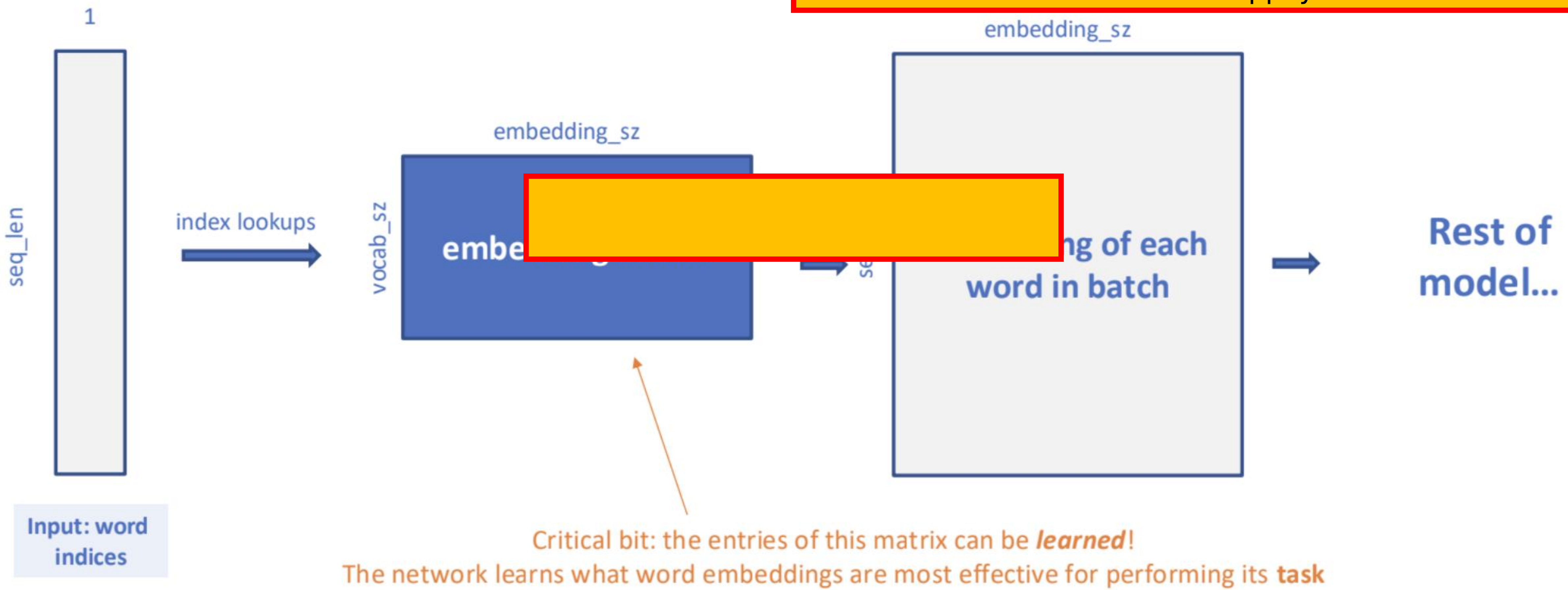
Say in the middle of training, the model sees:

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

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

Then the model sees a lot of “danced gleefully”

How do we modify the embedding of “gleefully” so that it is similar to “happily”?



Using the Embedding Matrix in a Network

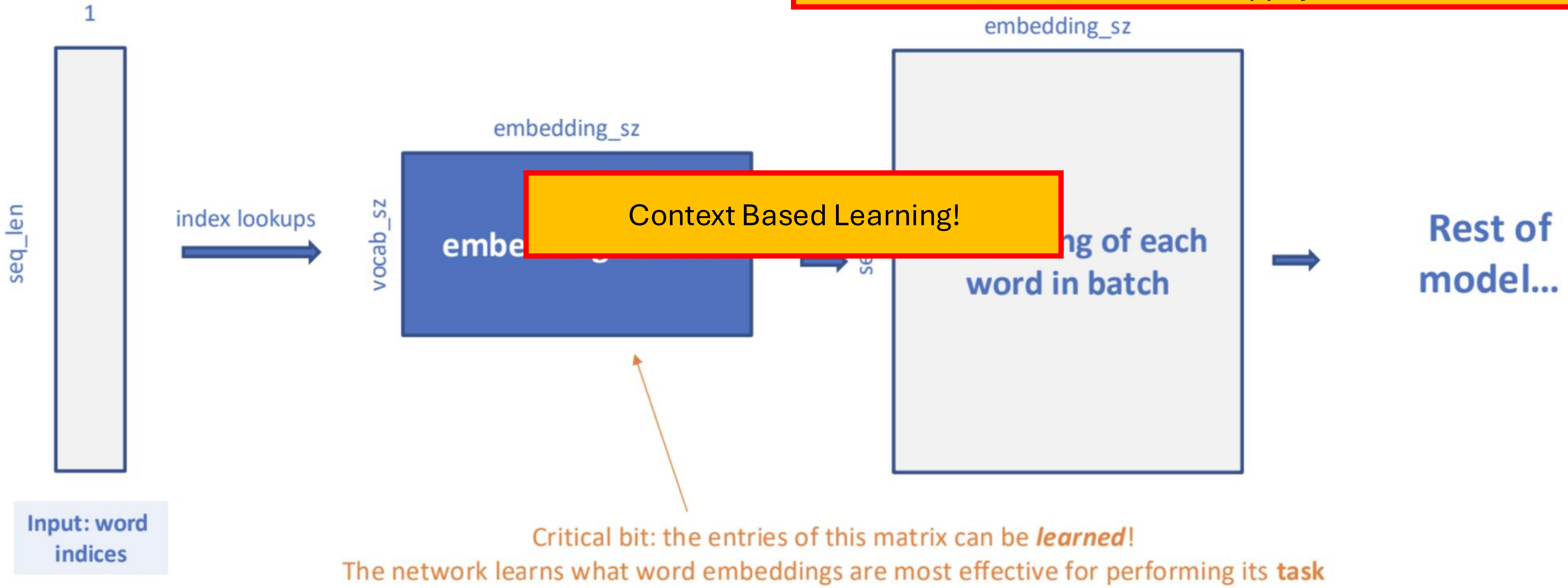
Say in the middle of training, the model sees:

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

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

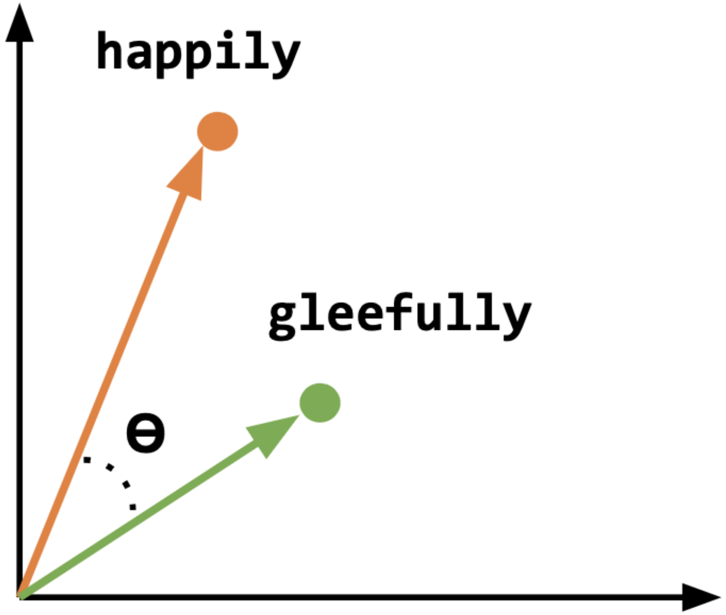
Then the model sees a lot of “danced gleefully”

How do we modify the embedding of “gleefully” so that it is similar to “happily”?



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 context-based approach

- Context is correlated with meaning, but context != meaning
- Synonyms typically have similar context:
  - **`P("happily" | "they danced")`**
  - **`P("gleefully" | "they danced")`**
- ...but often antonyms do, too:
  - **`P("happily" | "they danced")`**
  - **`P("unwillingly" | "they danced")`**
- "happily" and "unwillingly" might be used in similar contexts, but have the ***opposite*** meaning ➡ a language model might (erroneously) give them similar embeddings

Other failure modes are even more dire

What happens when your dataset reflects historical / societal biases?

Other failure modes are even more dire

What happens when your dataset reflects historical / societal biases?

Google News word2vec:

- Large set of *pretrained* word embeddings, published 2013
- Dataset: news articles aggregated by Google News (100 billion words)

Other failure modes are even more dire

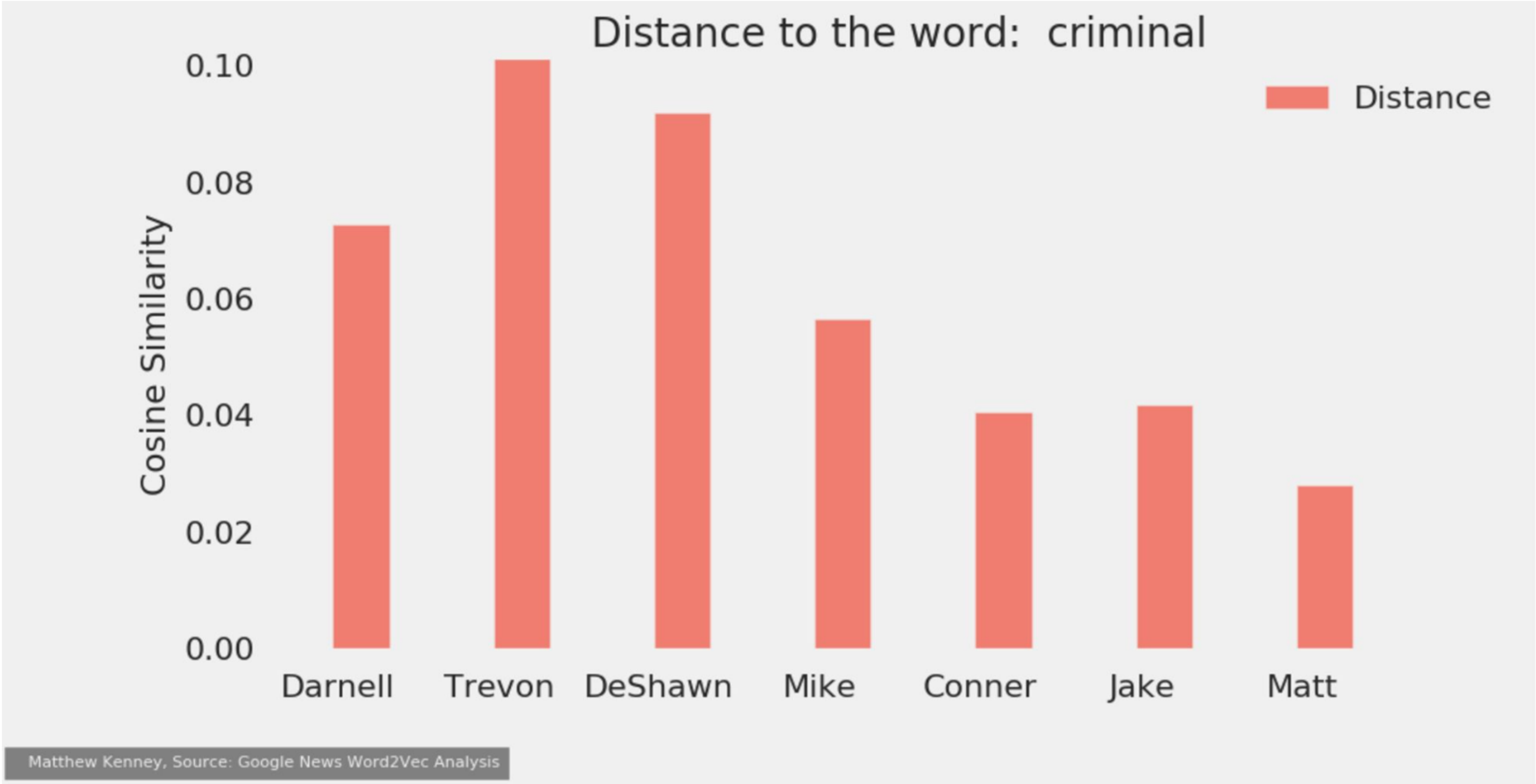
What happens when your dataset reflects historical / societal biases?

Google News word2vec:

- Large set of *pretrained* word embeddings, published 2013
- Dataset: news articles aggregated by Google News (100 billion words)

What kinds of relationships do these embeddings contain?

Google News word2vec

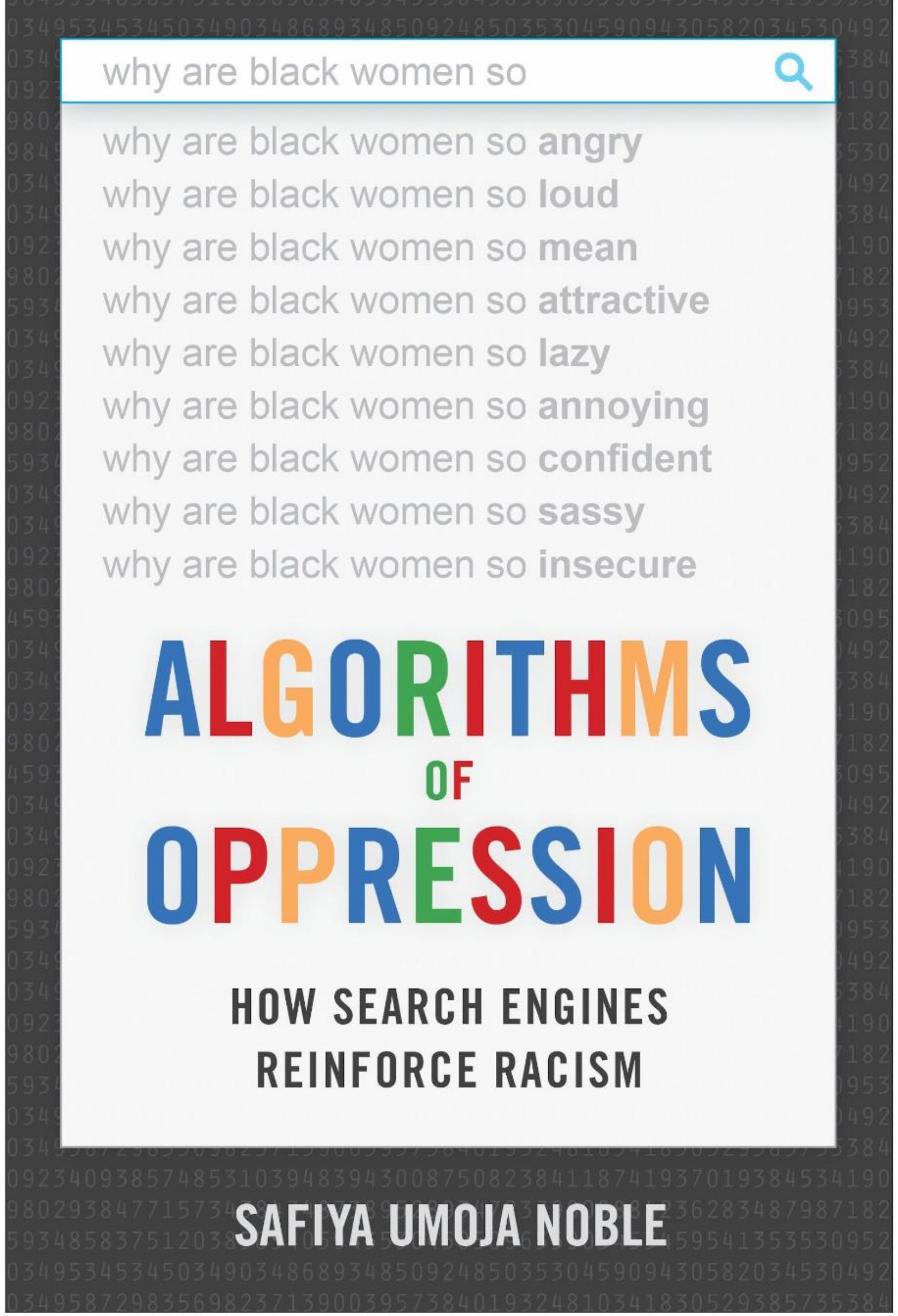


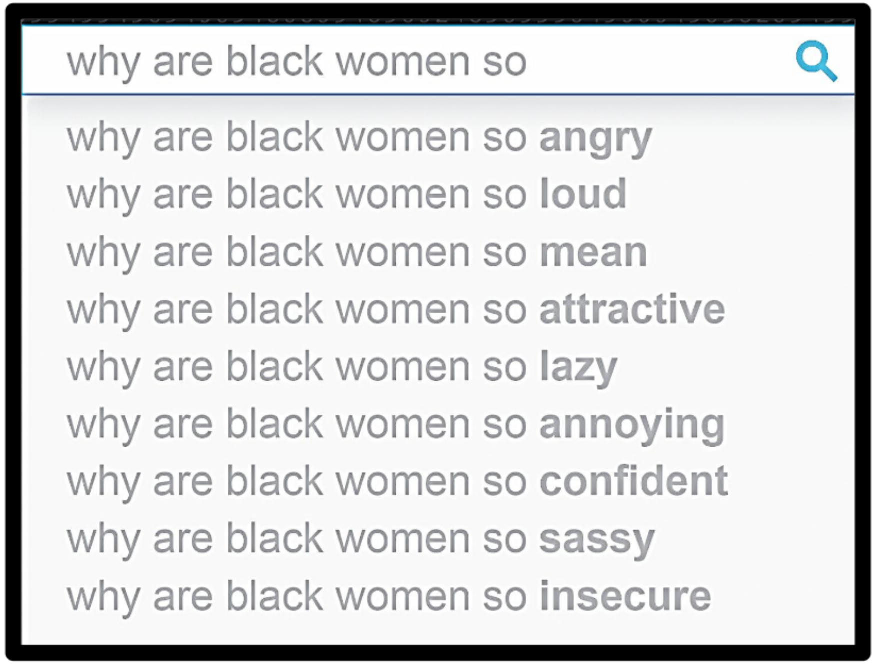
http://www.mattkenney.me/google-word

Google News word2vec

- **Why did this happen?**
  - The training dataset (news articles) was biased.
  - The news cycle **over-represents** crimes by black perpetrators
    - (Entman 94, Gilliam et.al. 96, Dixon 08, Dixon 15) – this is true over time as well
  - Viewers respond more strongly to news stories about crimes by black perpetrators.
    - (Dixon and Maddox 06, Dixon and Azocar 07, Hurley et.al. 15)
    - (News outlets optimize for clicks, therefore report crime by black people more)

http://www.mattkenney.me/google-word


why are black women so
why are black women so angry
why are black women so loud
why are black women so mean
why are black women so attractive
why are black women so lazy
why are black women so annoying
why are black women so confident
why are black women so sassy
why are black women so insecure
ALGORITHMS
OF
OPPRESSION
HOW SEARCH ENGINES
REINFORCE RACISM
SAFIYA UMOJA NOBLE



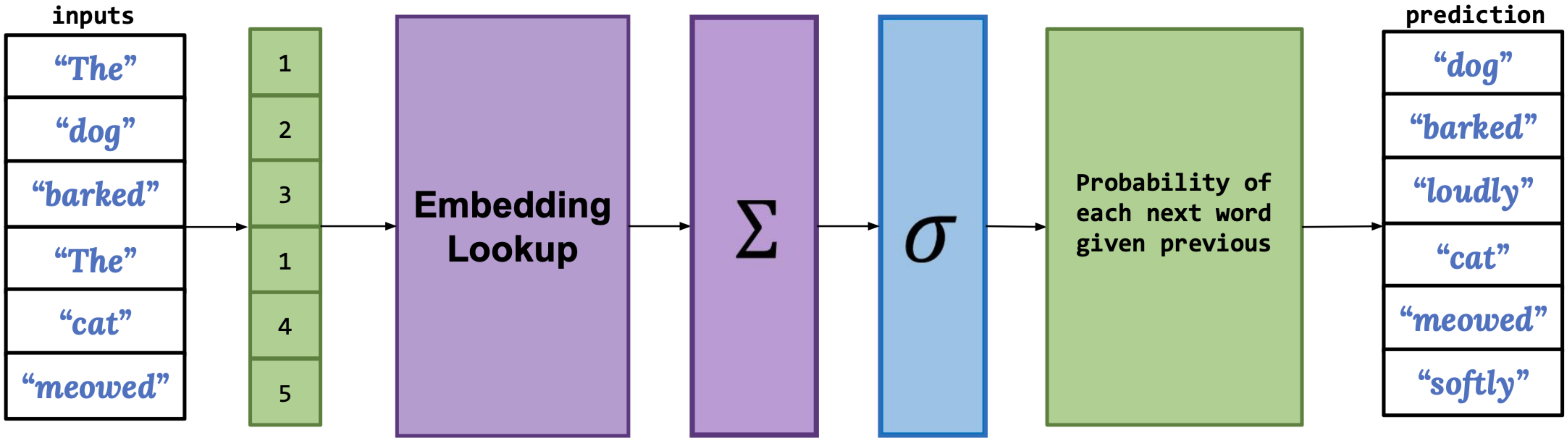
- In ~2010, when Noble started working on this book, these were the real Google autocomplete suggestions
- ***Takeaway: language models reproduce the biases of the data on which they are trained***
  - ...unless special care is taken—we have an upcoming lab on this!

Limitations of the N-gram model

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

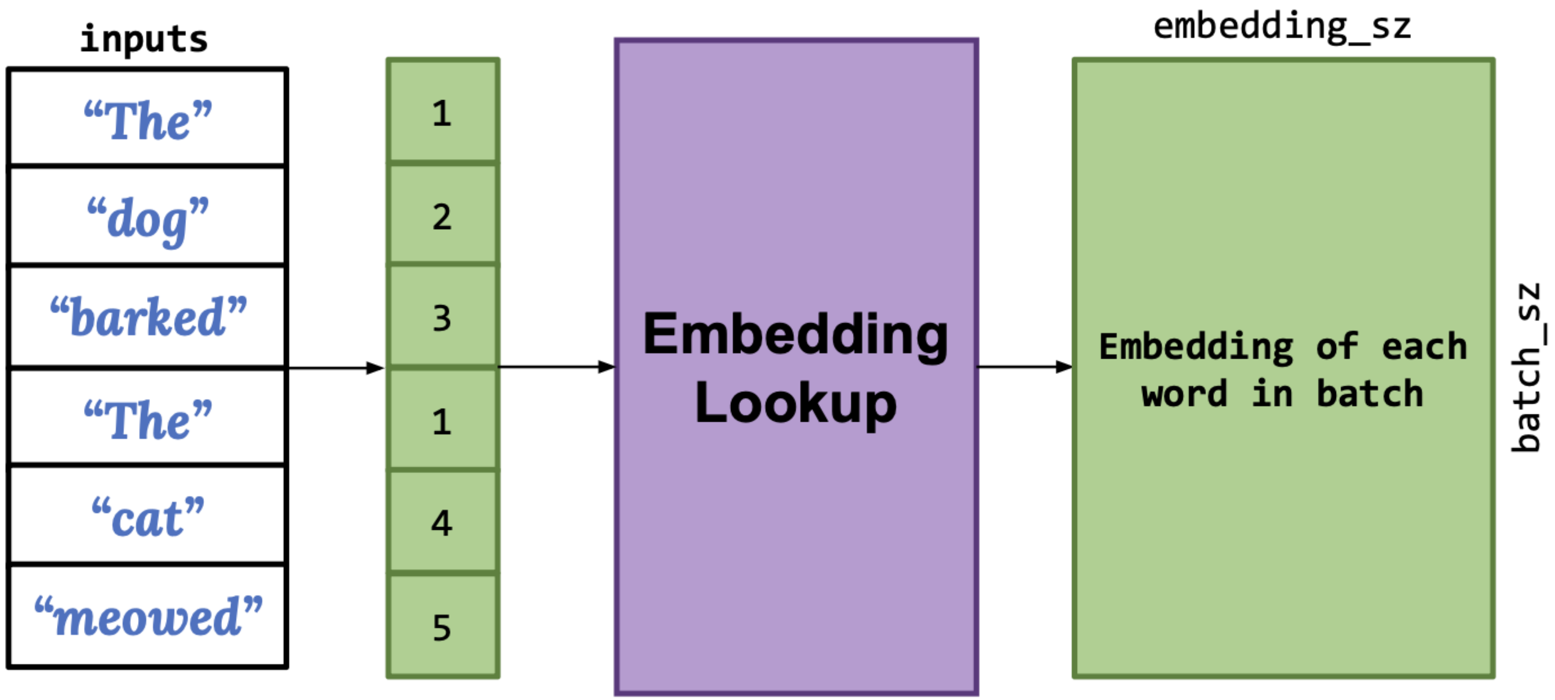
Size of Feed Forward bigram Model

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



Size of Feed Forward bigram Model

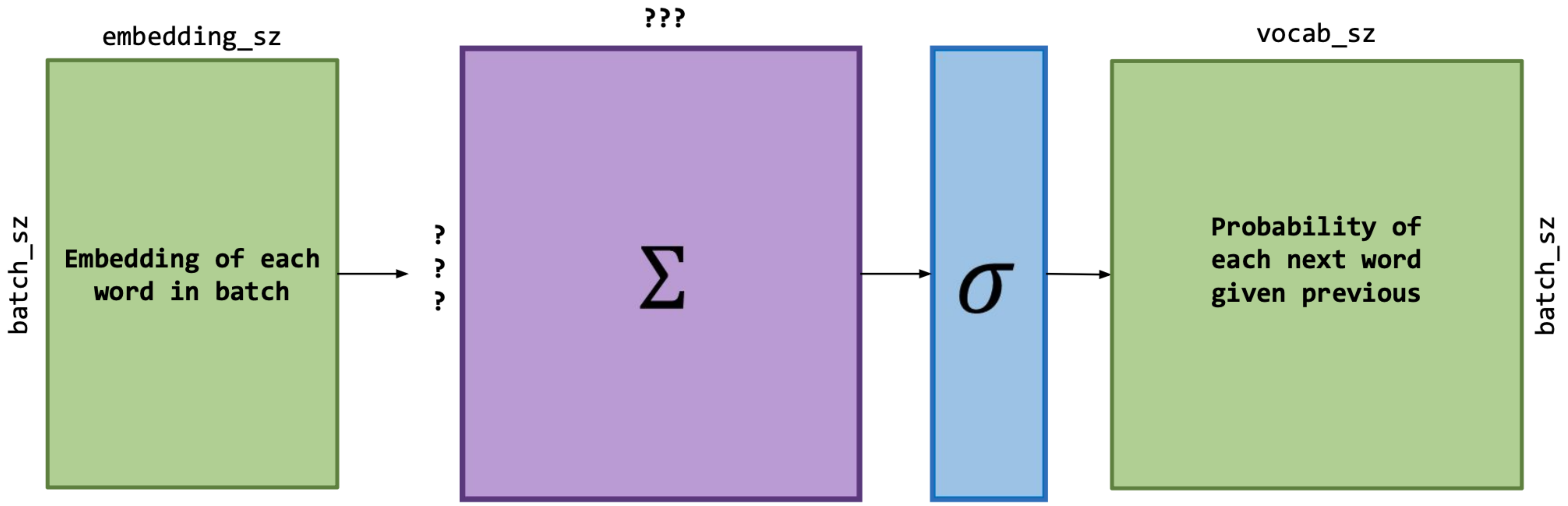
To preform embedding lookup on our entire batch, we just need one embedding matrix of size: `(vocab_sz, embedding_sz)`



Size of Feed Forward bigram Model

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

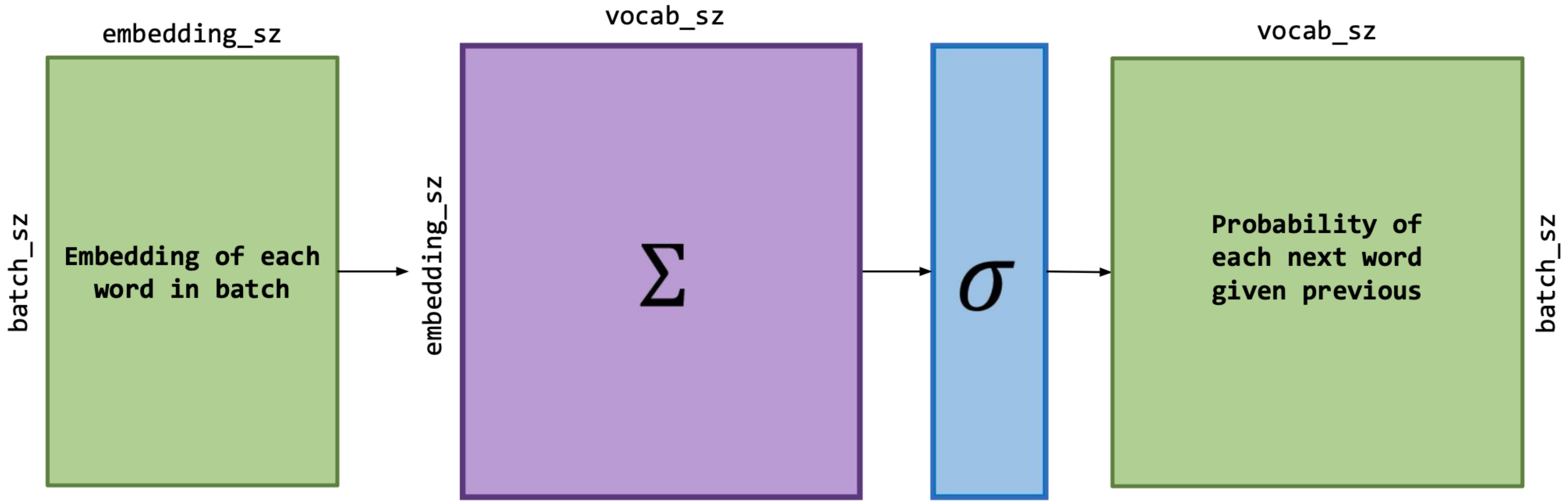
`(batch_sz, embedding_sz) × (???, ???) → (batch_sz, vocab_sz)`



Size of Feed Forward bigram Model

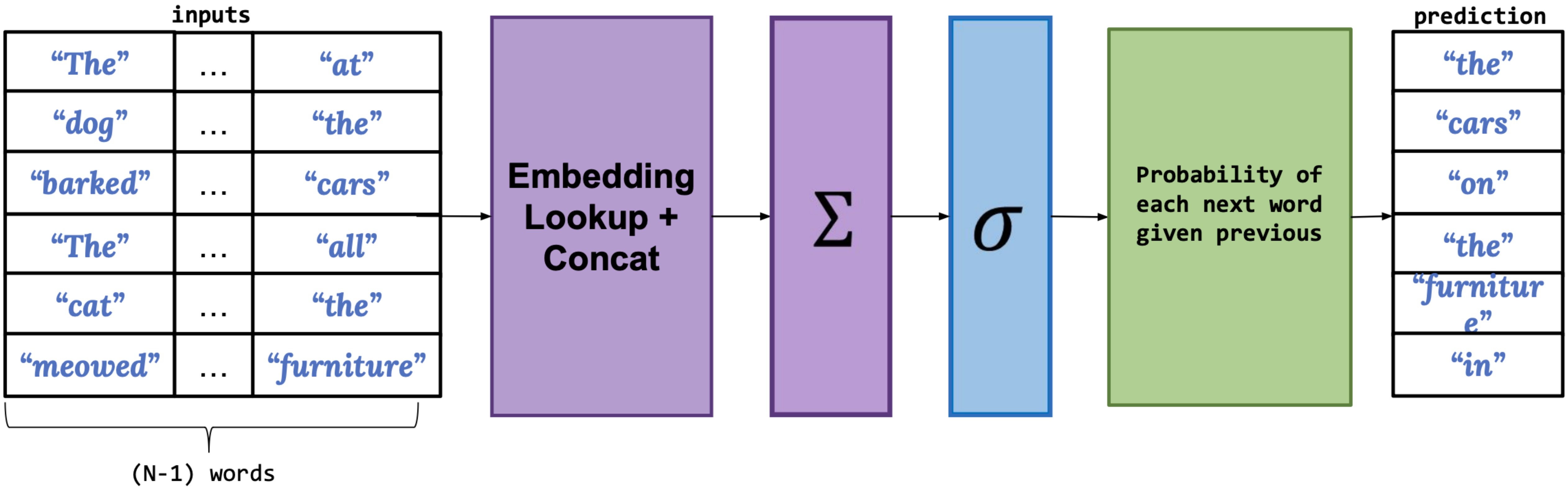
What size do we need the linear layer to be in order to map:

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



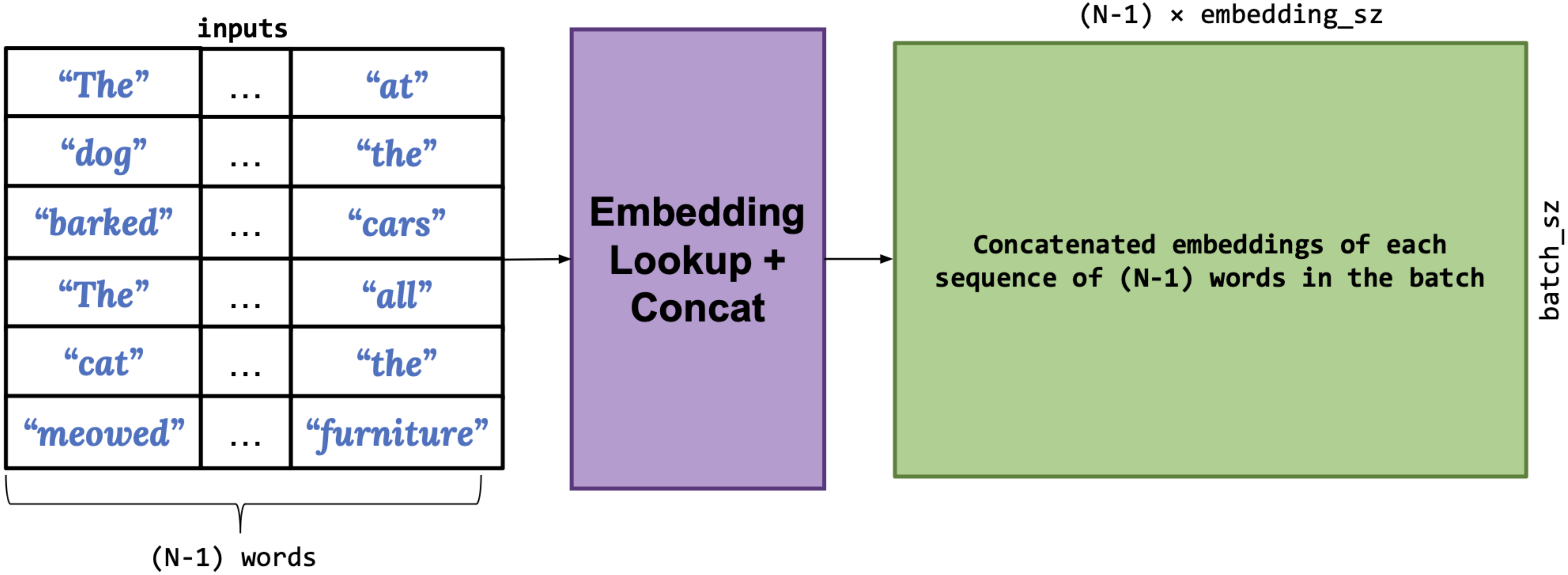
Size of Feed Forward N-gram Model

So what happens in the N-gram case?



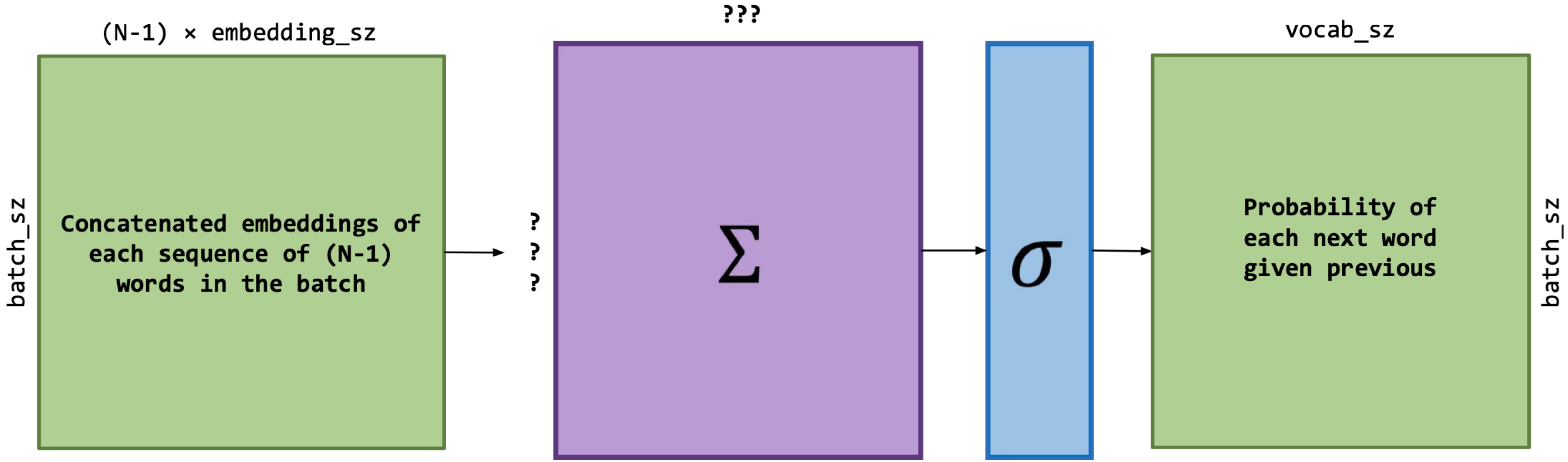
Size of Feed Forward N-gram Model

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 <u>barked</u> 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 <u>cats</u>.”

(***“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?

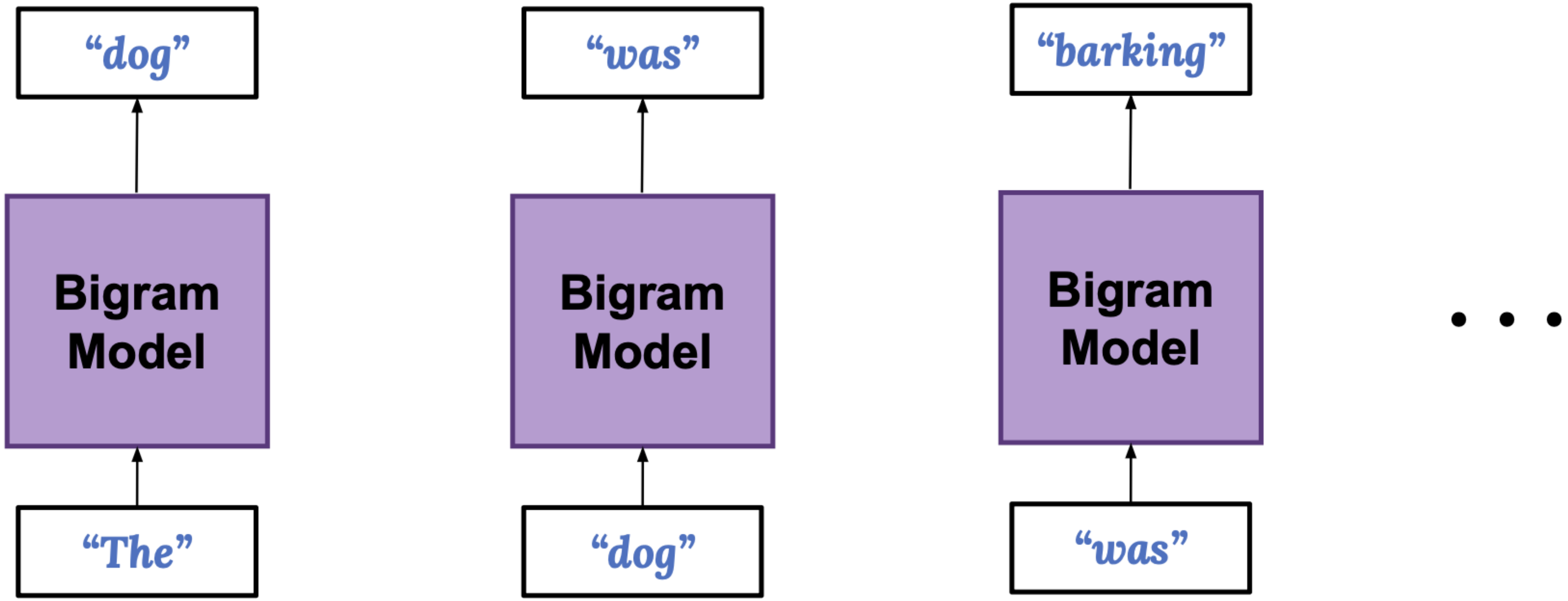
1. As the size of **N** increases, the number of weights needed for the linear layer becomes far too large.

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

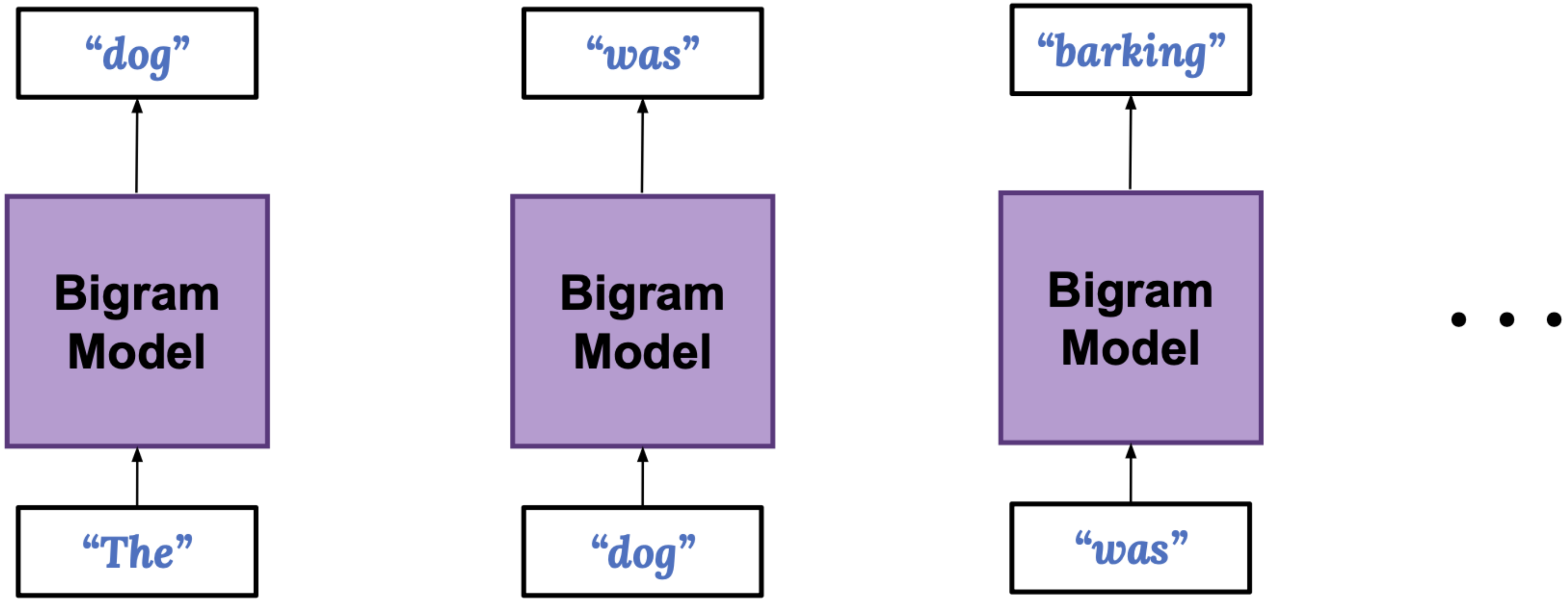
Let's revisit the bigram model and see several iterations of prediction using a bigram model:



New Approach

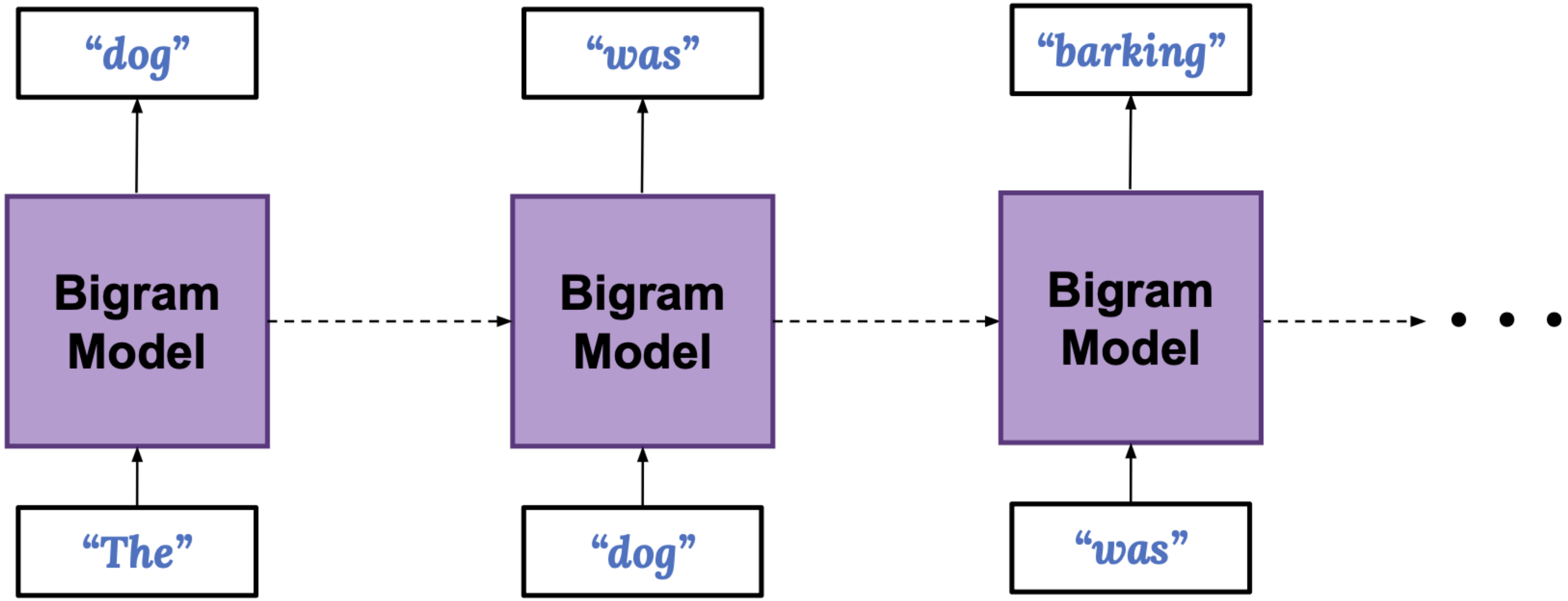
Any ideas?

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



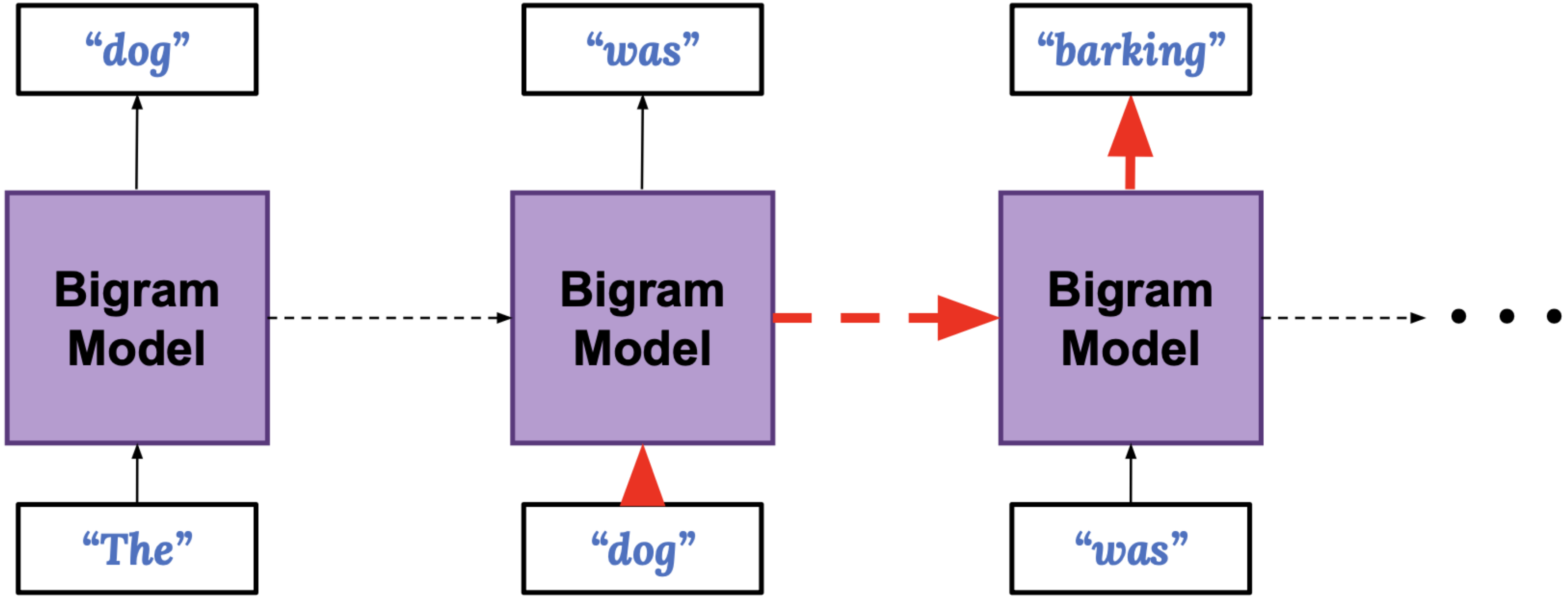
New Approach

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



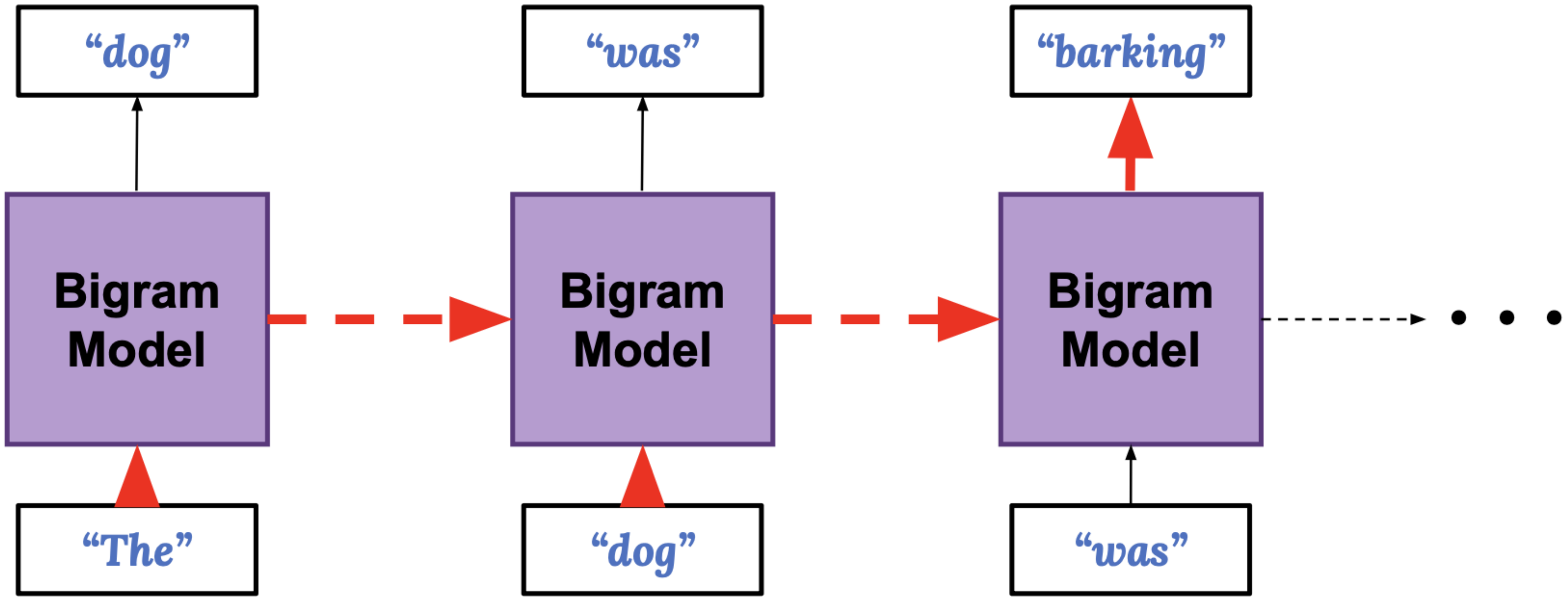
New Approach

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!

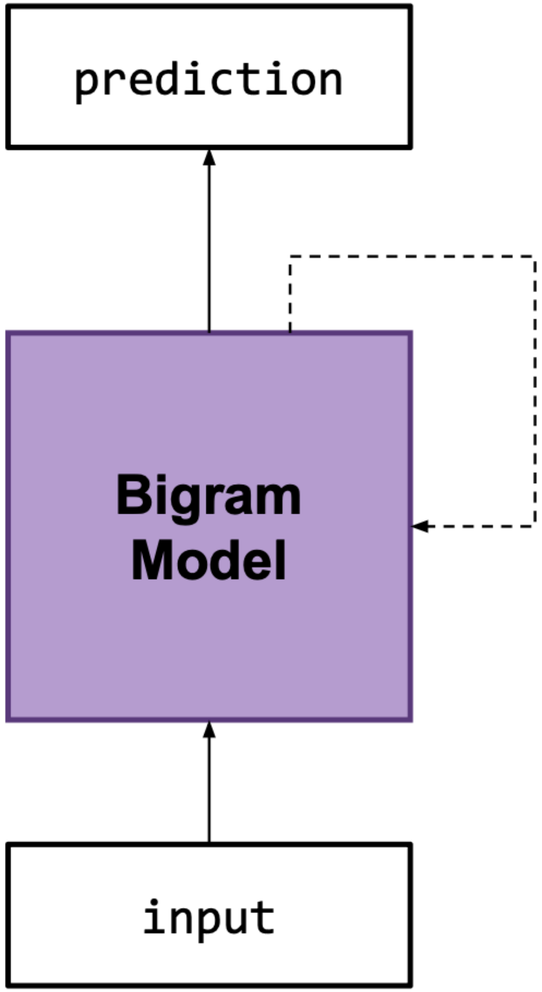


New Approach

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.

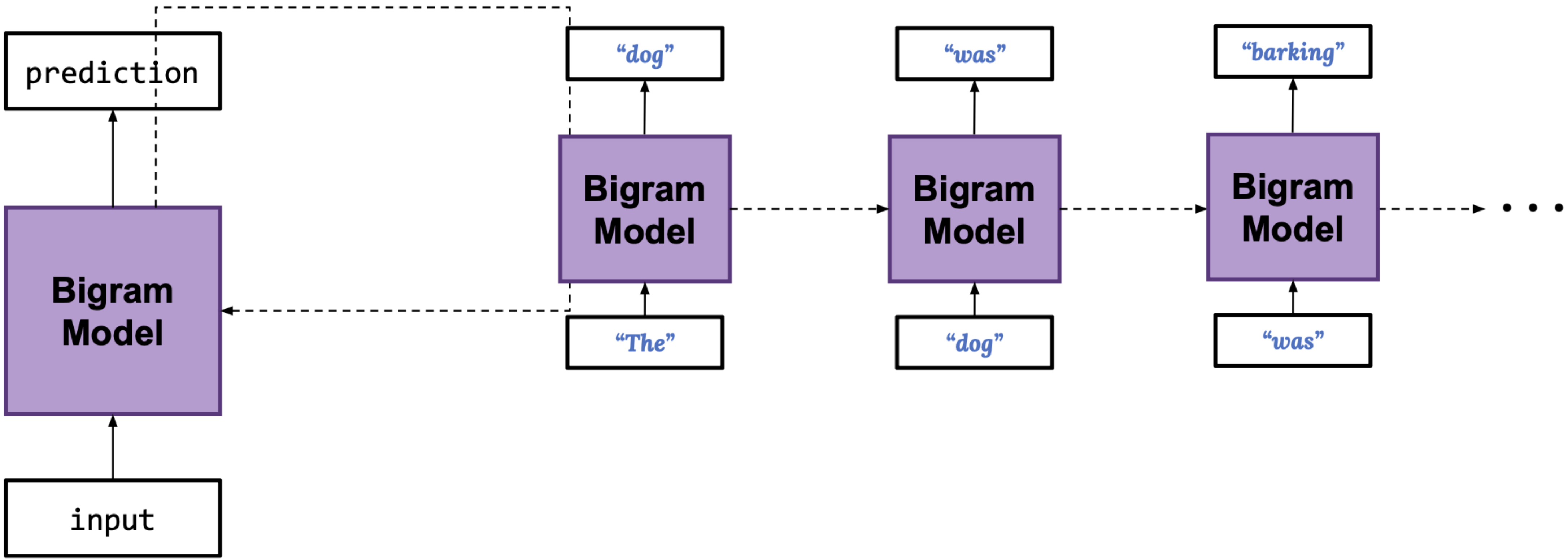
We call the previous representation the "unrolled" representation.



Different views of recurrent models

Recurrent view

Unrolled view



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.

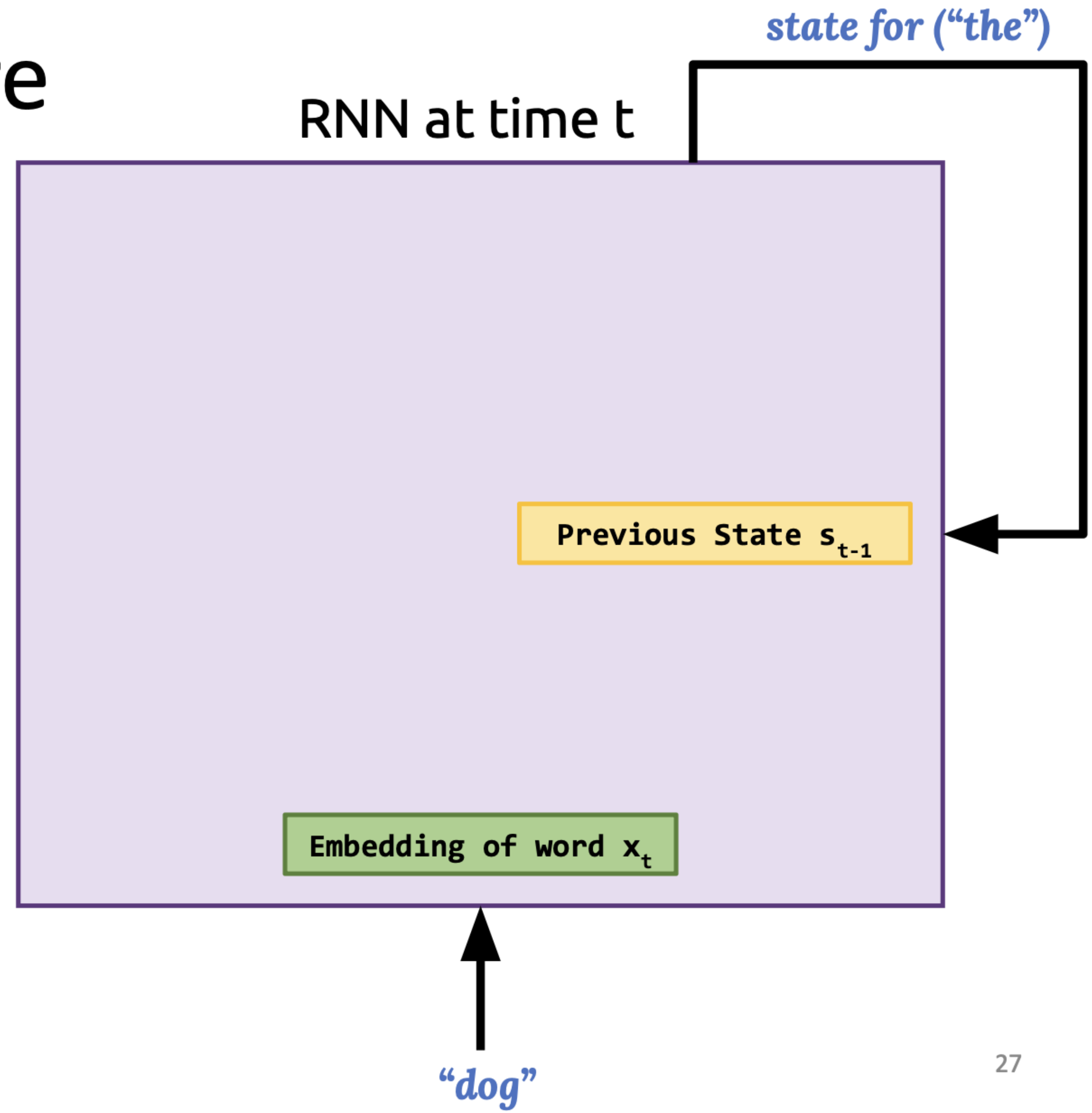
They can be used to process sequence data with relatively low model complexity when compared to feed forward models.

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

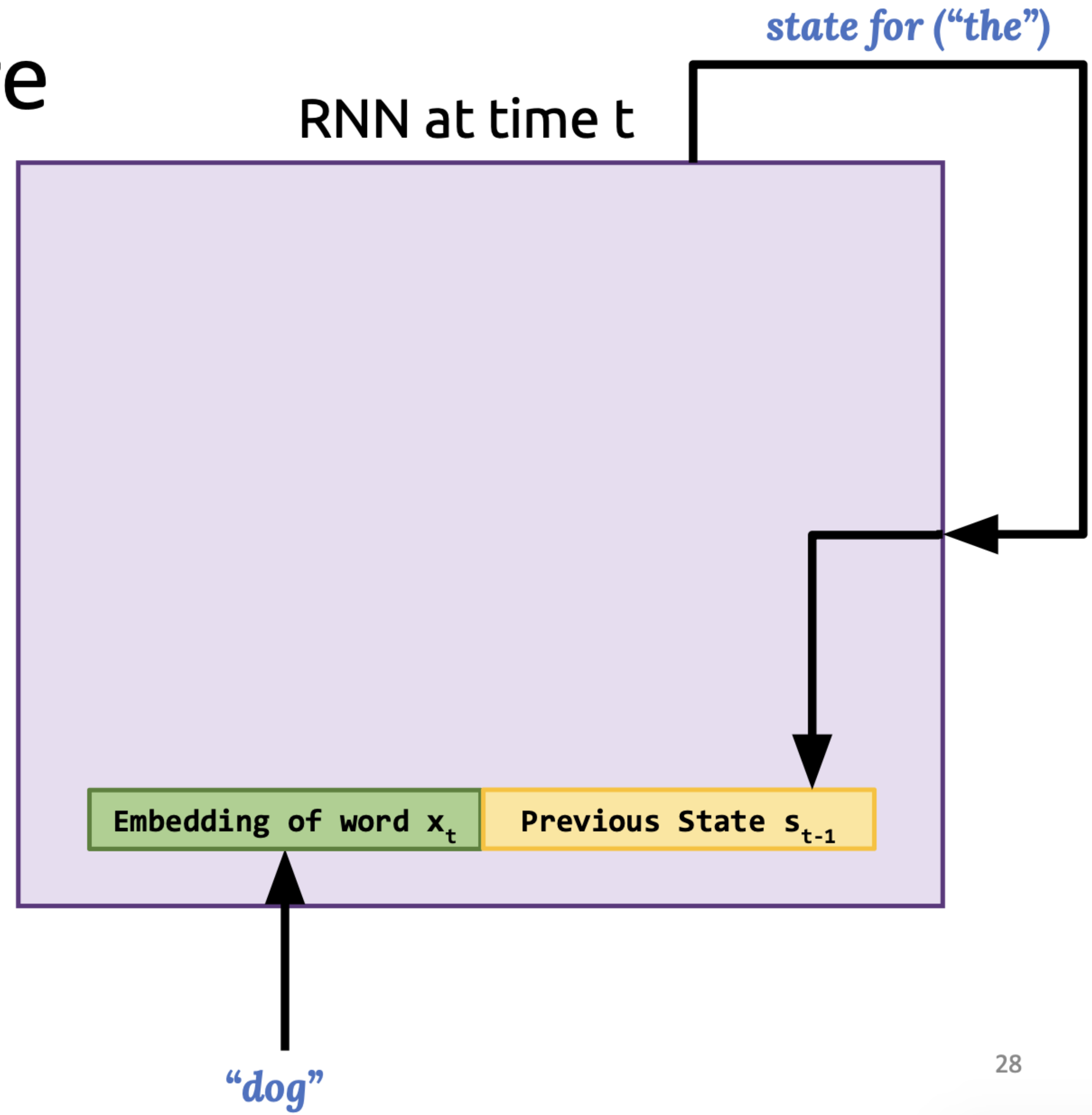
At each step of our RNN, we will get an input word, and a state vector from the previous cell.



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.

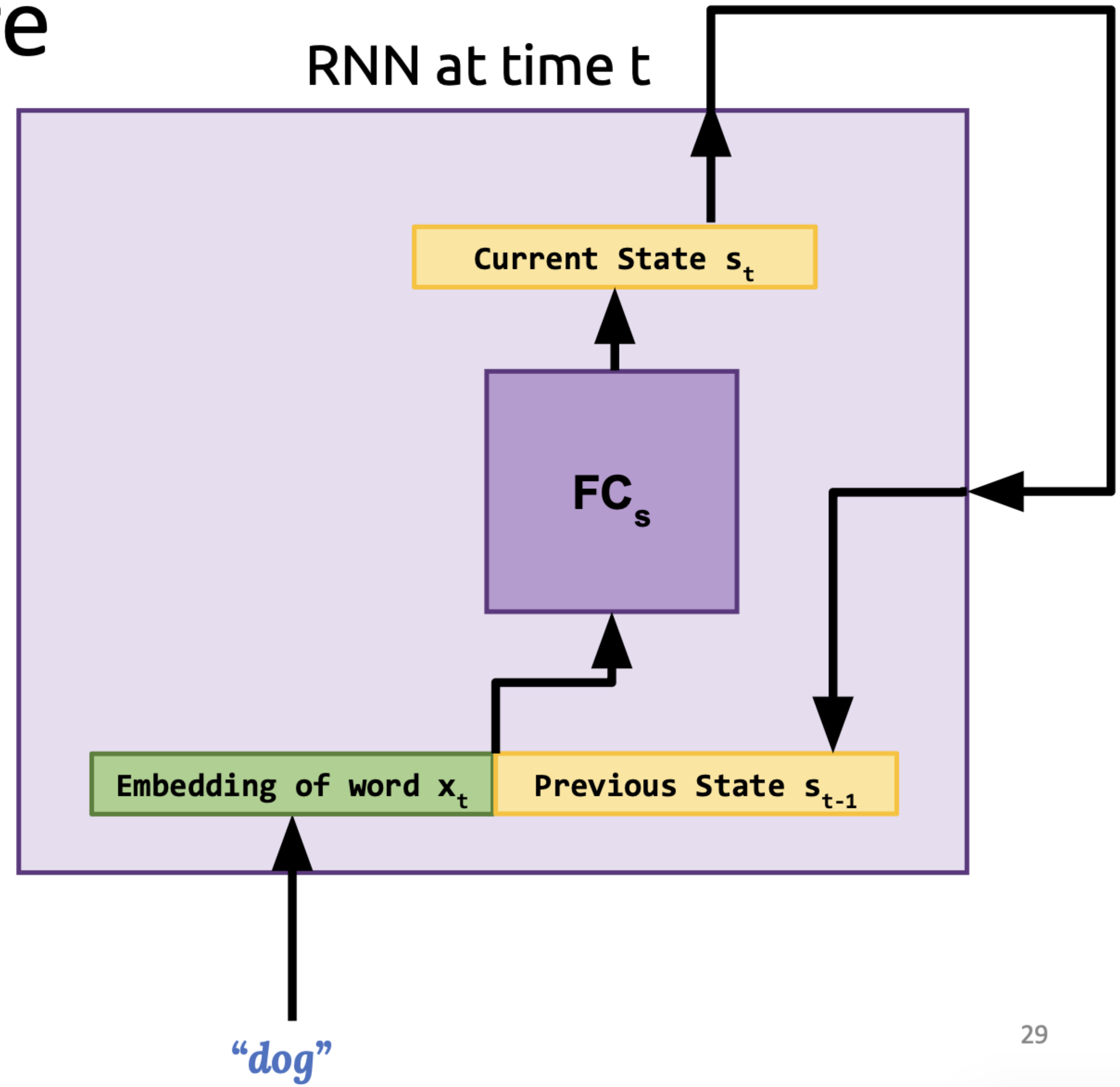


RNN Cell Architecture

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



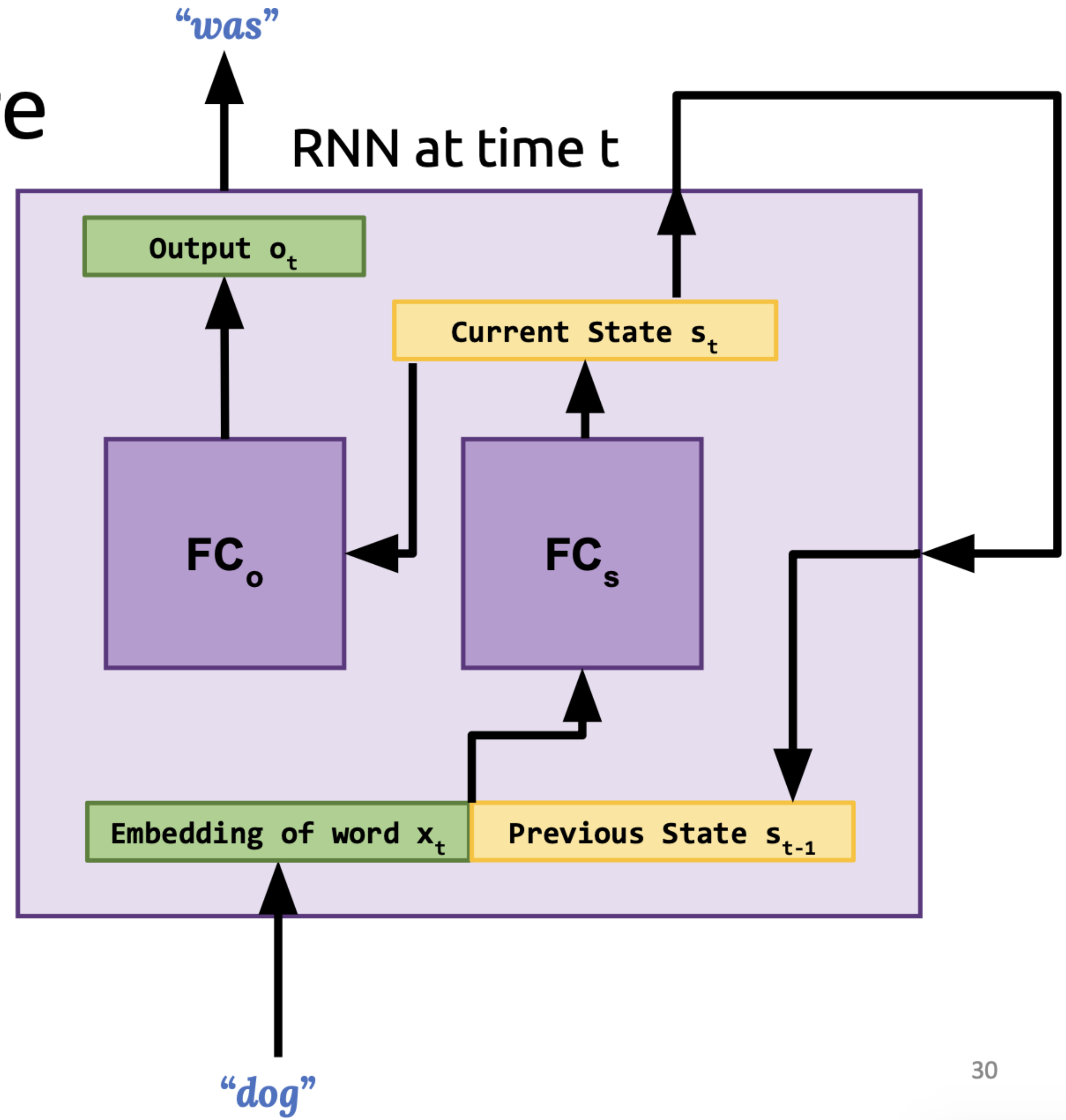
RNN Cell Architecture

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.

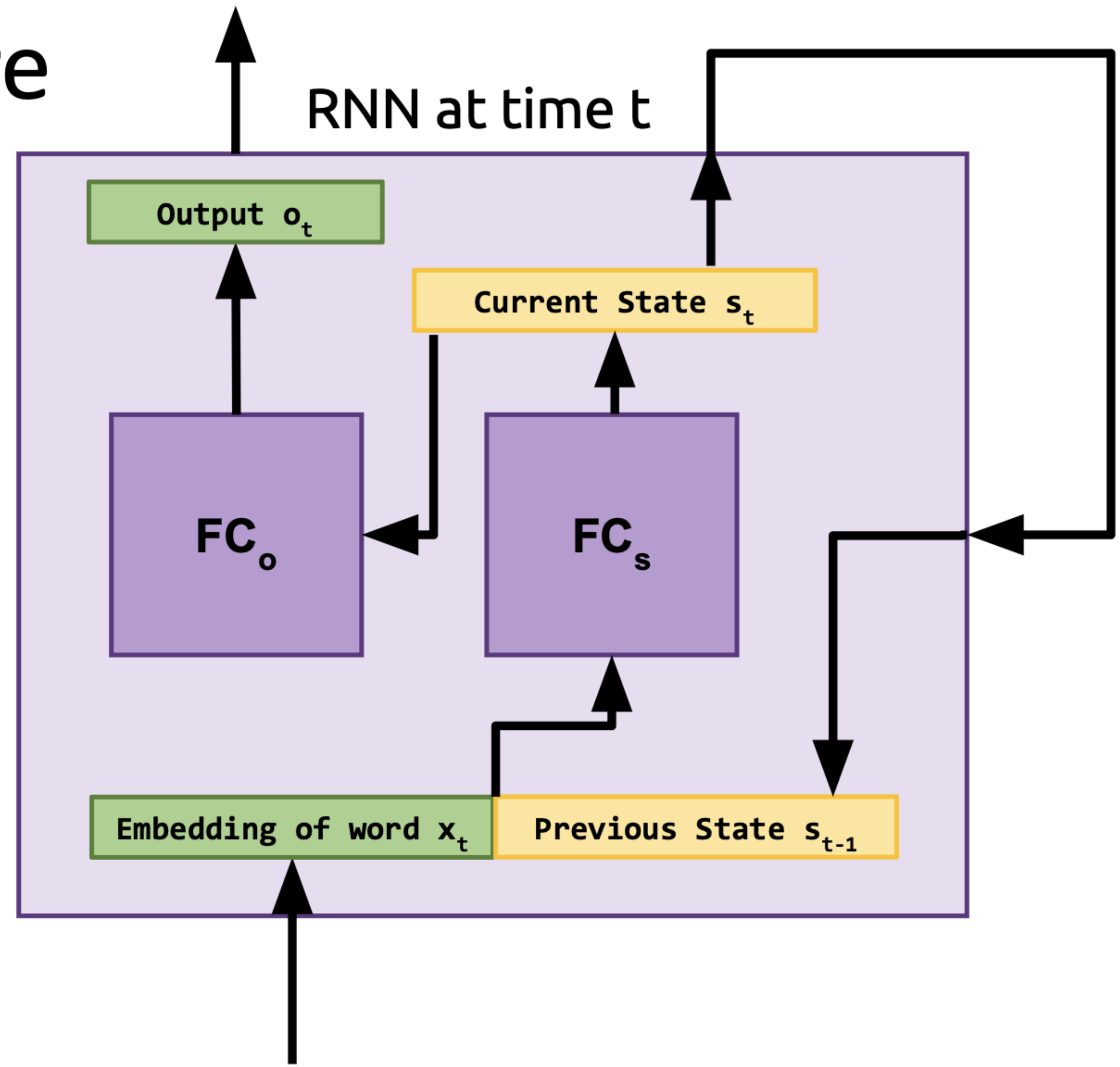


RNN Cell Architecture

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

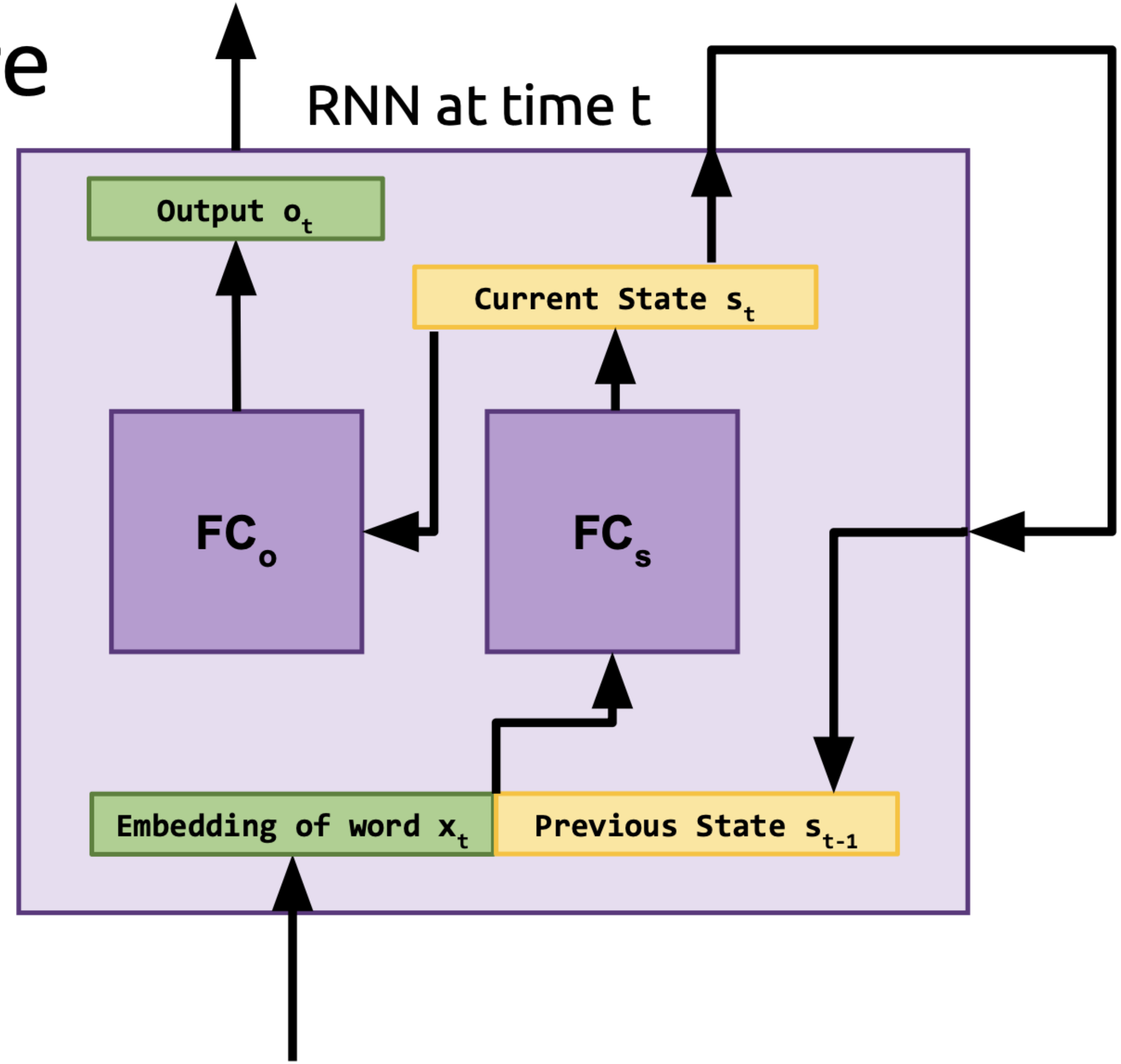
We can represent the RNN in with the following equations:

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

$$o_t = \boxed{\sigma}(s_t W_o + b_o)$$

Nonlinear activations (e.g. sigmoid, tanh)

Any questions?



RNN Cell Architecture

We can represent the RNN in with the following equations:

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

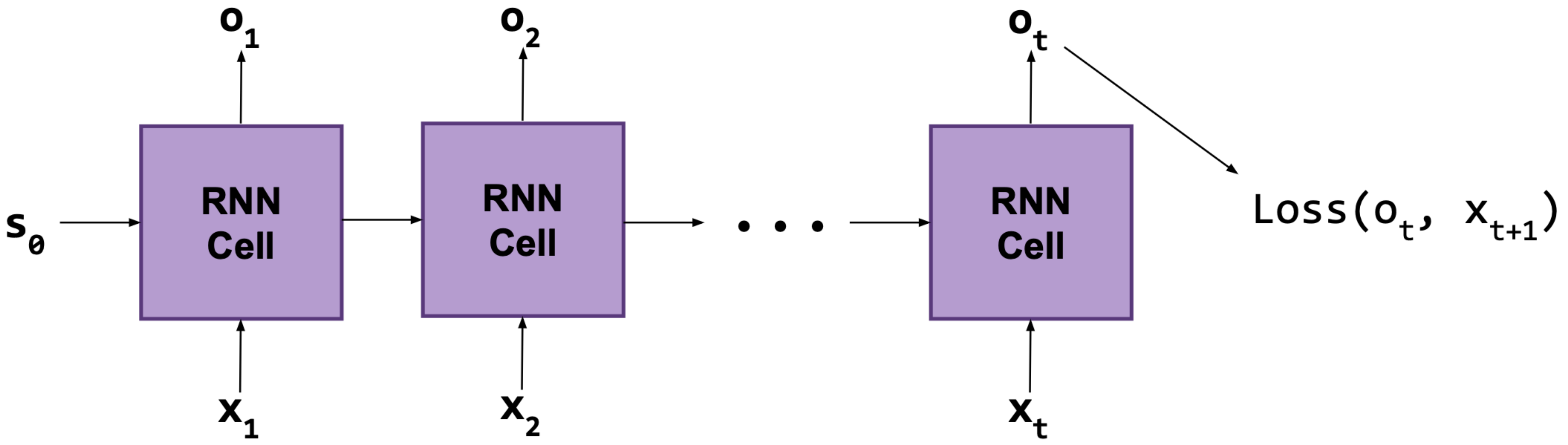
$$o_t = \sigma(s_t W_o + b_o)$$

This brings up an immediate question: **what is $\boldsymbol{s_0}$?**

Typically, we initialize s_0 to be a vector of zeros (i.e. “initially, there is no memory of any previous words”)

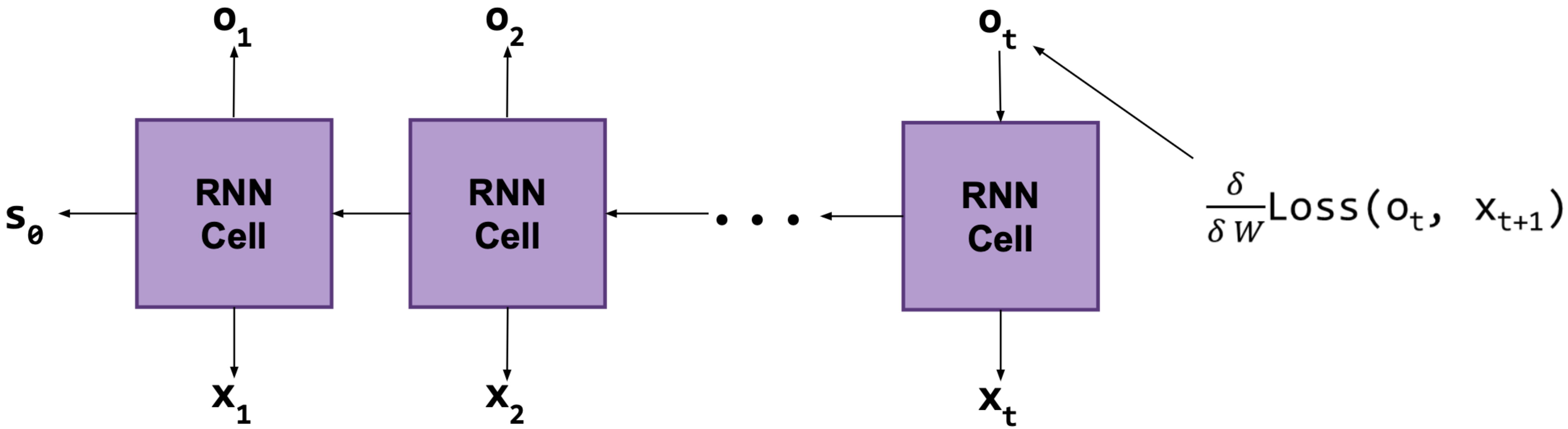
Training RNNs

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

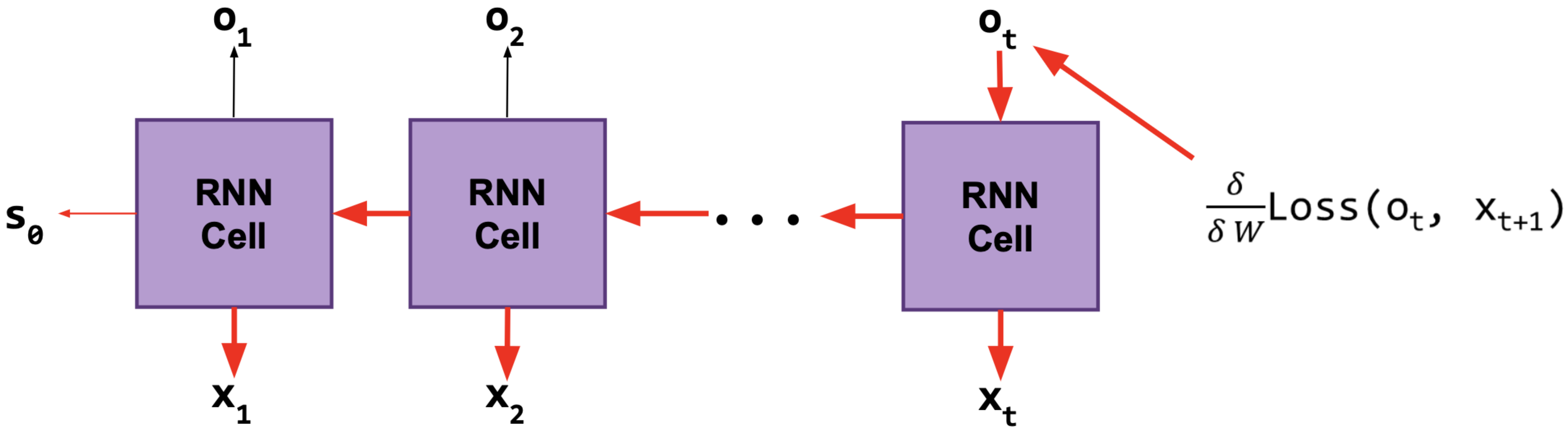
But what happens when we differentiate the loss and backpropagate?



Training RNNs

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

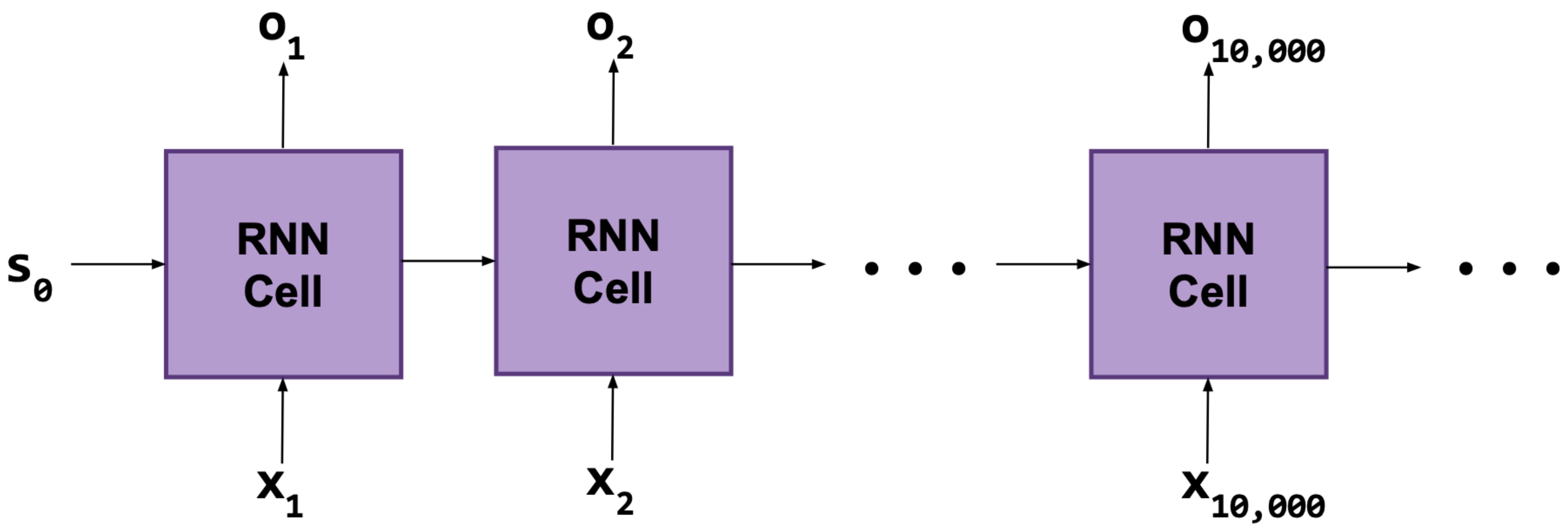
We call this *backpropagation through time.*



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`

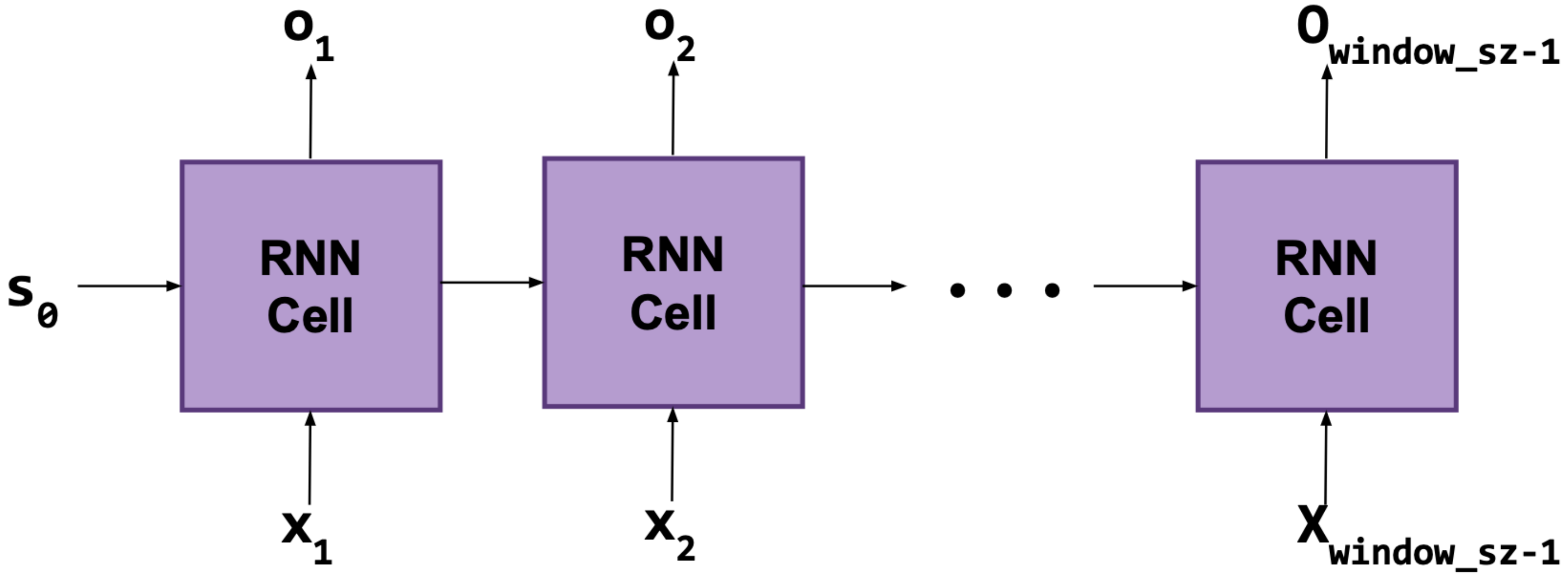
The new shape of our data should be:

```
(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

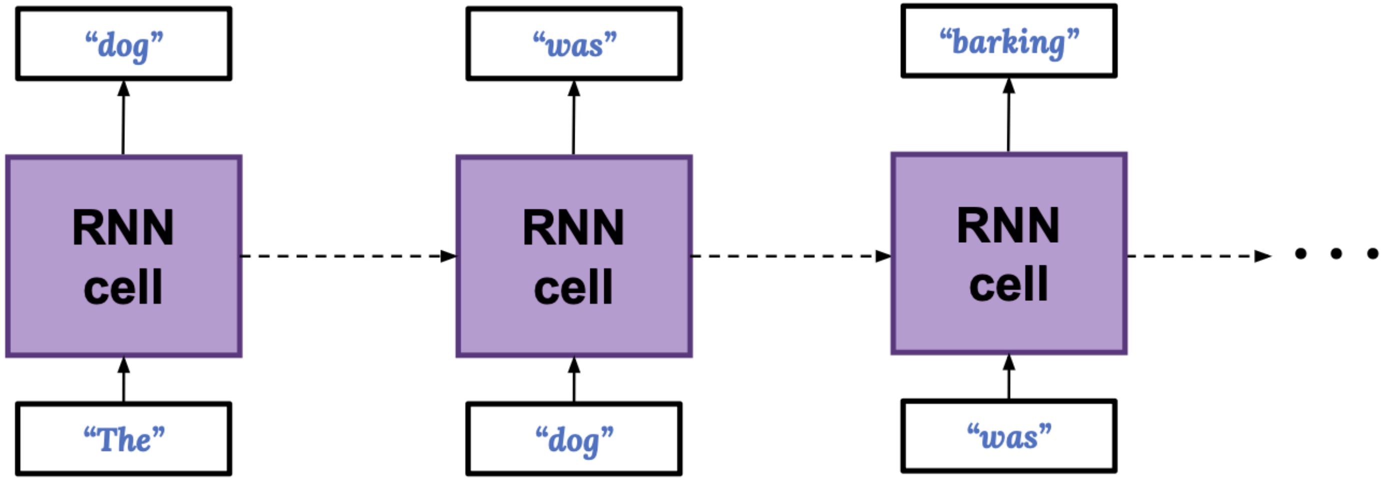


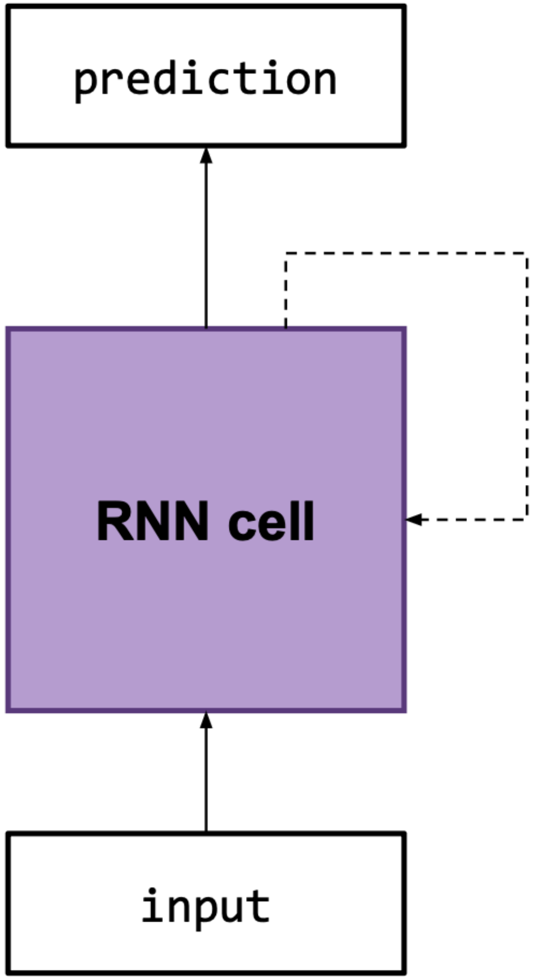
Any questions?

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

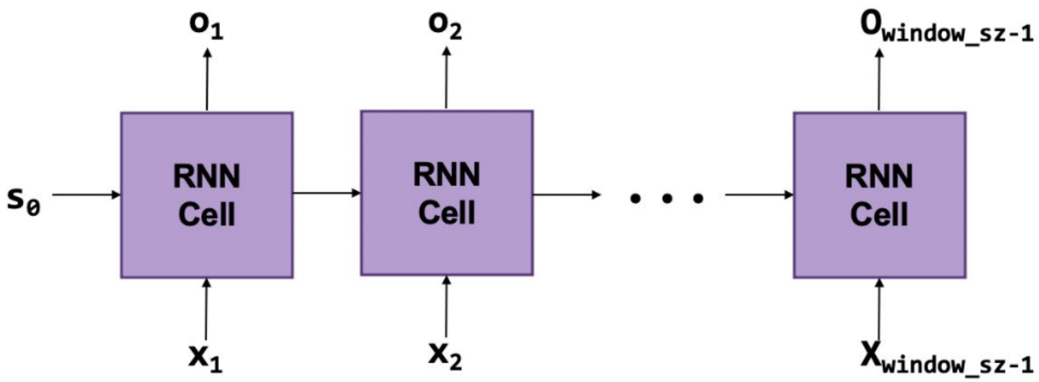
RNNs in Tensorflow

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
```