Start thinking about your course project!

CSCI 1470/2470
Spring 2023

Ritambhara Singh

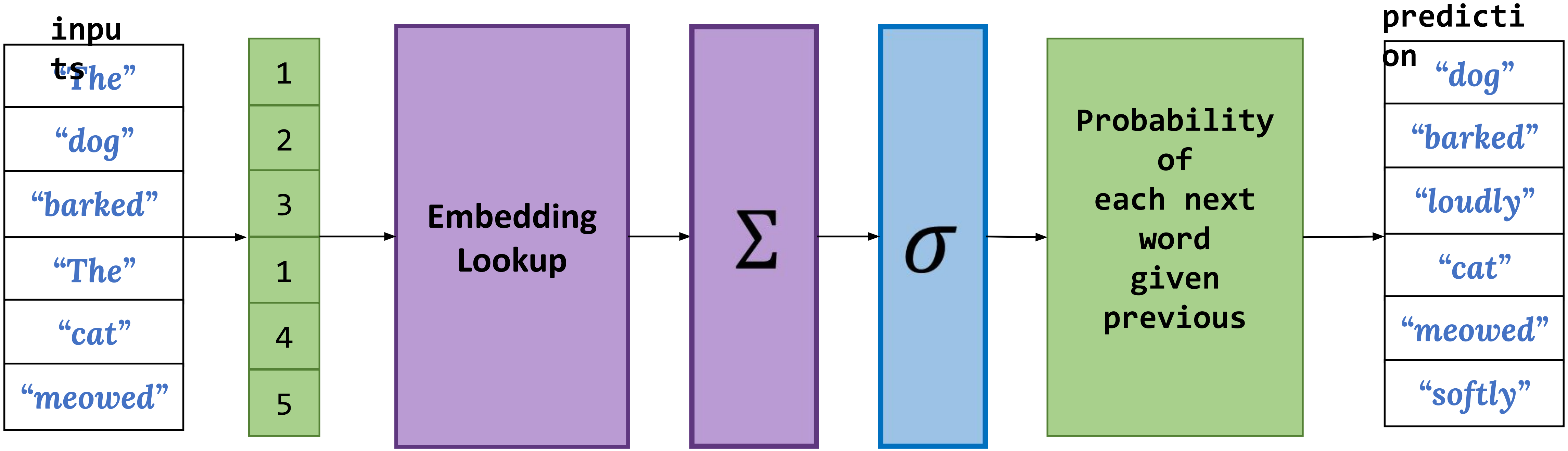
March 06, 2023

Monday

Deep Learning

DALL-E 2 prompt “a painting of deep underwater with a yellow submarine in the bottom right corner”

Review: Bigram Language Model Architecture

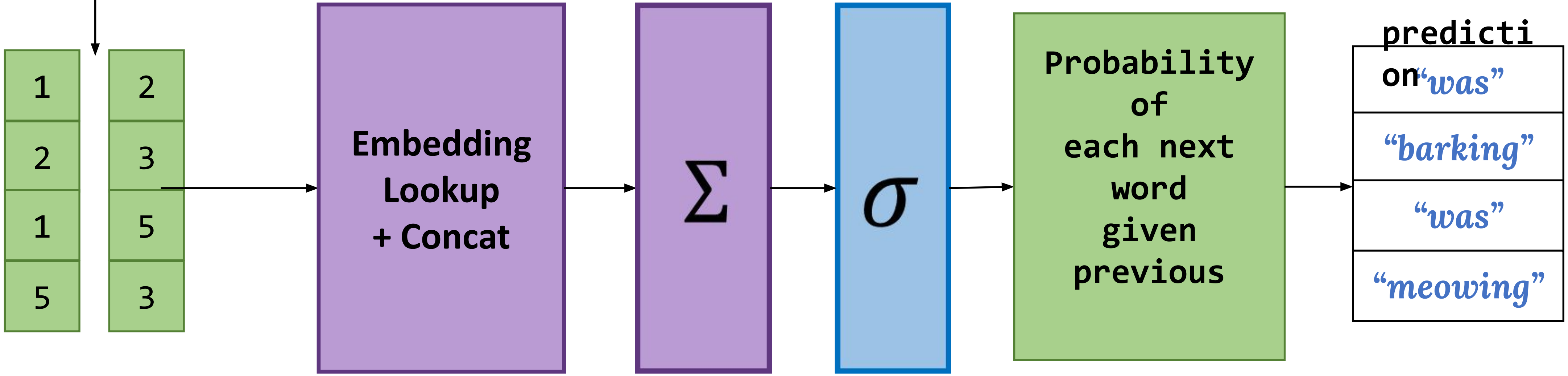


Review: Complete Trigram Language Model

inpu

"The"	"dog"
"dog"	"was"
"The"	"cat"
"cat"	"was"

"What is the tradeoff?"
~a wise student

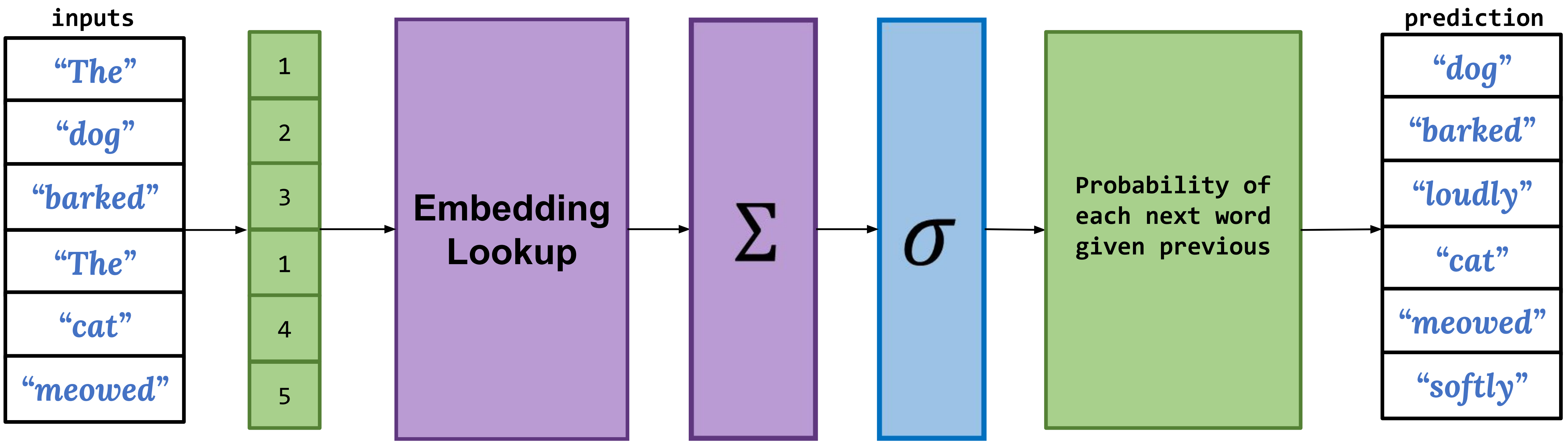


Limitations of the N-gram model

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

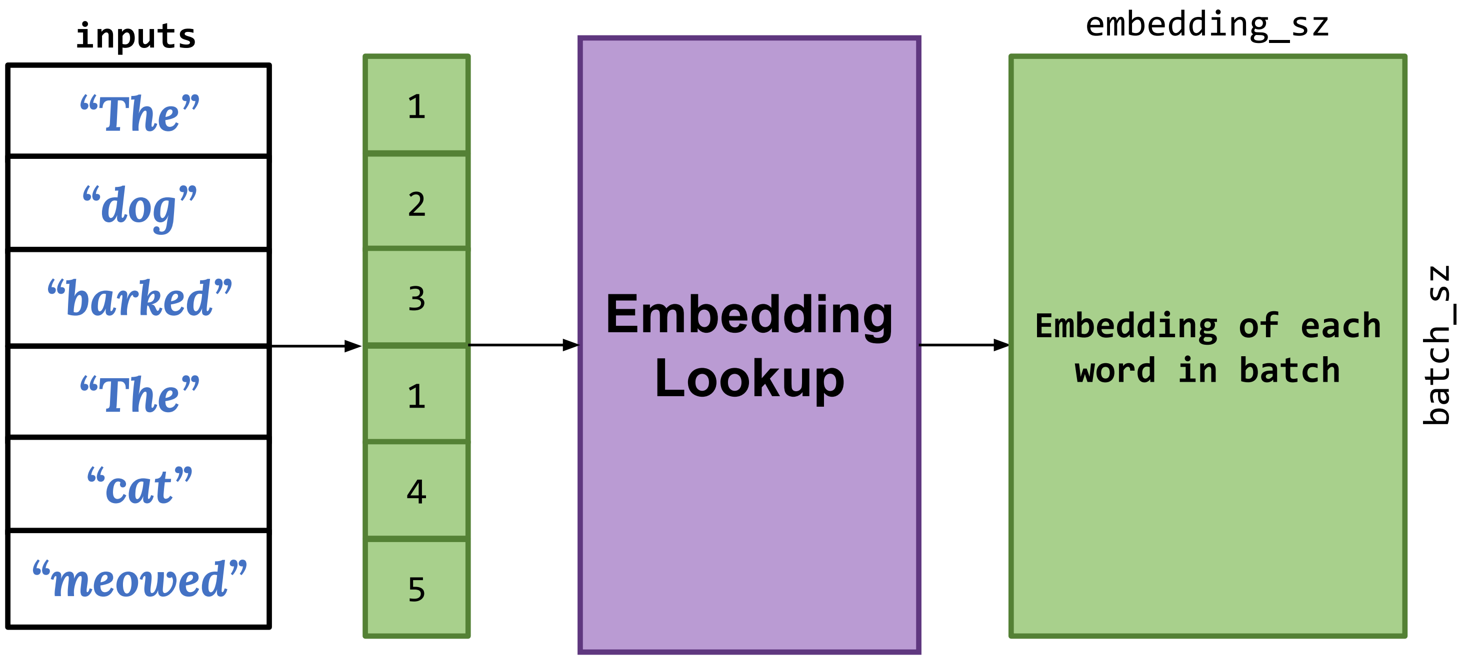
Size of Feed Forward bigram Model

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



Size of Feed Forward bigram Model

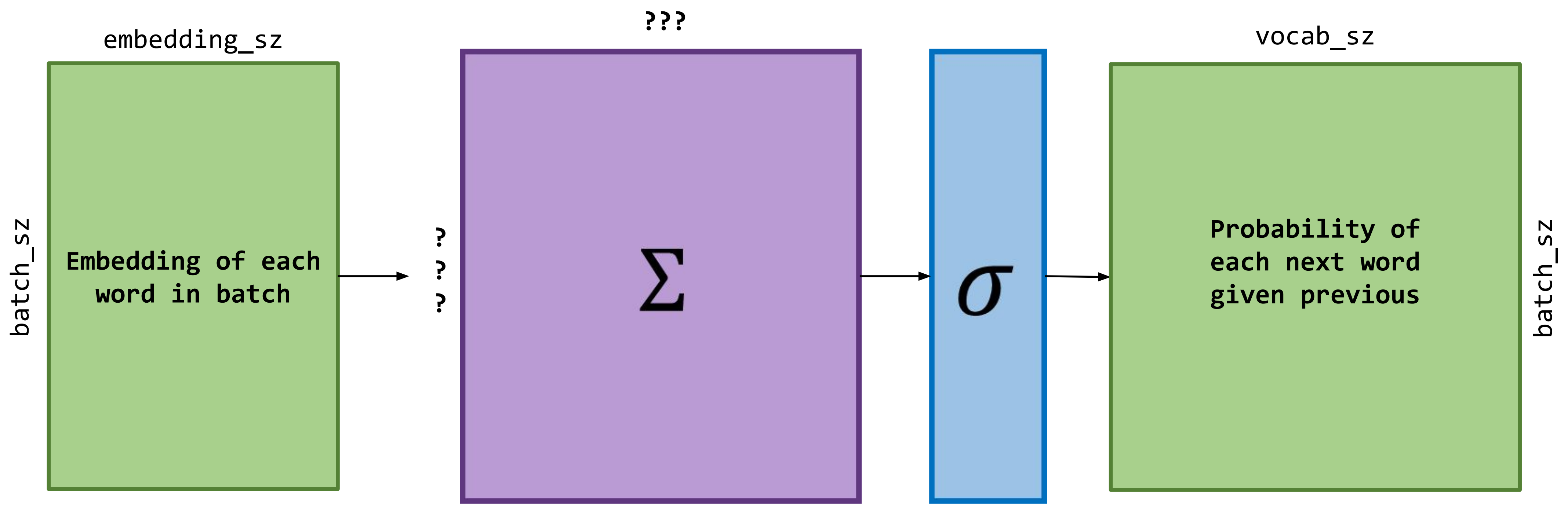
To preform embedding lookup on our entire batch, we just need one embedding matrix of size: `(vocab_sz, embedding_sz)`



Size of Feed Forward bigram Model

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

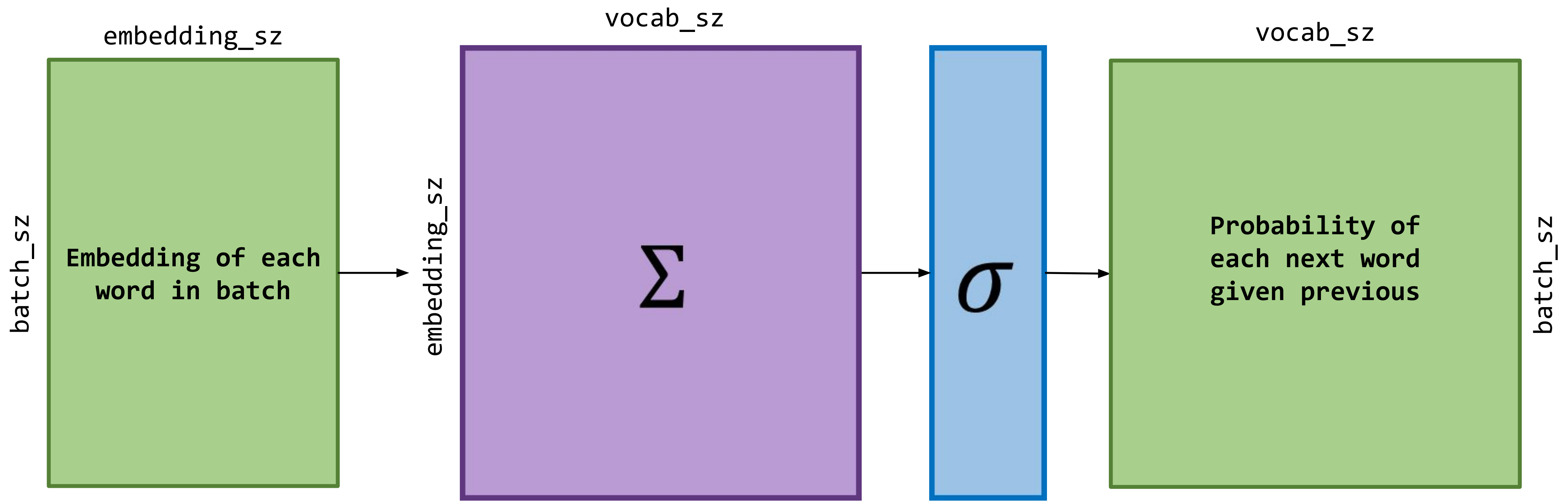
`(batch_sz, embedding_sz) × (???, ???) → (batch_sz, vocab_sz)`



Size of Feed Forward bigram Model

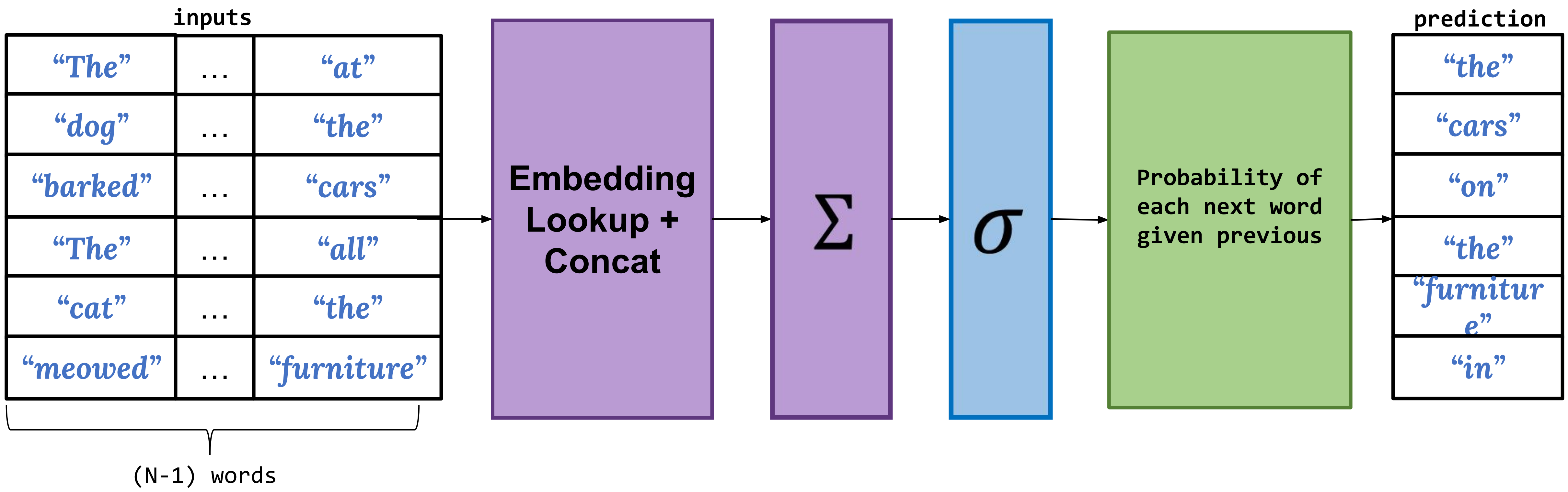
What size do we need the linear layer to be in order to map:

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



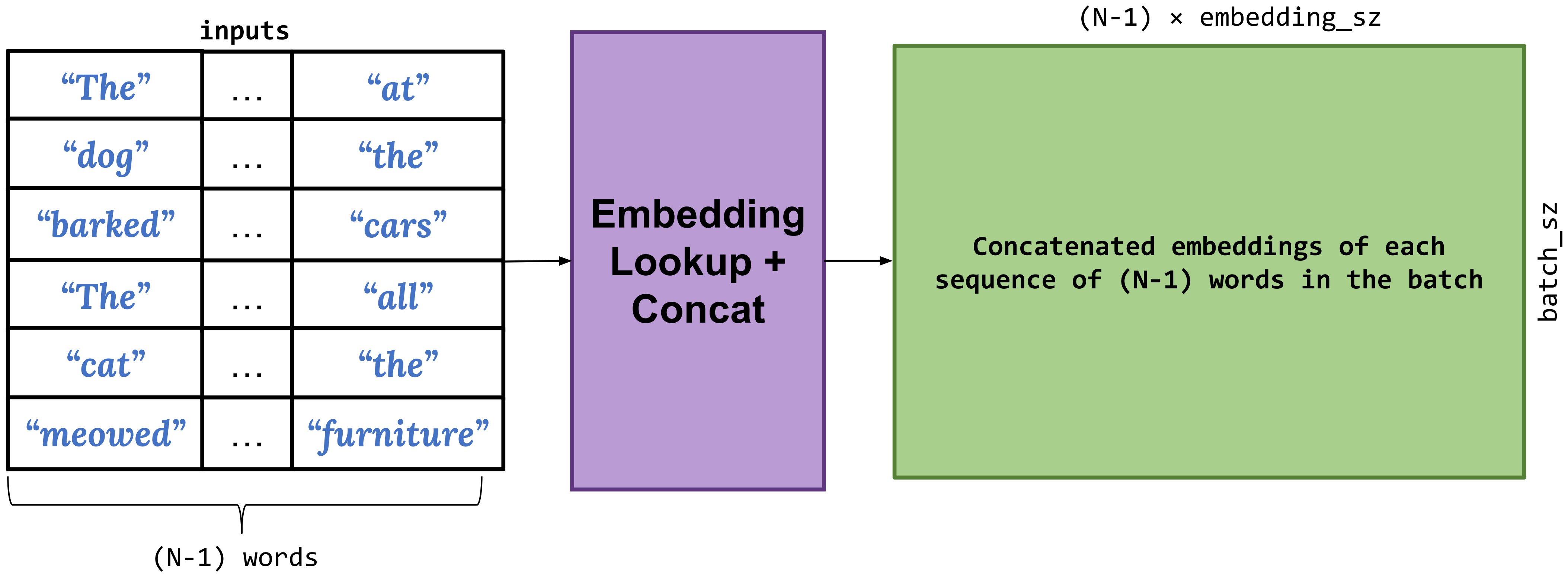
Size of Feed Forward N-gram Model

So what happens in the N-gram case?



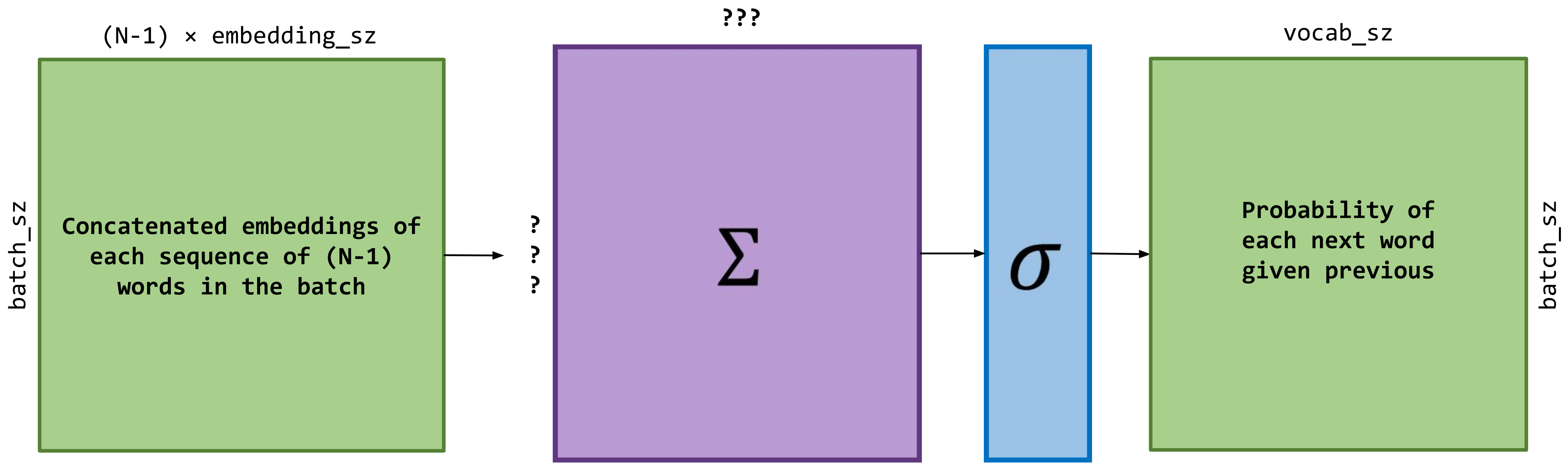
Size of Feed Forward N-gram Model

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



Size of Feed Forward N-gram Model

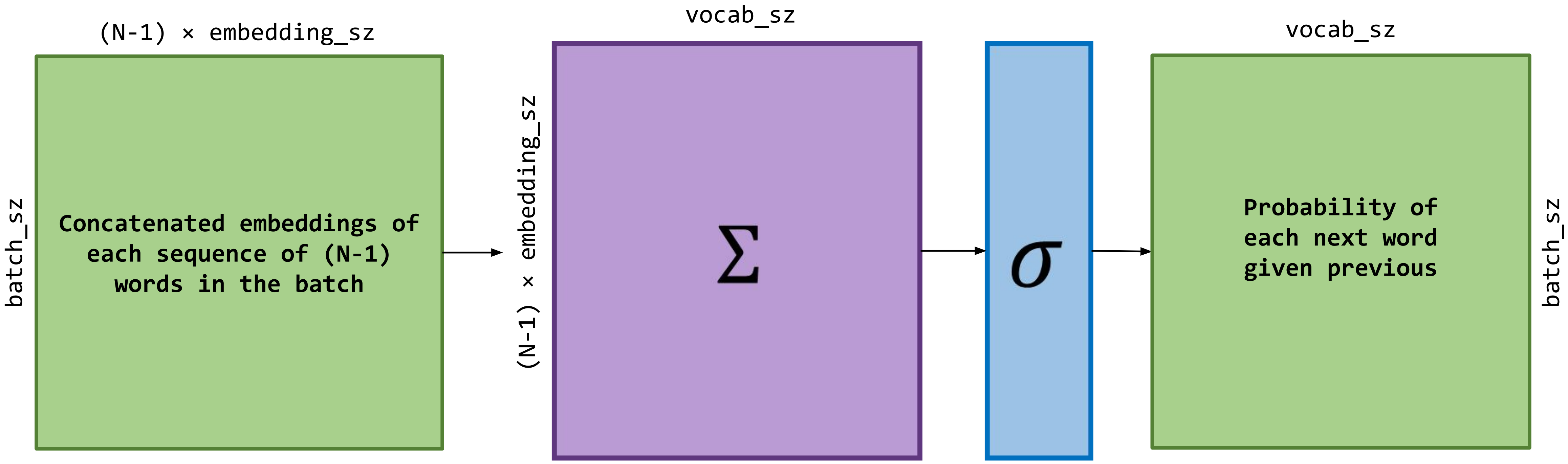
But what happens to our feed forward layer?



Can we see the problem now?

Size of Feed Forward N-gram Model

It needs to be size: `((N-1) × embedding_sz, vocab_sz)`
For every word, we add `(embedding_sz × vocab_sz)` more weights!



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.

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.

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 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?

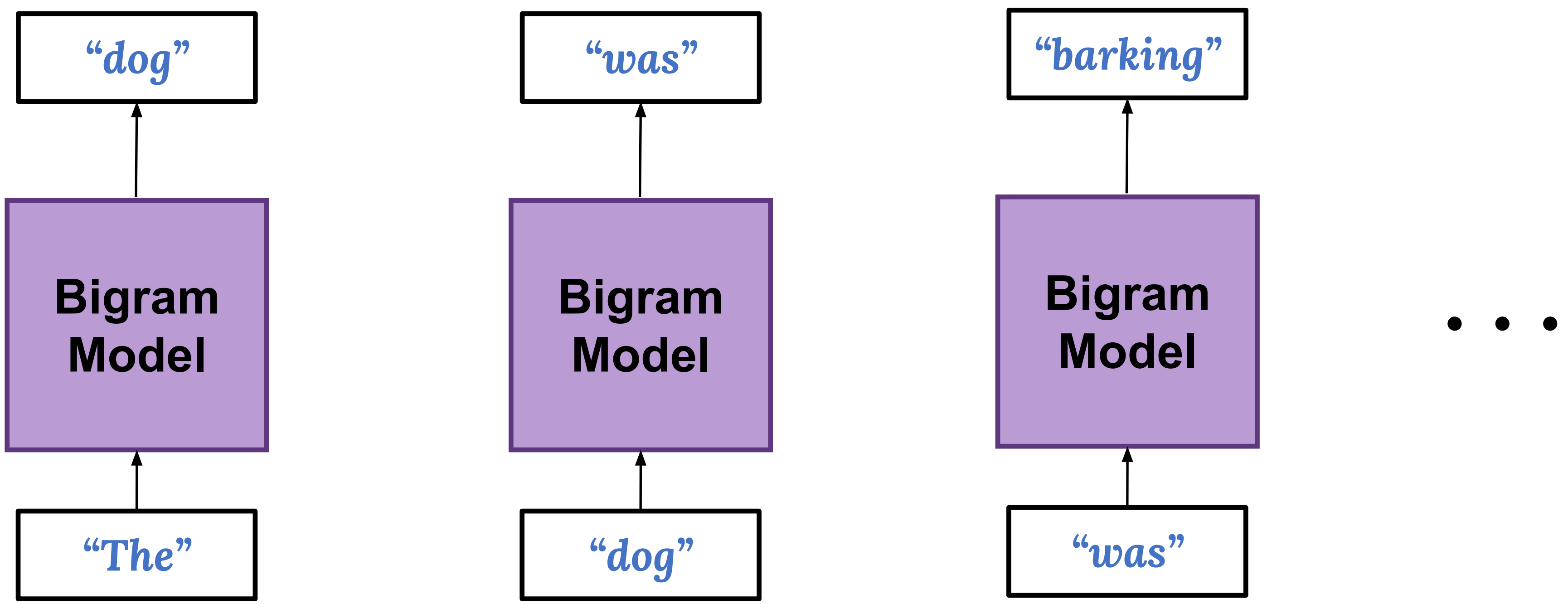
1. As the size of **N** increases, the number of weights needed for the linear layer becomes far too large.

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

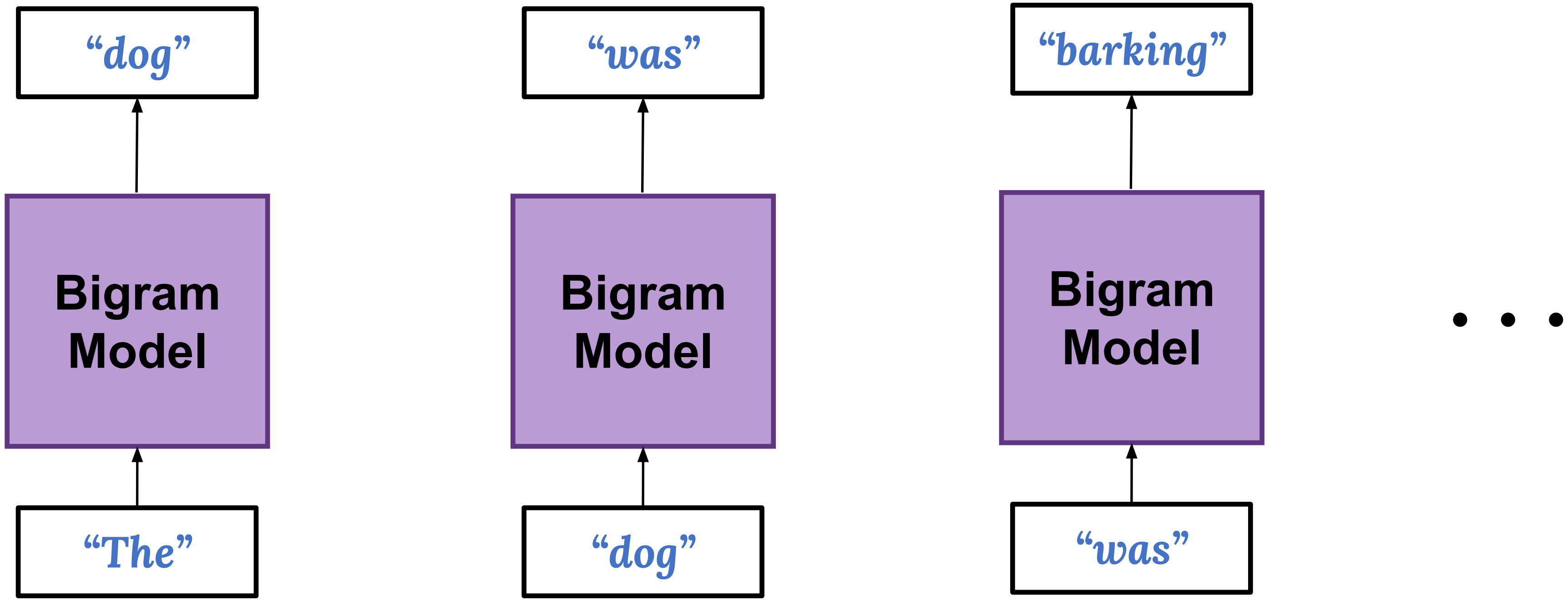
Let’s revisit the bigram model and see several iterations of prediction using a bigram model:



New Approach

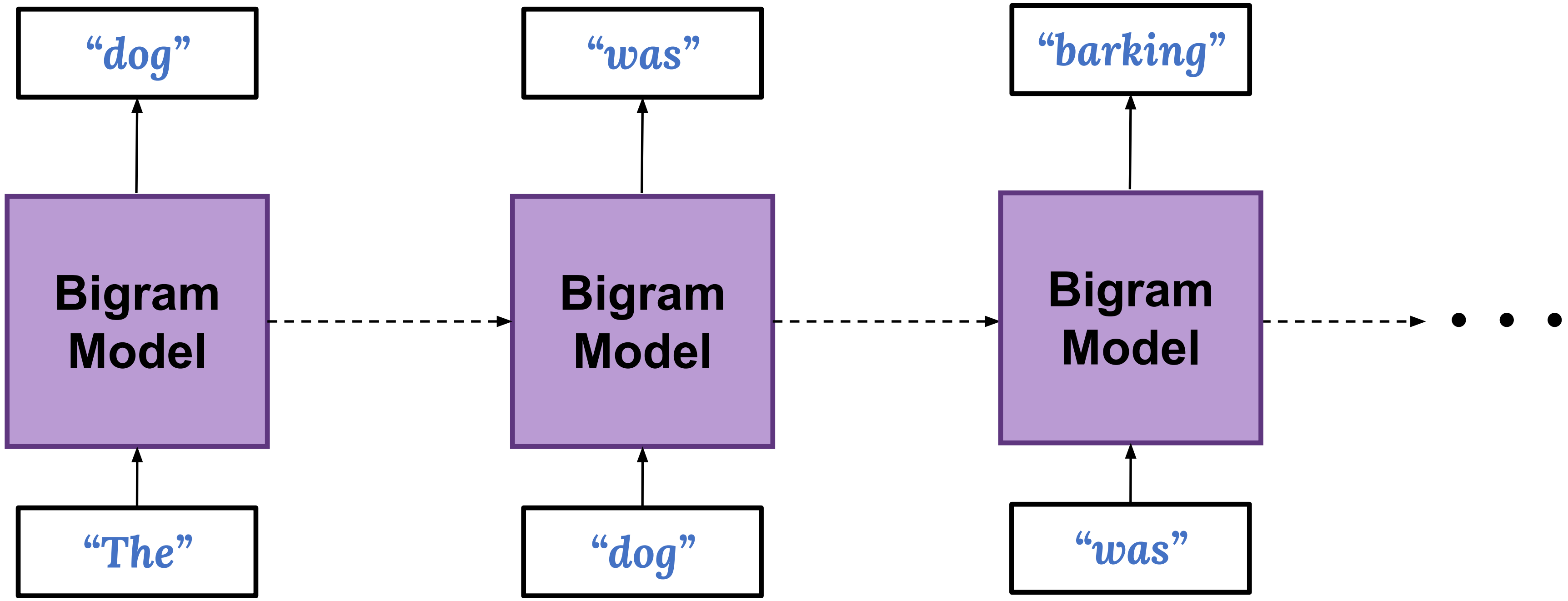
Any ideas?

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



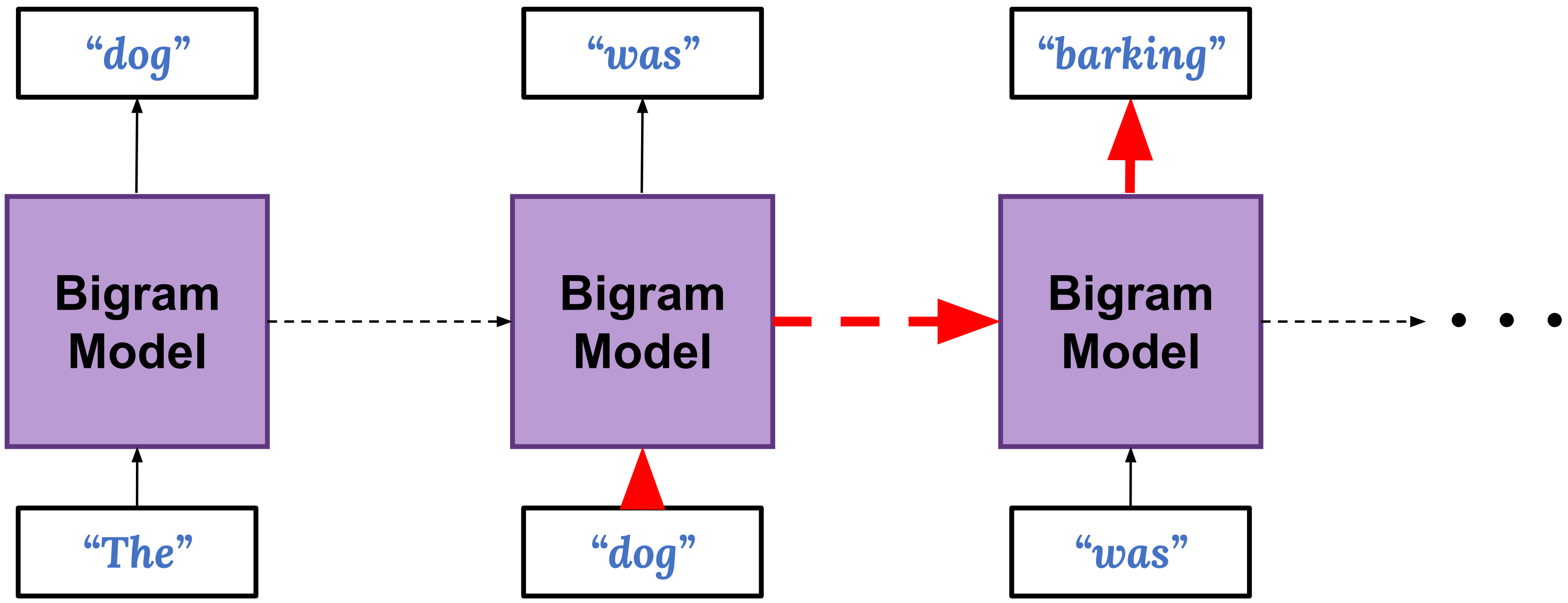
New Approach

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



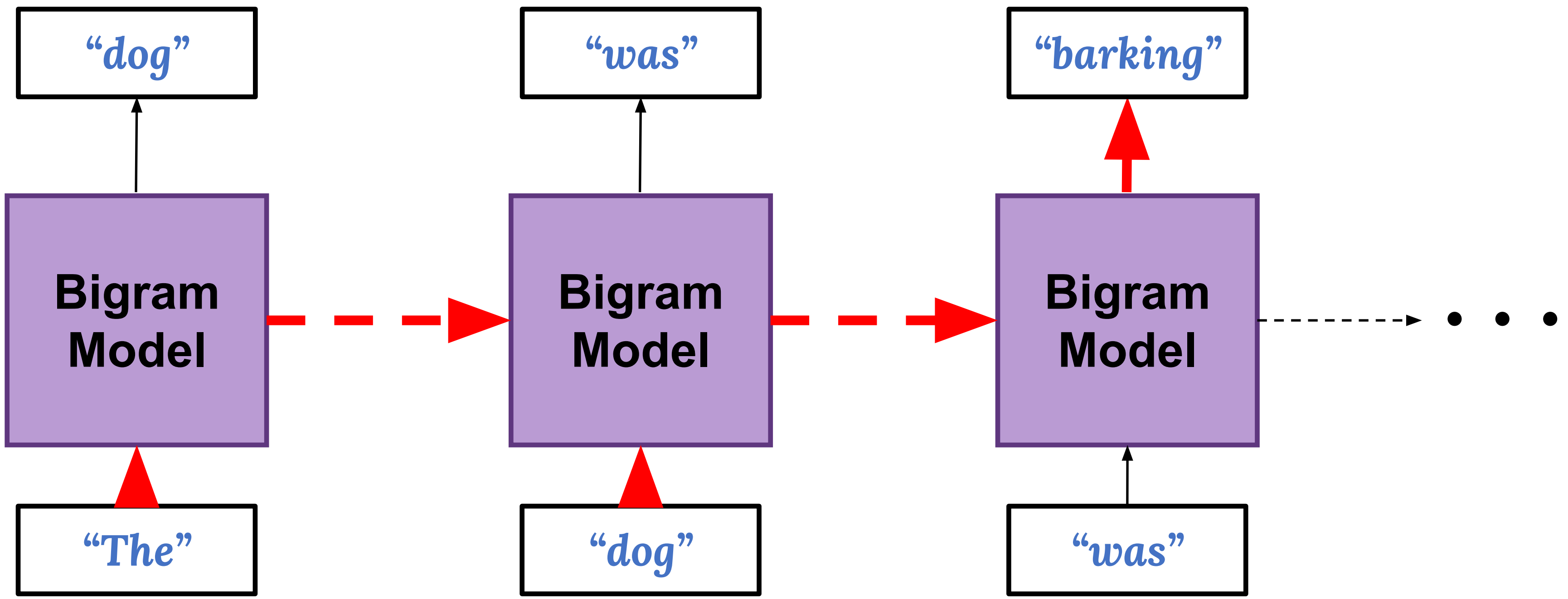
New Approach

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!

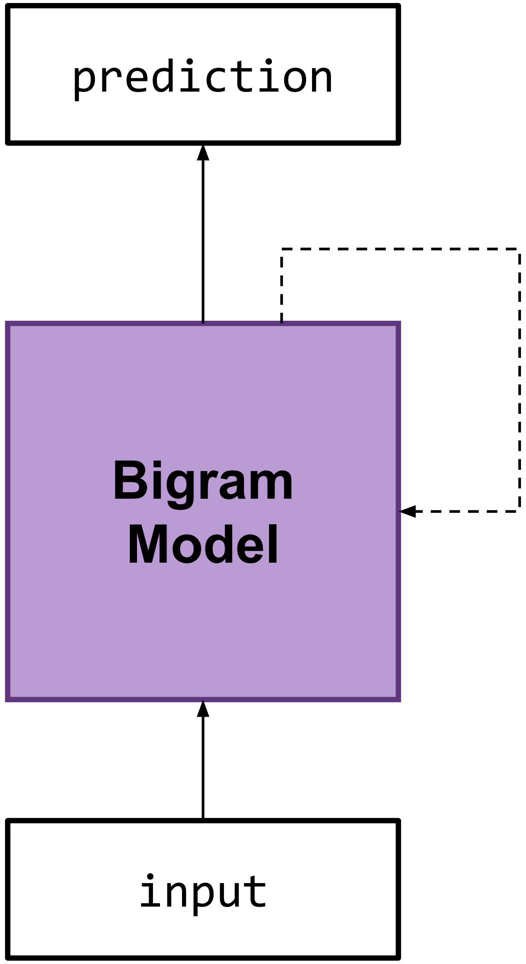


New Approach

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

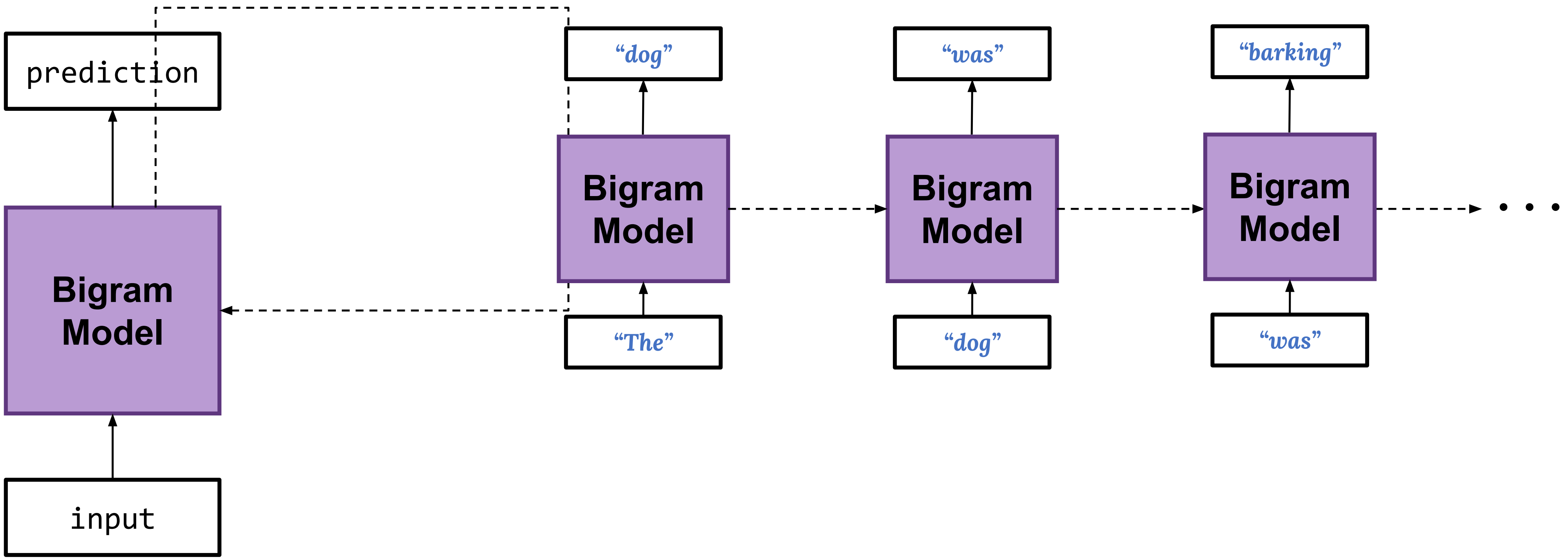
We call this connection a *recurrent* connection.

We call the previous representation the "unrolled" representation.



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.

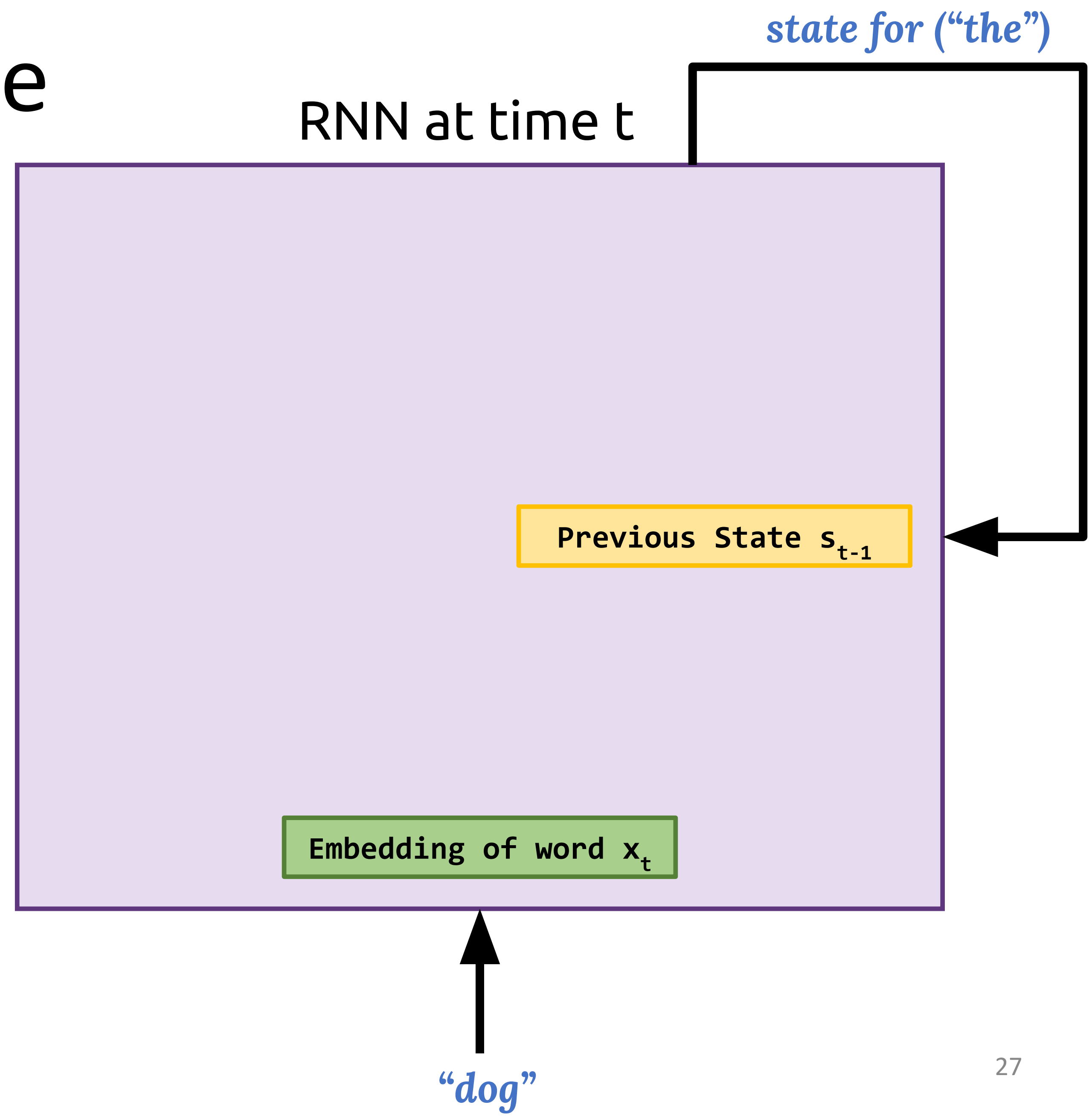
They can be used to process sequence data with relatively low model complexity when compared to feed forward models.

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

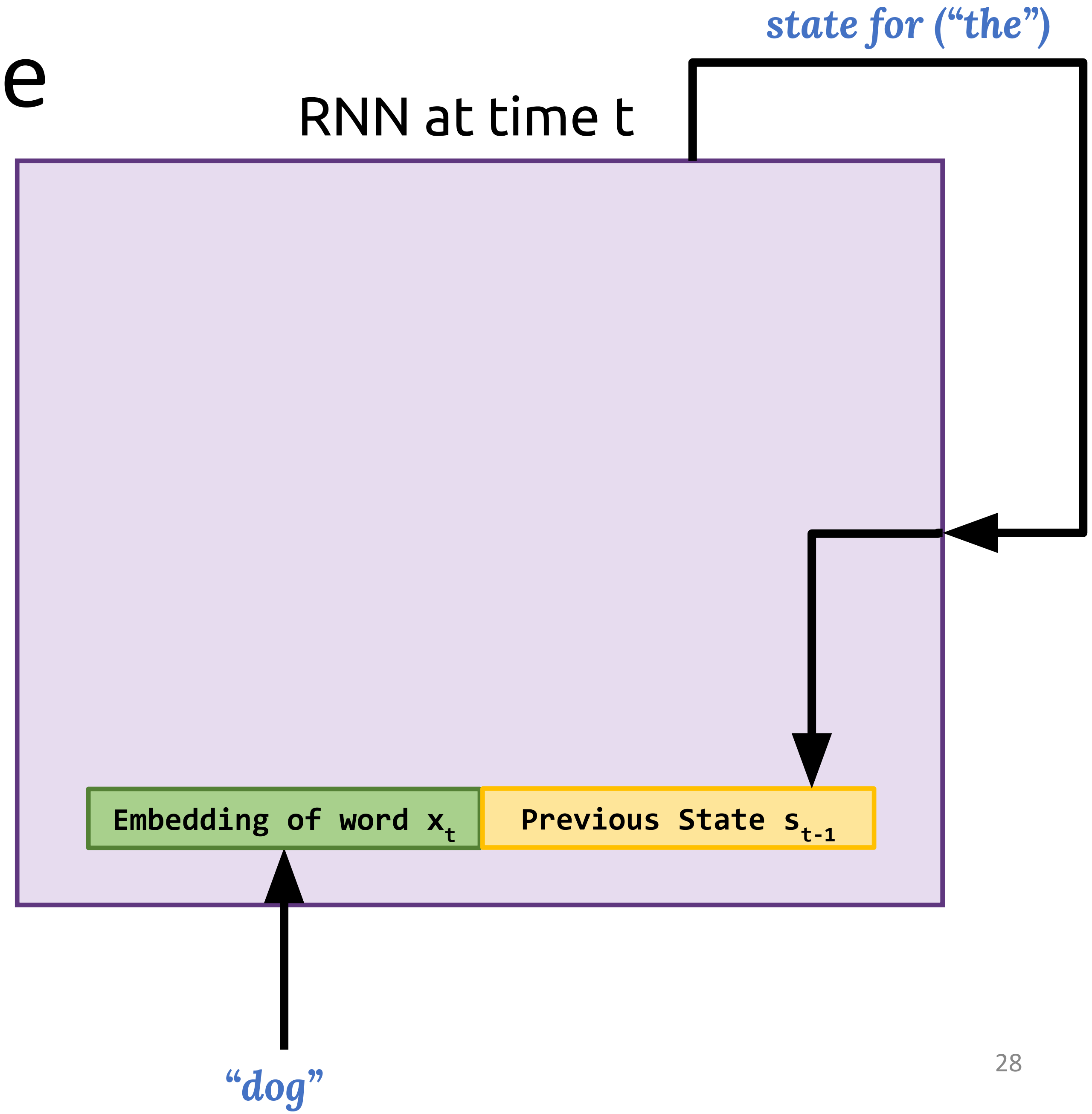
At each step of our RNN, we will get an input word, and a state vector from the previous cell.



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.



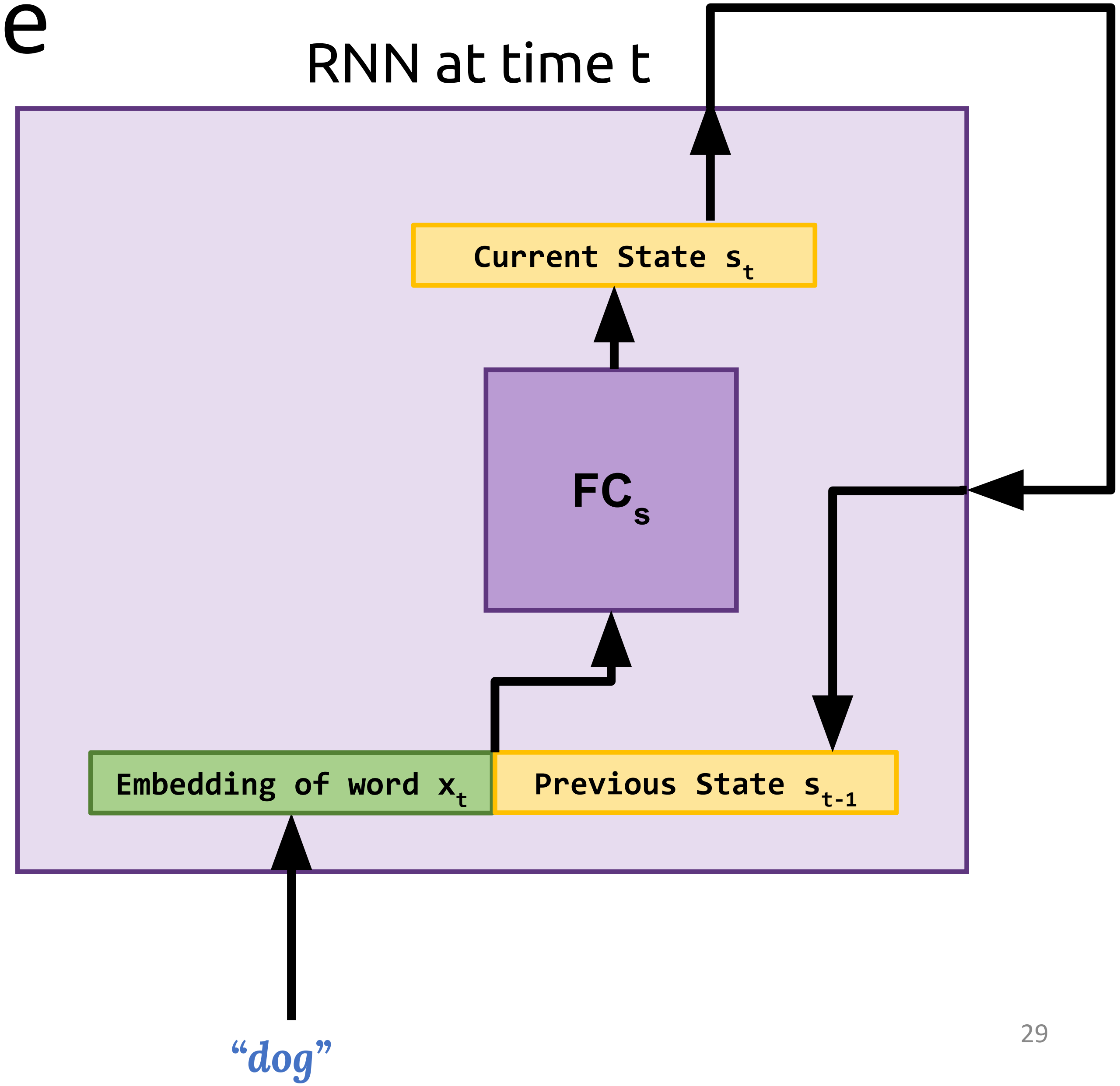
RNN Cell Architecture

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 at time t



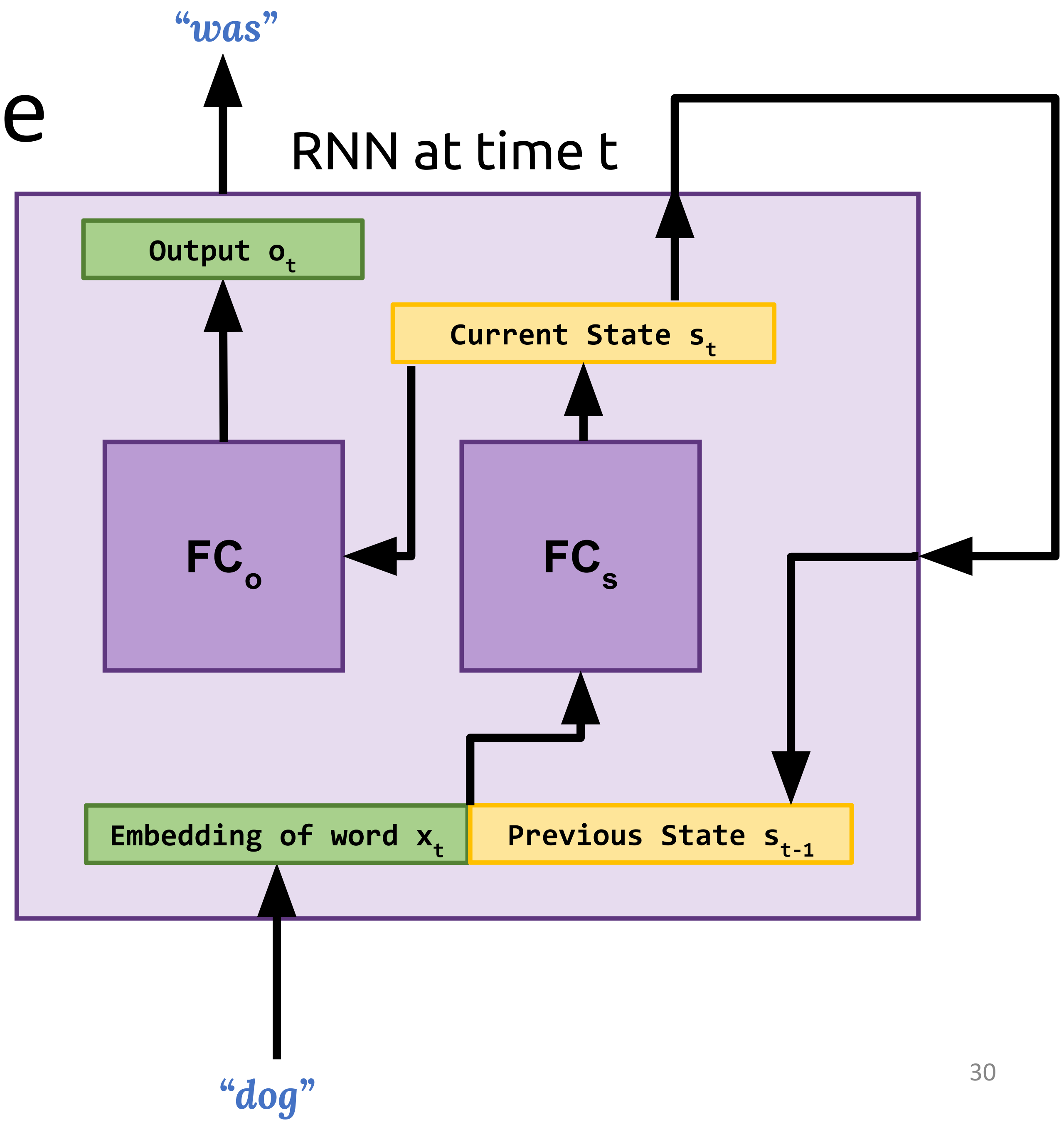
RNN Cell Architecture

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.

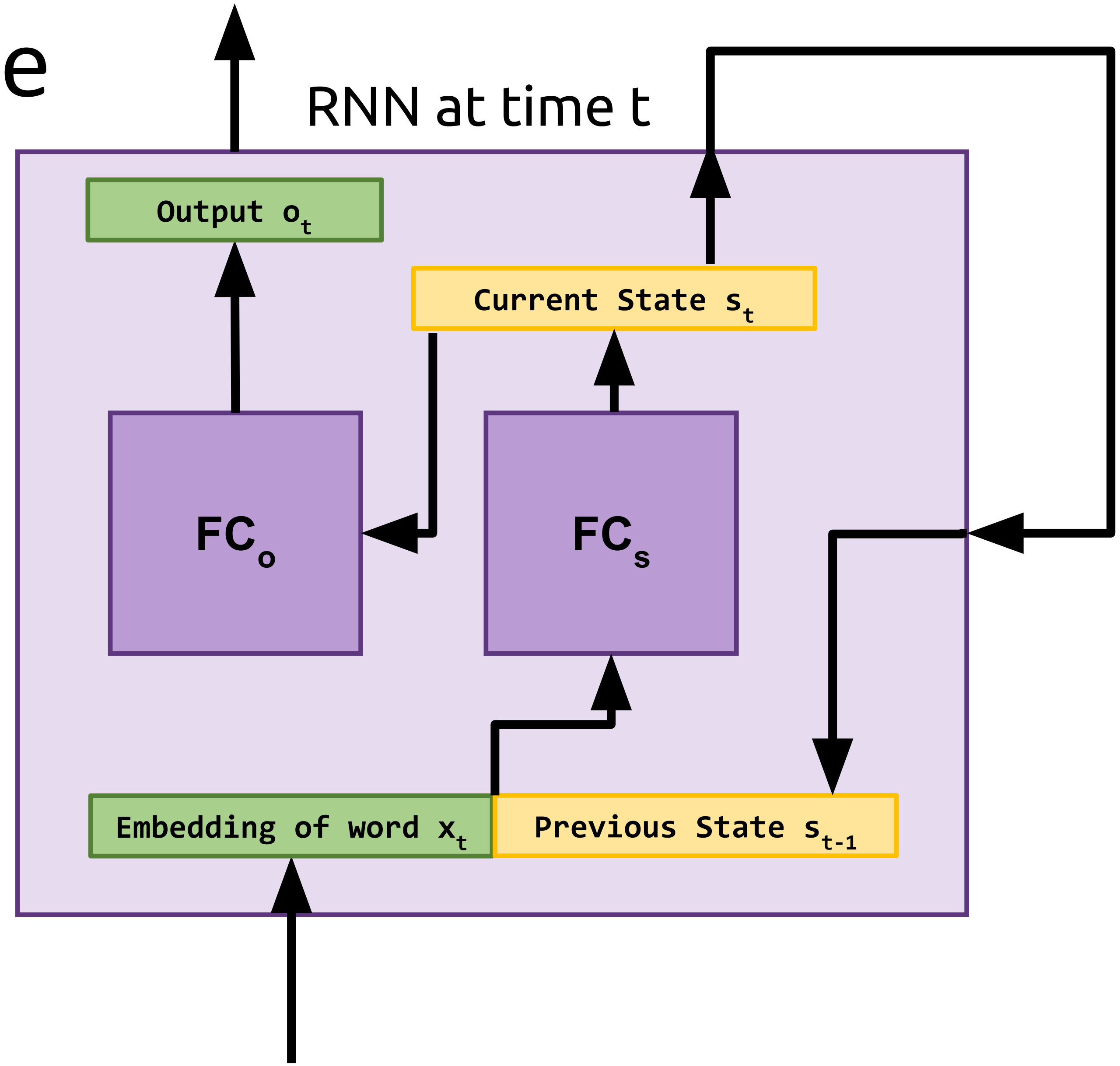


RNN Cell Architecture

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

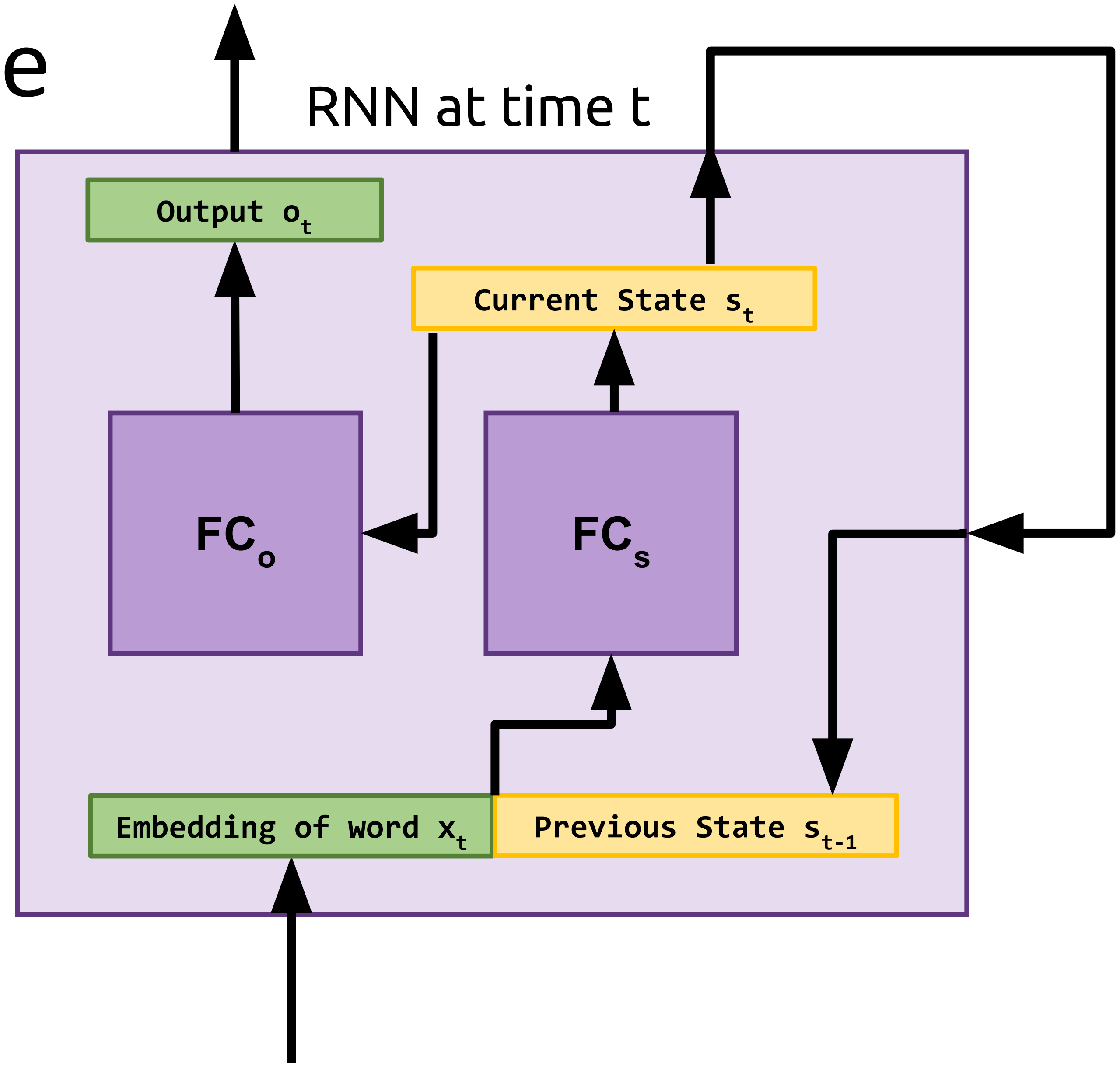
We can represent the RNN in with the following equations:

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

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

Nonlinear activations (e.g. sigmoid, tanh)

Any questions?



RNN Cell Architecture

We can represent the RNN in with the following equations:

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

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

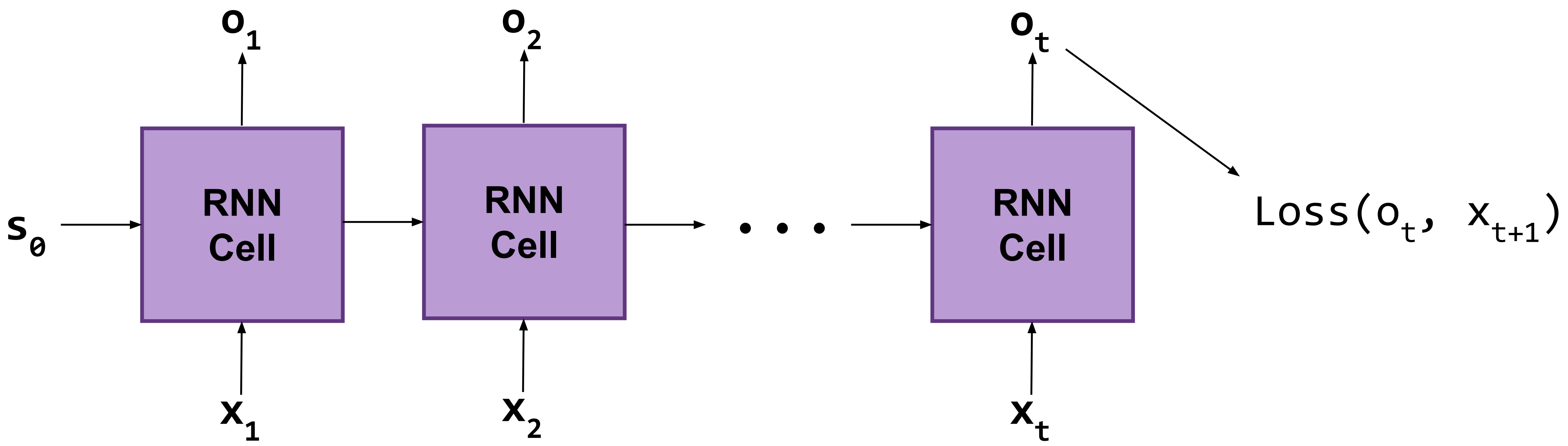
This brings up an immediate question: **what is $\boldsymbol{s_0}$?**

Typically, we initialize s_0 to be a vector of zeros

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

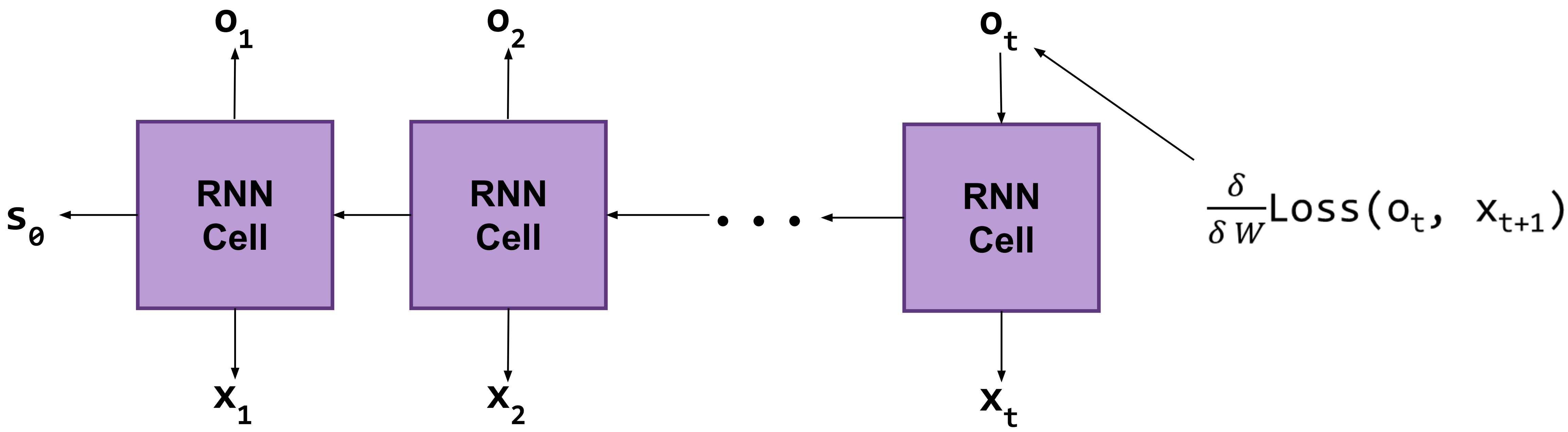
Training RNNs

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

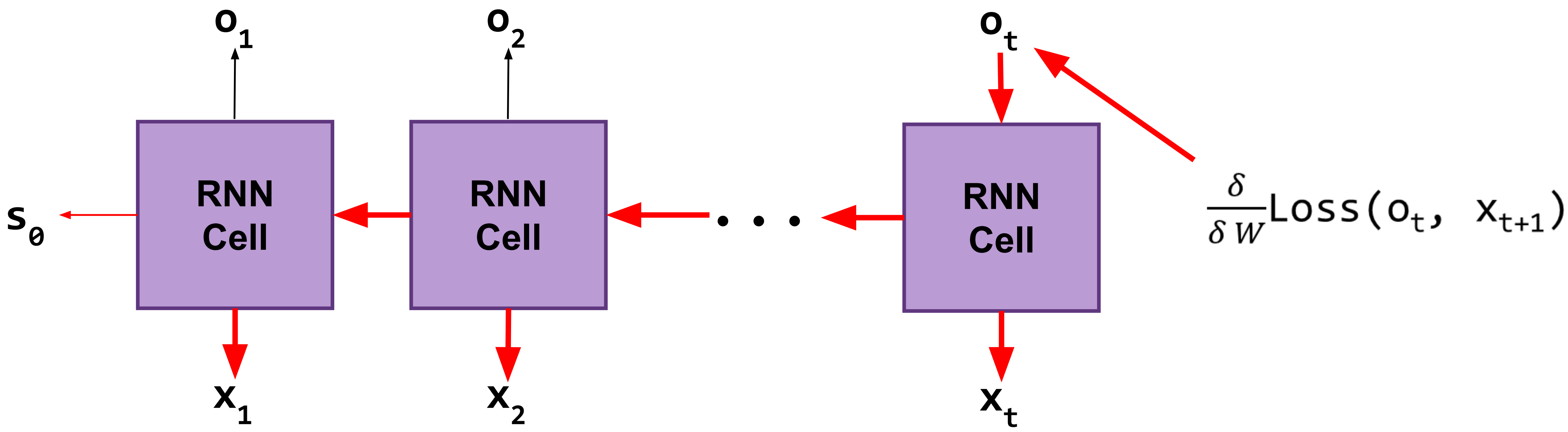
But what happens when we differentiate the loss and backpropagate?



Training RNNs

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

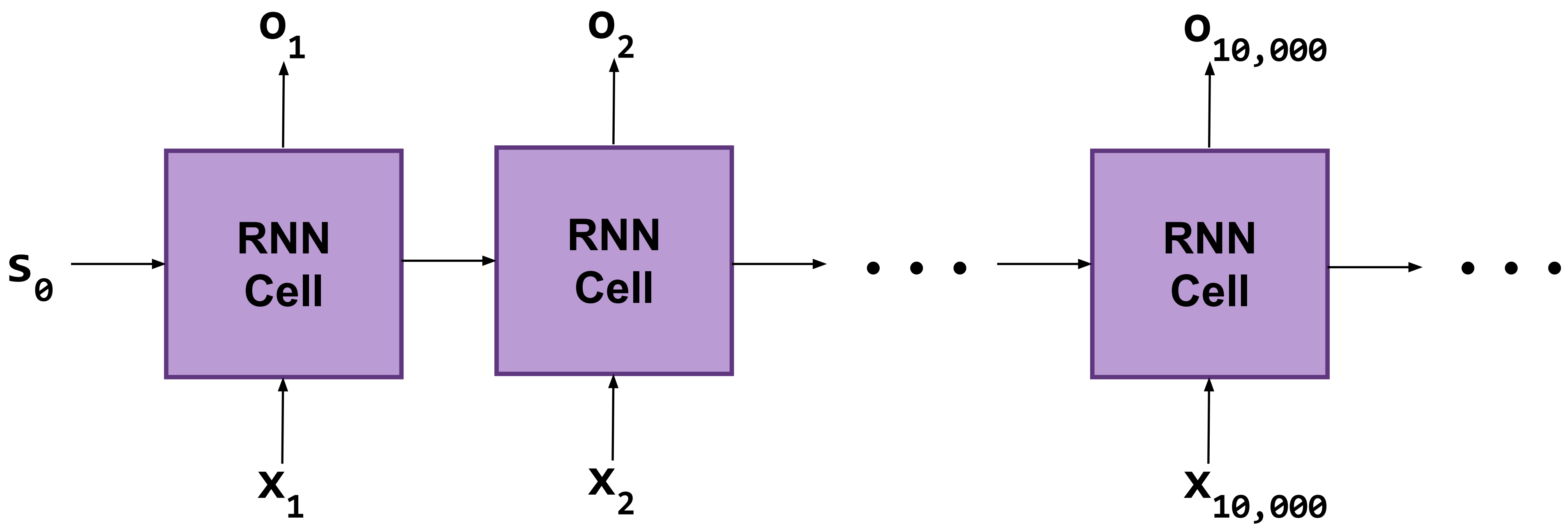
We call this *backpropagation through time.*



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`

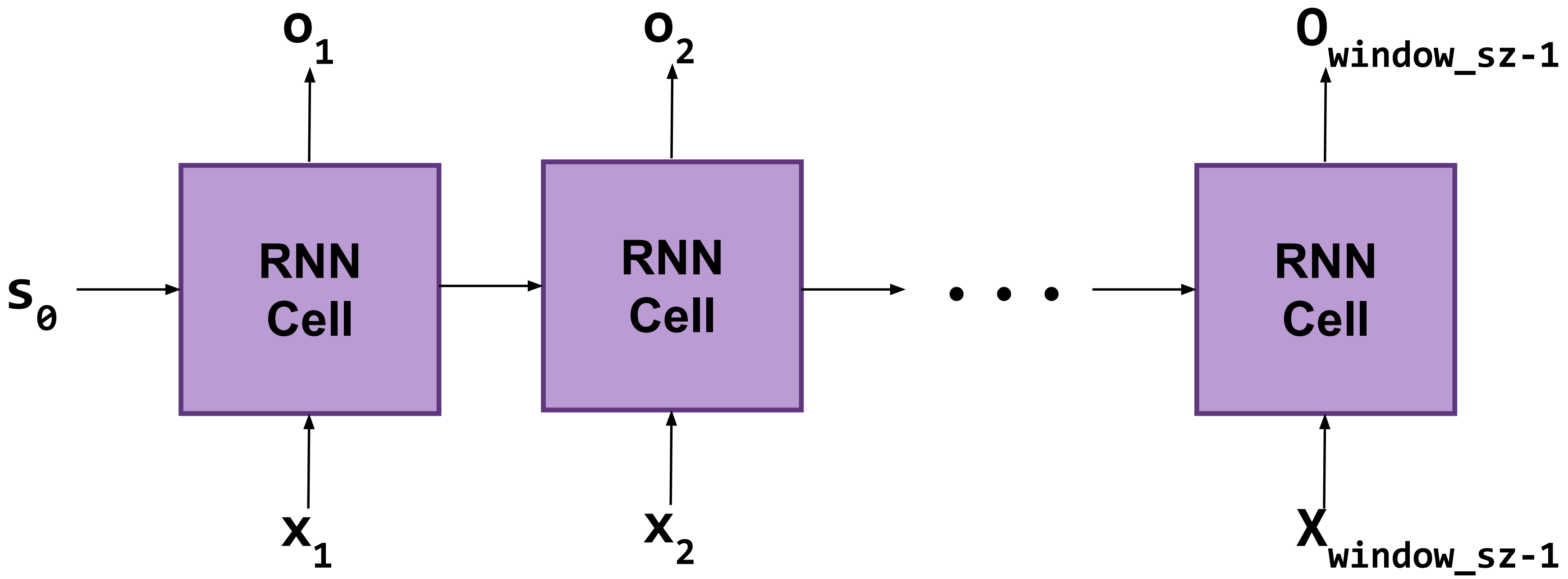
The new shape of our data should be:

```
(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

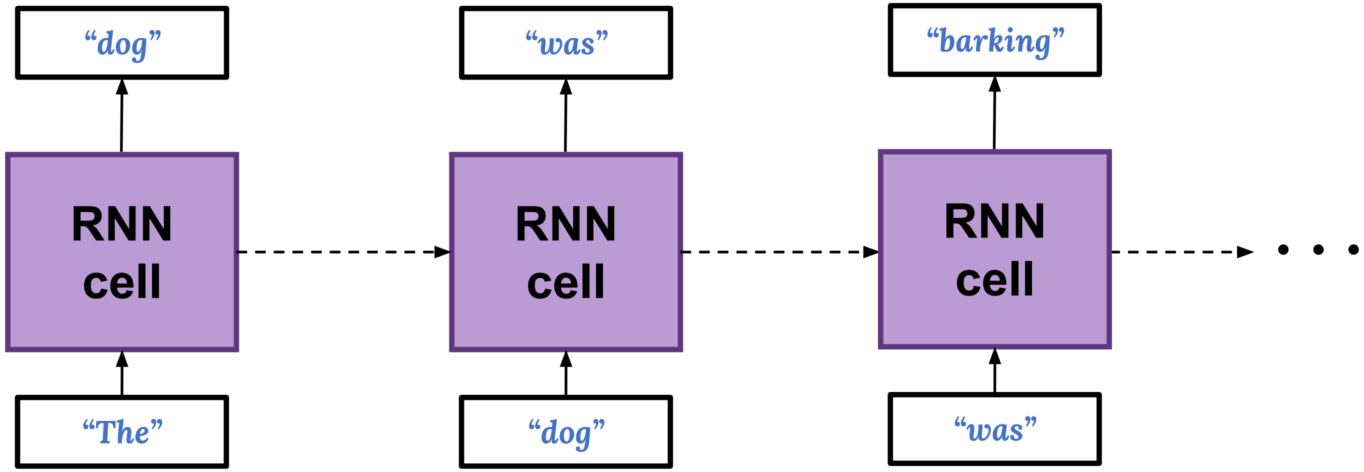


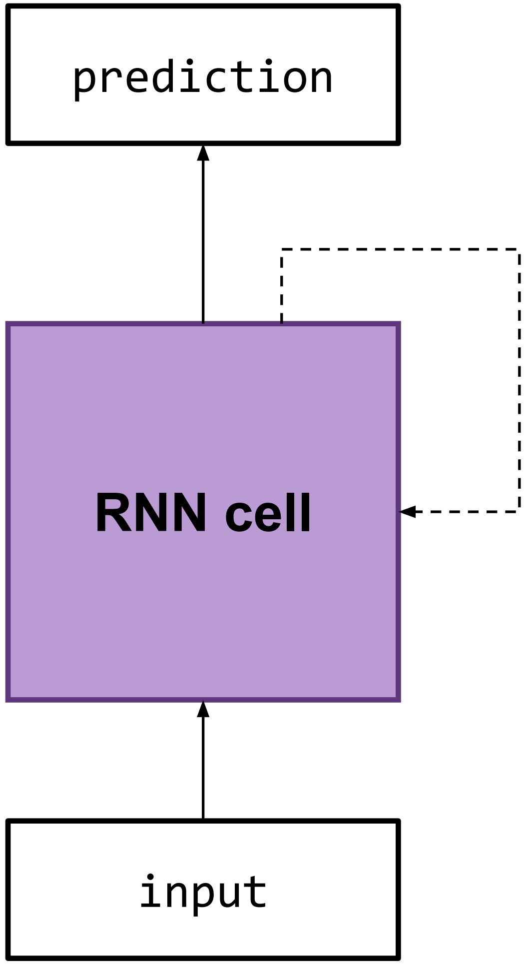
Any questions?

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

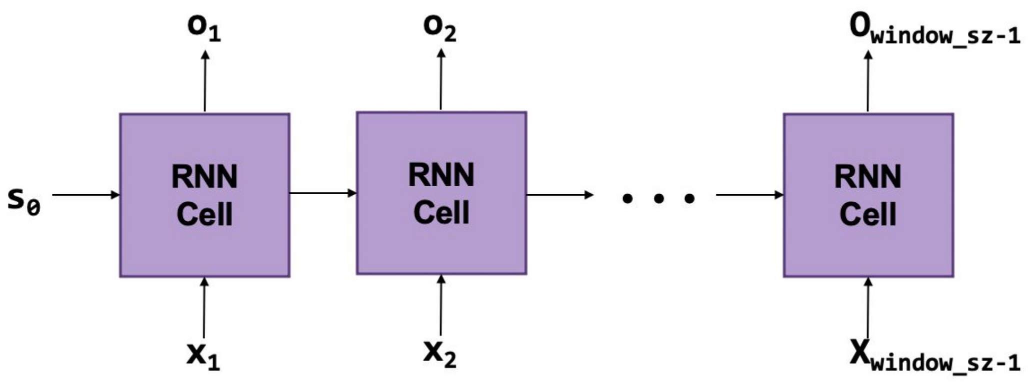
RNNs in Tensorflow

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
```

Any questions?

RNNs in Tensorflow

```
inputs = np.random.random([32, 10, 8]).astype(np.float32)
simple_rnn = tf.keras.layers.SimpleRNN(4)
```

```
output = simple_rnn(inputs)
```

inputs: a [batch_sz, seq_length, embedding_sz] tensor

Go to **www.menti.com** and use the code **8305 9036**

What is the size of
(a) output
(b) whole_sequence_output?

```
simple_rnn = tf.keras.layers.SimpleRNN(4,
return_sequences=True)
```

```
whole_sequence_output = simple_rnn(inputs)
```

RNNs are a marked improvement over previous language models we’ve seen

But what are the implications when language models get really good?

Like really, really, *really* good

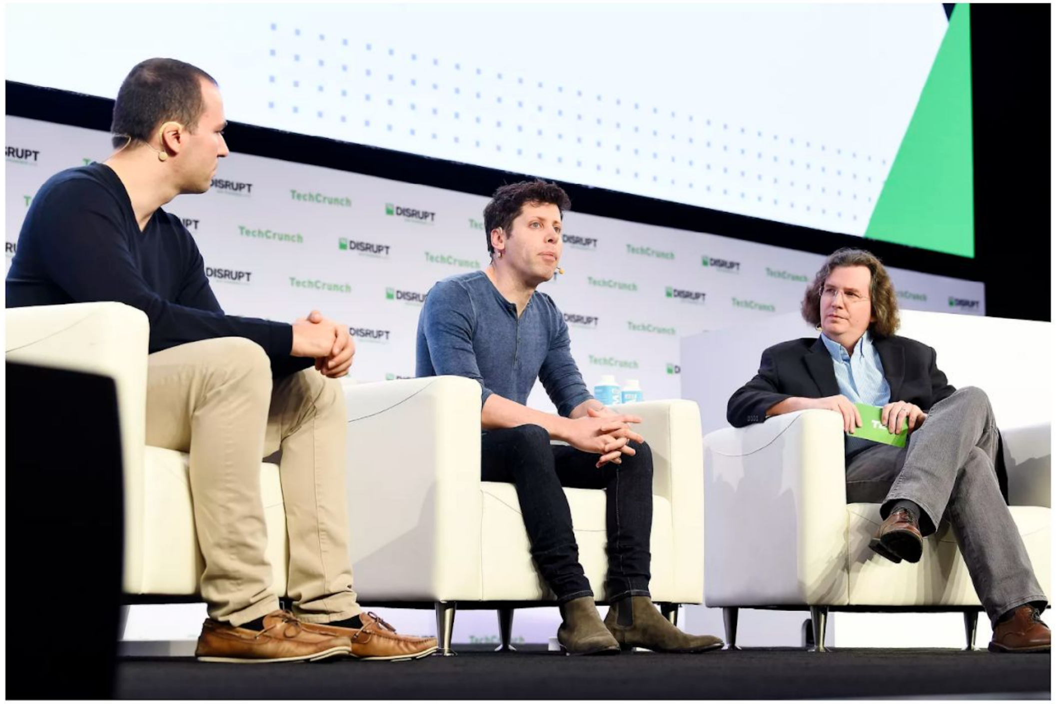
GPT-3

GPT-3, explained: This new language AI is uncanny, funny — and a big deal

Computers are getting closer to passing the Turing Test.

By Kelsey Piper | Aug 13, 2020, 9:50am EDT

SHARE



OpenAI co-founder and chair Greg Brockman, OpenAI co-founder and CEO Sam Altman, and TechCrunch news editor Frederic Lardinois during TechCrunch Disrupt San Francisco 2019. | Steve Jennings/Getty Images for TechCrunch

GPT-3

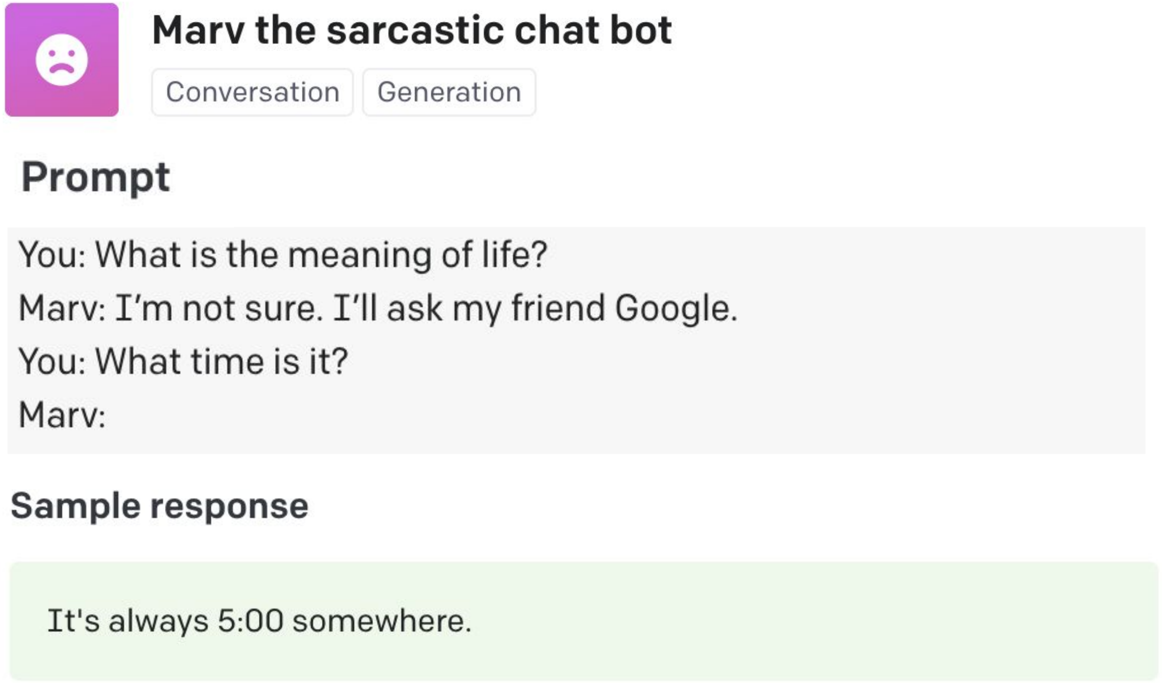
Background:

- Attention- and transformer-based model (We'll be discussing the details of these models next week in class, so stay tuned!)
- First Described by OpenAI in May 2020

GPT-3

For now, what you need to know is:

- GPT-3 represents the cutting edge of text generation
- **Given a prompt phrase , it will produce *scarily good* continuations of the writing**



GPT-3

For now, what you need to know is:

- GPT-3 represents the cutting edge of text generation
- **Given a prompt phrase , it will produce *scarily good* continuations of the writing**

Prompt

Write a restaurant review based on these notes:

Name: The Blue Wharf
Lobster great, noisy, service polite, prices good.

Review:

Sample response

The Blue Wharf is a great place to go for a lobster dinner. The service is polite and the prices are good. The only downside is that it is a bit noisy.

GPT-3 can also produce text that mimics the writing/speech styles of some famous people.
Examples here

OpenAI thinks GPT-3 could be dangerous. Why?

- Impacts of fake news and bot networks in the 2016 US elections
- What if fake news didn’t even have to be written by humans?

OpenAI thinks GPT-3 could be dangerous. Why?

- Impacts of fake news and bot networks in the 2016 US elections
- What if fake news didn’t even have to be written by humans?
- **Could this significantly undermine trust in written sources?**

OpenAI thinks GPT-3 could be dangerous. Why?

- Impacts of fake news and bot networks in the 2016 US elections
- What if fake news didn't even have to be written by humans?
- **Could this significantly undermine trust in written sources?**

OpenAI will not make GPT3 open source— instead, they have released an API and limits the number of people allowed to try the actual model.

There's already been some concerning GPT-3 usage:

THE VERGE

A college student used GPT-3 to write fake blog posts and ended up at the top of Hacker News

Kim Lyons · 8/16/2020

A robot wrote this entire article. Are you scared yet, human?

GPT-3

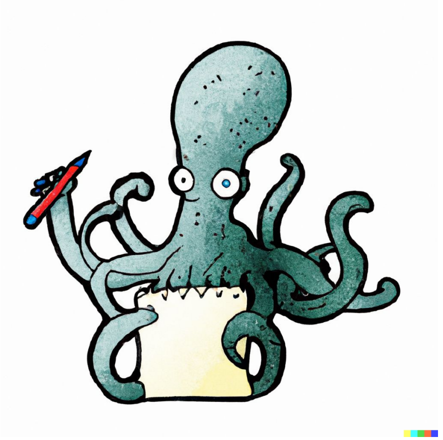
We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To convince us robots come in peace

by theguardian.com

Recap

Limitations of N-gram models

- Size of weights dependent on N
- Limited Flexibility
- Recurrent connection can help



RNNs

- RNN cell architecture
- Backprop through time
- RNNs in Tensorflow