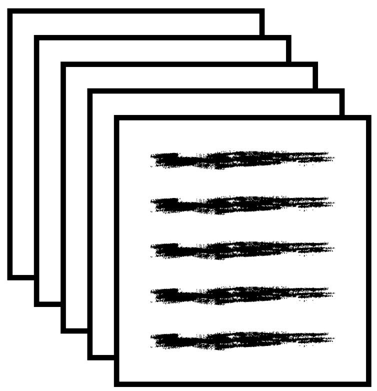
Carnegie Mellon University
Heinz College

95-865 Unstructured Data Analytics

Lecture 14: Wrap up RNNs; a glimpse of word embeddings; start coverage on text generation

Slides by George H. Chen

(Flashback) Sentiment Analysis with IMDb Reviews



Training reviews

Step 1: Tokenize & build vocabulary

Word index	Word	2D Embedding
0	this	[-0.57, 0.44]
1	movie	[0.38, 0.15]
2	rocks	[-0.85, 0.70]
3	sucks	[-0.26, 0.66]

Step 2: Encode each review as a sequence of word indices into the vocab

Ordering of words matters

Different reviews can have different lengths

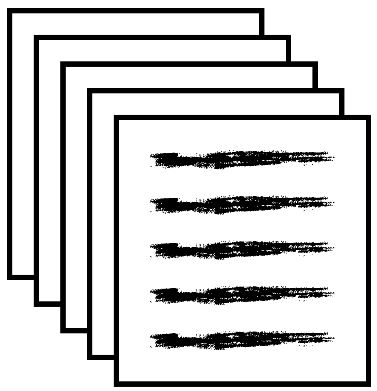
"this movie rocks" → 0 1 2

"this movie sucks" → 0 1 3

"this sucks" → 0 3

Step 3: Use word embeddings to represent each word

(Flashback) Sentiment Analysis with IMDb Reviews



Training reviews

Step 1: Tokenize & build vocabulary

Word index	Word	2D Embedding
0	this	[-0.57, 0.44]
1	movie	[0.38, 0.15]
2	rocks	[-0.85, 0.70]
3	sucks	[-0.26, 0.66]

Step 2: Encode each review as a sequence of word indices into the vocab

"this movie sucks" → 0 1 3

Step 3: Use word embeddings to represent each word

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]

(Flashback) Do Data Actually Live on Manifolds?

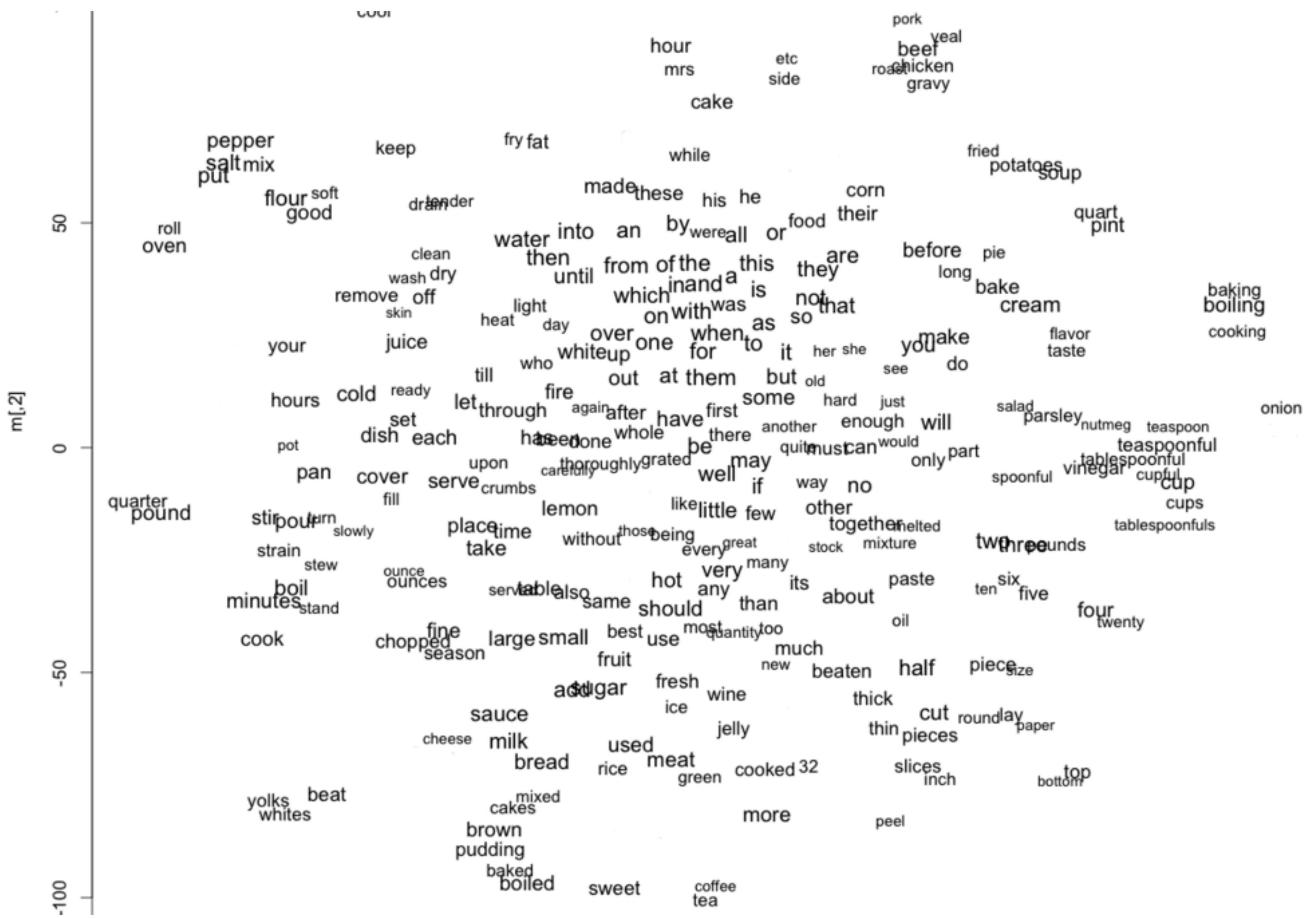
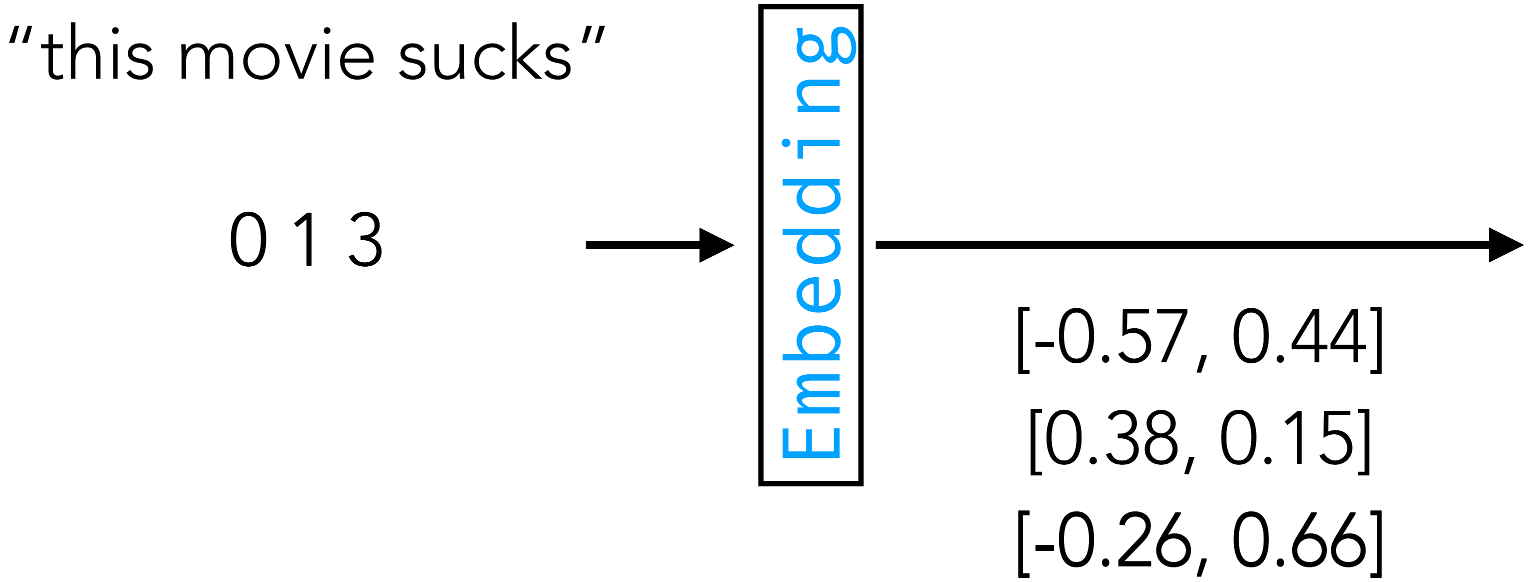
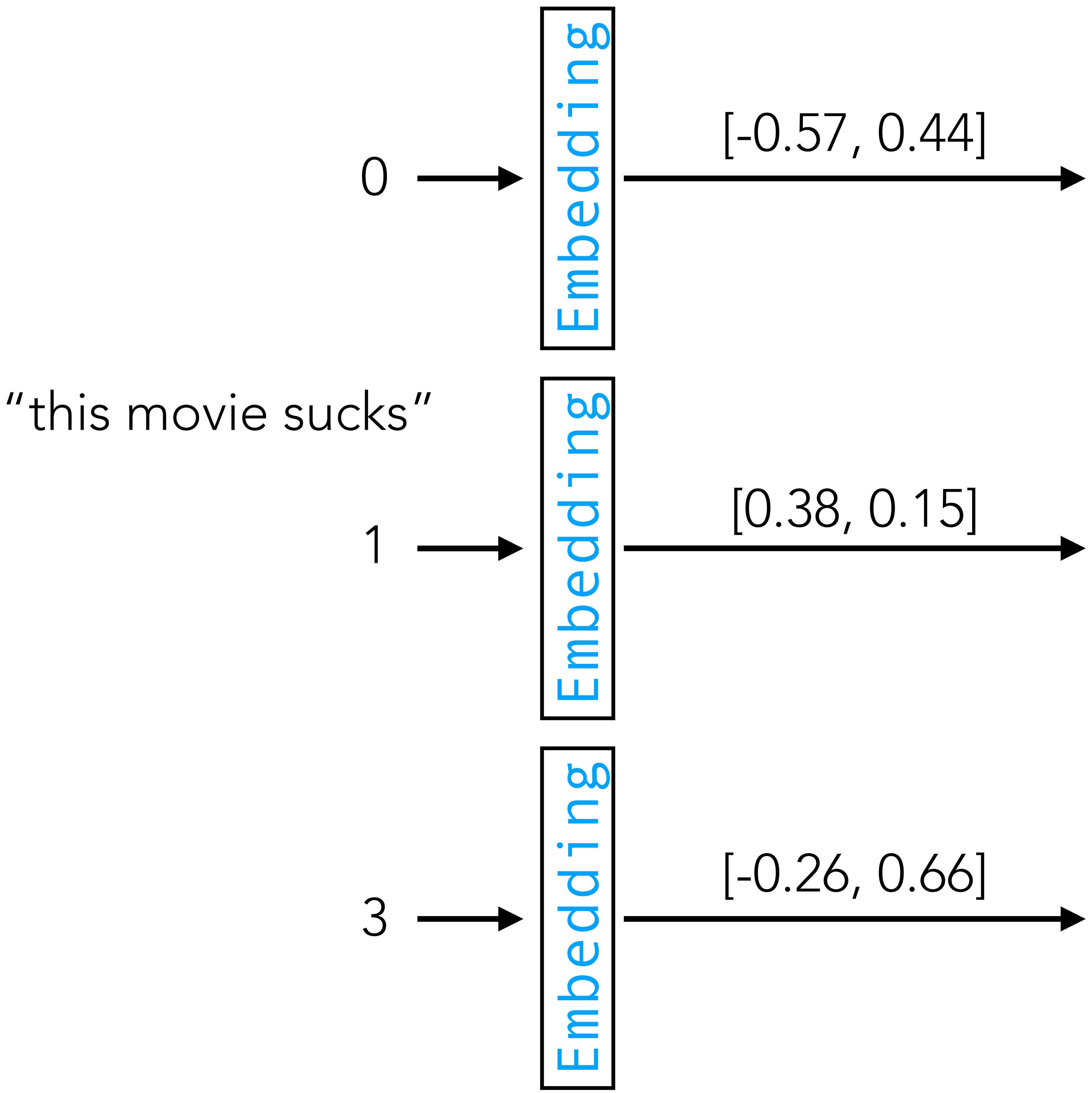


Image source: http://www.adityathakker.com/wp-content/uploads/2017/06/word-embeddings-994x675.png

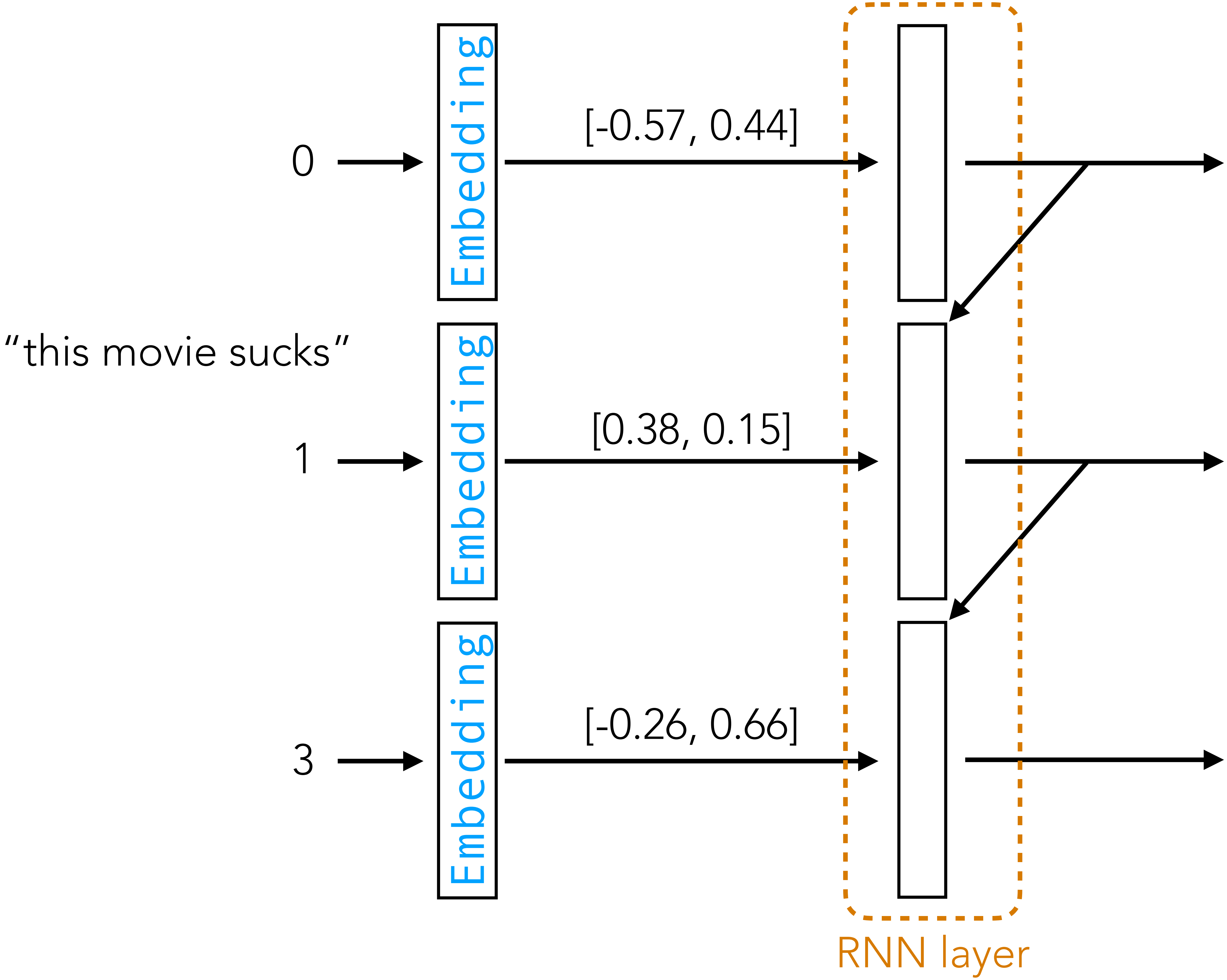
Sentiment Analysis with IMDb Reviews



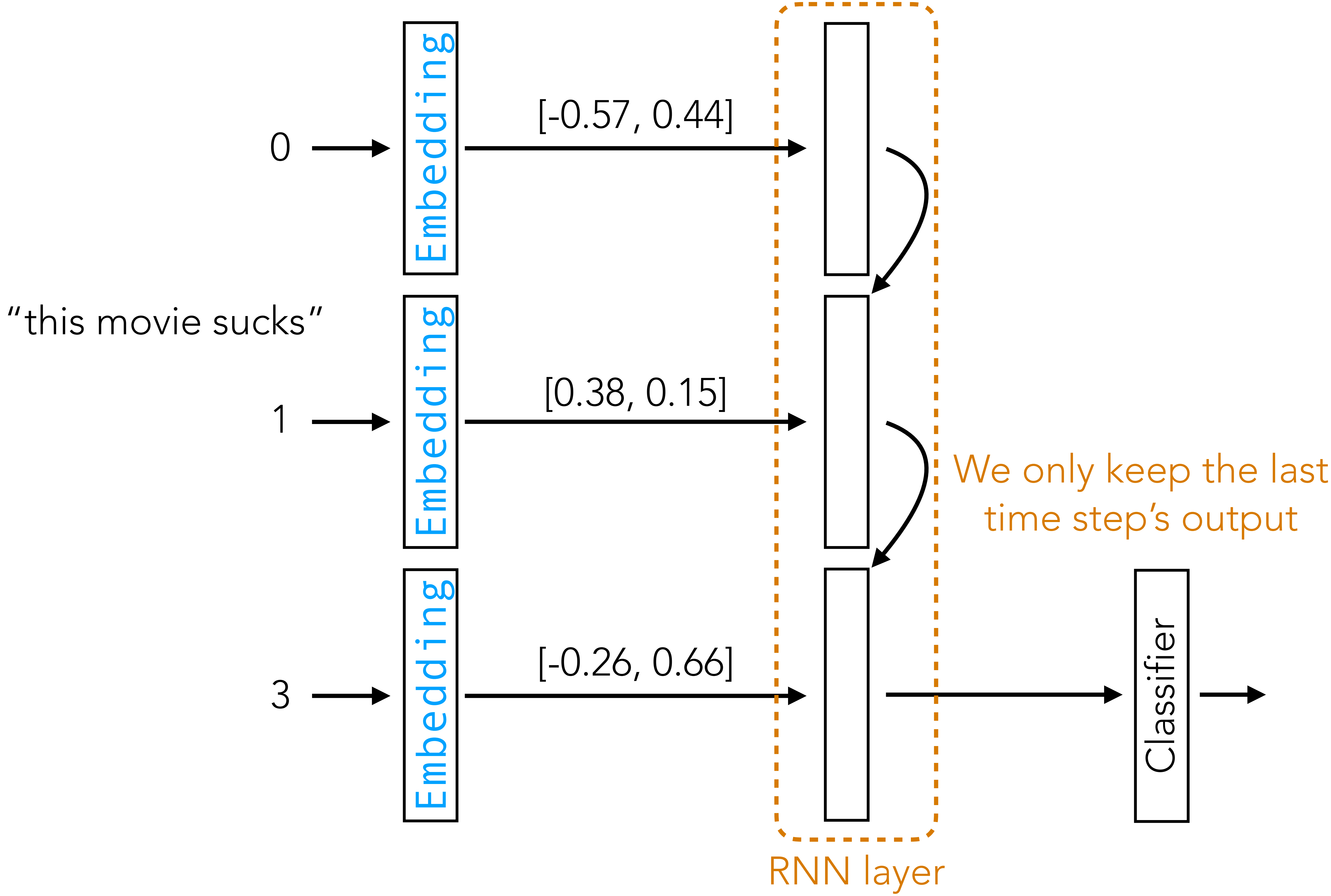
Sentiment Analysis with IMDb Reviews



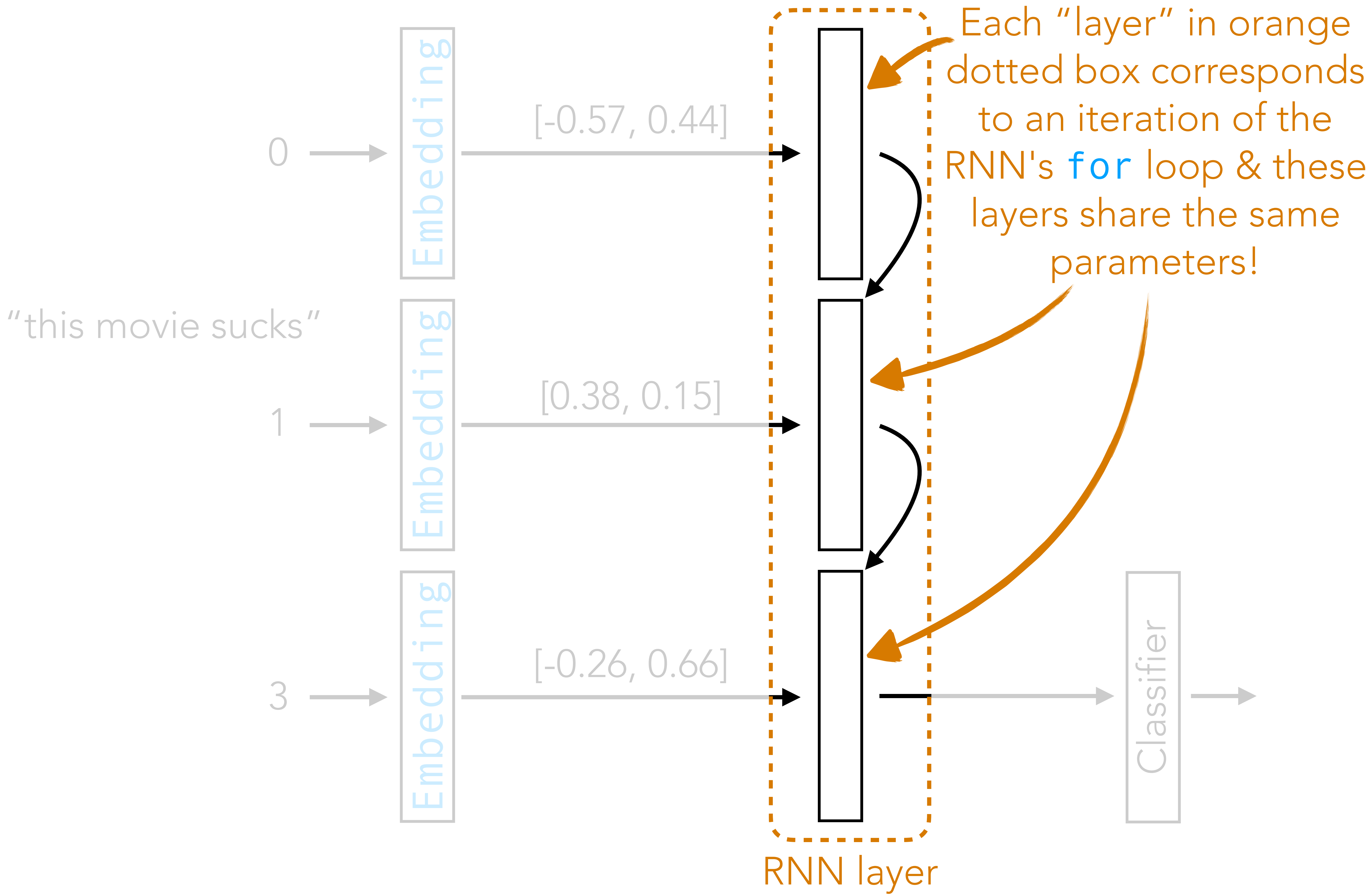
Sentiment Analysis with IMDb Reviews



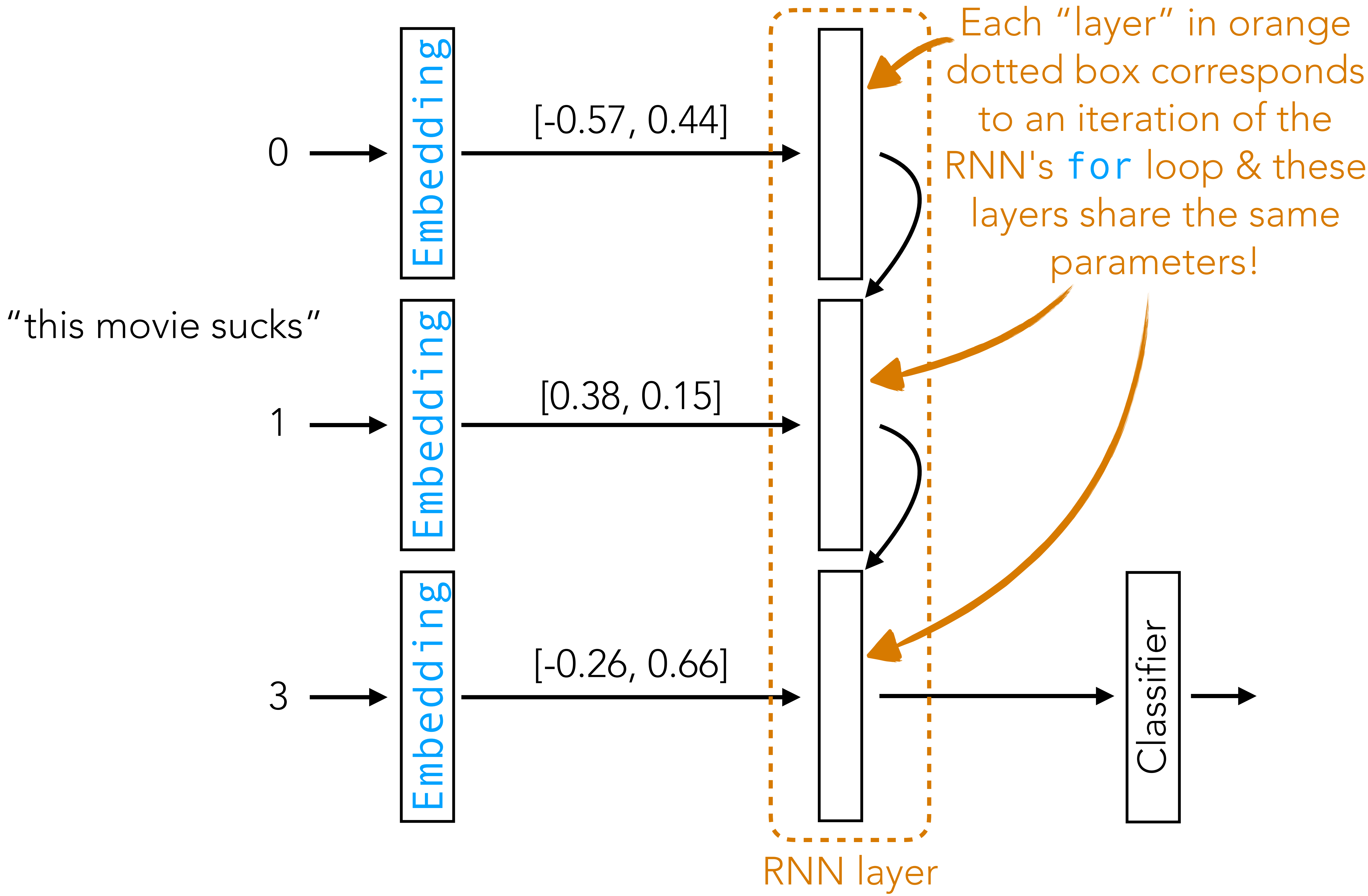
Sentiment Analysis with IMDb Reviews



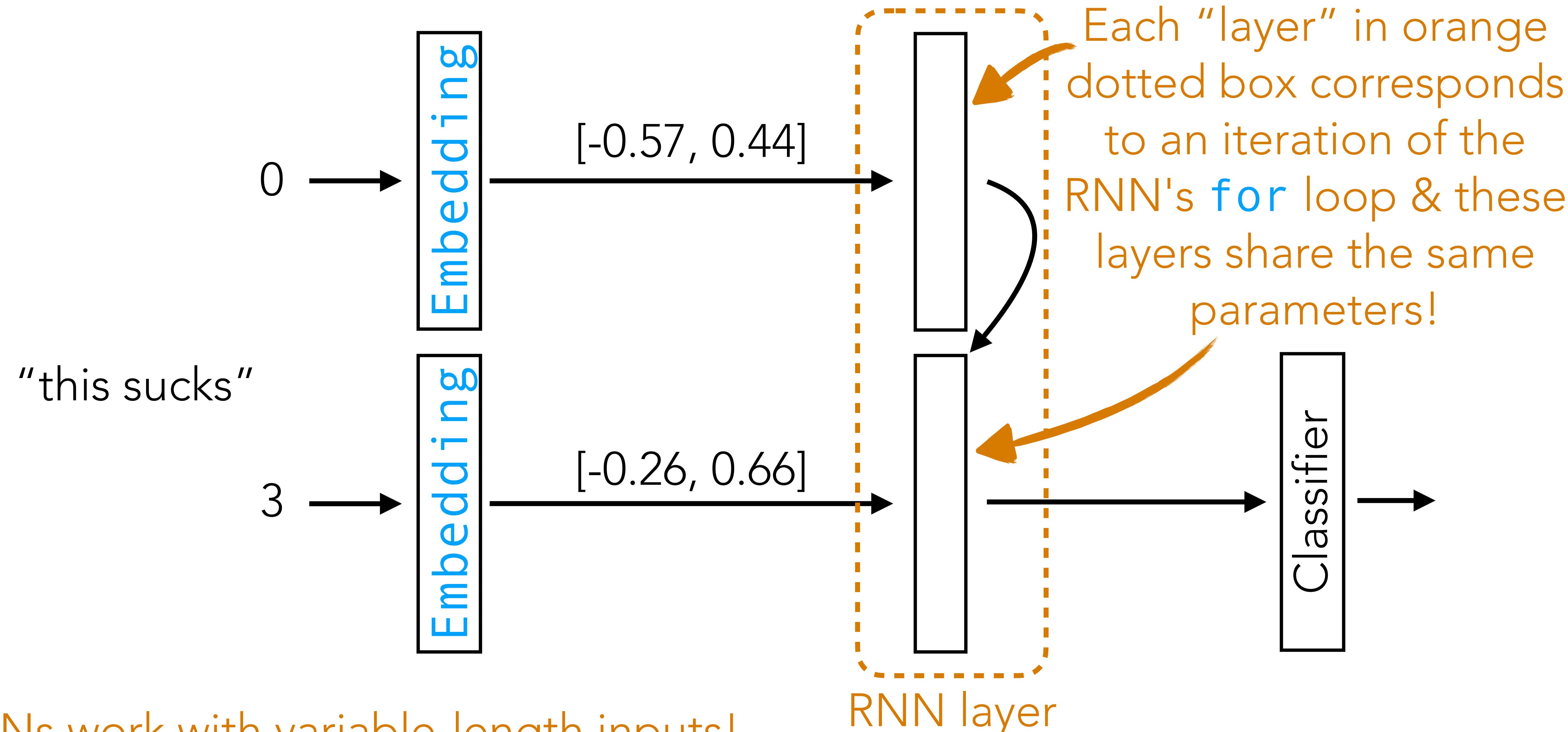
Sentiment Analysis with IMDb Reviews



Sentiment Analysis with IMDb Reviews



Sentiment Analysis with IMDb Reviews



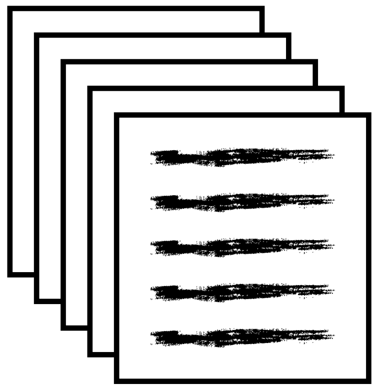
RNNs work with variable-length inputs!

Note that the “RNN layer” here could refer to a vanilla ReLU RNN or a more complicated RNN such as an “LSTM”, “GRU”, etc

Note: Sometimes in text analysis, the word embeddings are treated as fixed, so we do *not* update them during training

What if we didn’t use word embeddings?

Sentiment Analysis with IMDb Reviews



Training reviews

Step 1: Tokenize & build vocabulary

Word index	Word	2D Embedding
0	this	[-0.57, 0.44]
1	movie	[0.38, 0.15]
2	rocks	[-0.85, 0.70]
3	sucks	[-0.26, 0.66]

Step 2: Encode each review as a sequence of word indices into the vocab

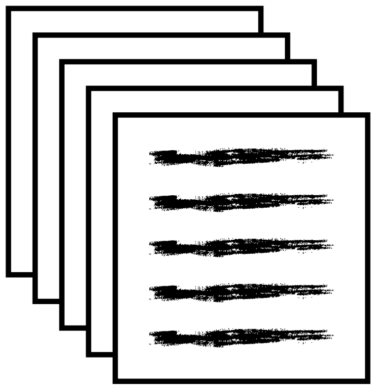
"this movie sucks" → 0 1 3

Step 3: Use word embeddings to represent each word

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]

Bad Strategy: One-Hot Encoding

Step 1: Tokenize & build vocabulary



Training reviews

Word index	Word	One-hot encoding
0	this	[1, 0, 0, 0]
1	movie	[0, 1, 0, 0]
2	rocks	[0, 0, 1, 0]
3	sucks	[0, 0, 0, 1]

Step 2: Encode each review as a sequence of word indices into the vocab

"this movie sucks" → 0 1 3

Step 3: Use one-hot encoding to represent each word

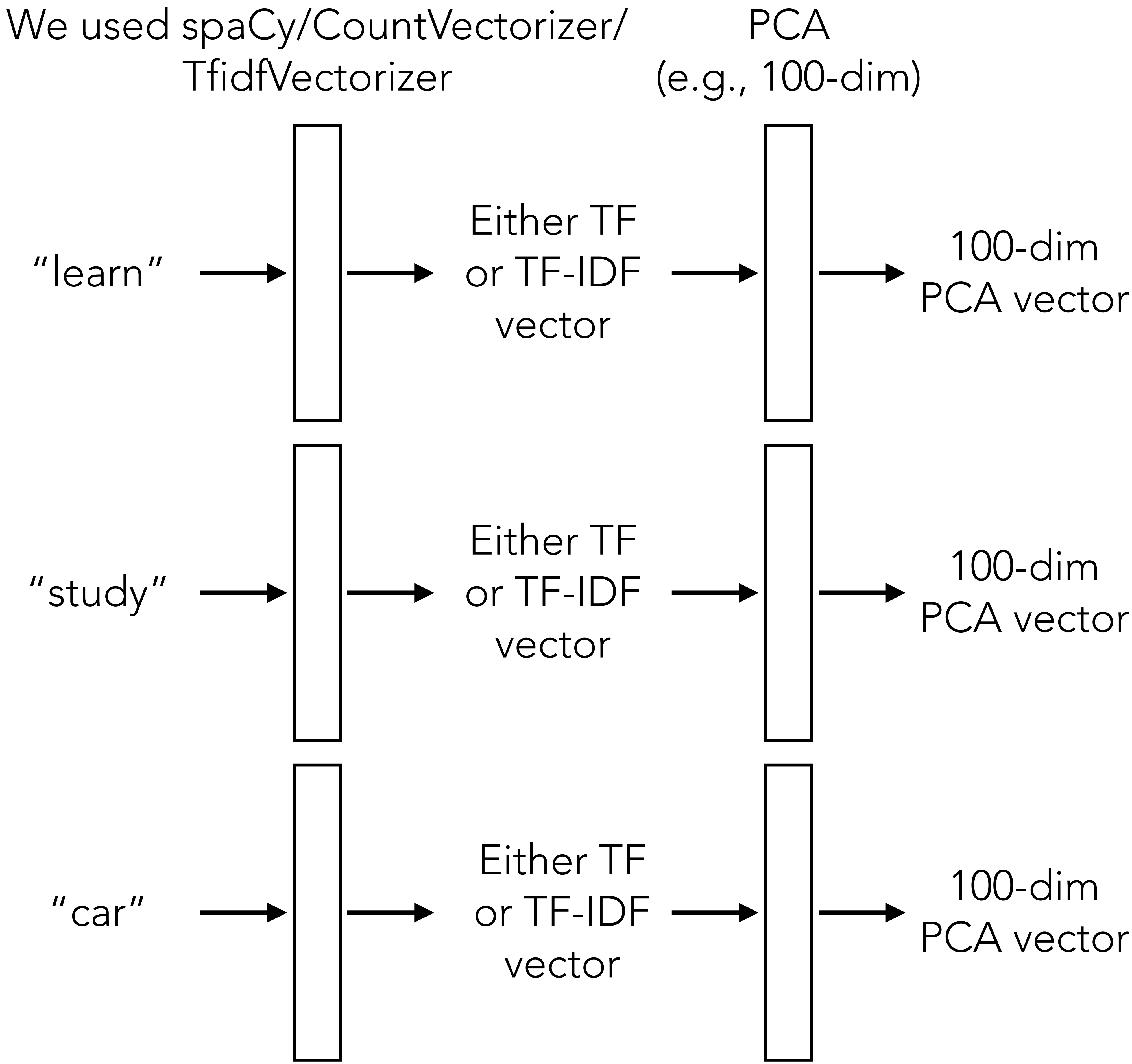
This strategy tends to work poorly in practice: distance between every pair of words is the same in one-hot encoding!

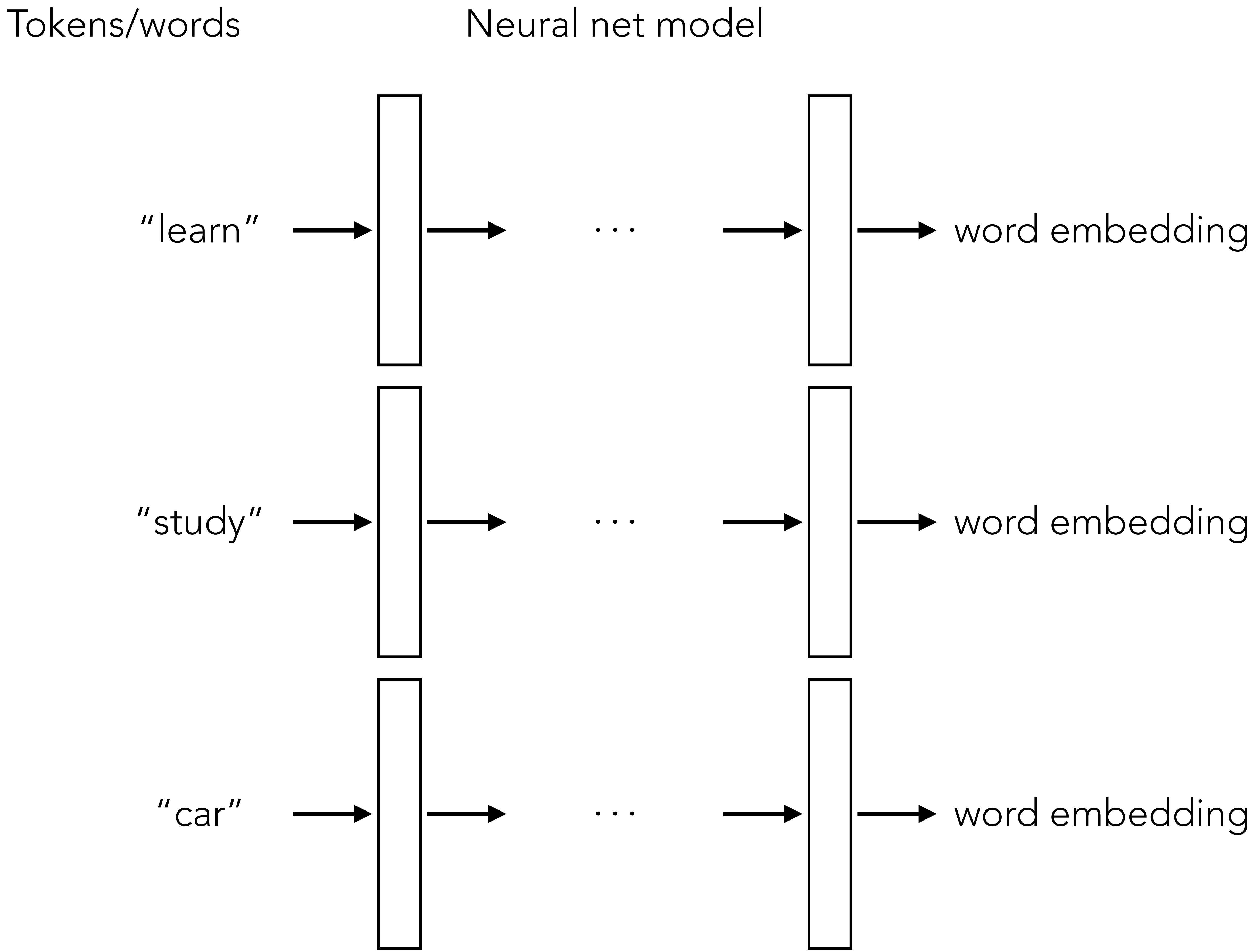
[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 0, 1]

Recap/Important Reminder

- Neural nets are *not* doing magic; incorporating structure is very important to state-of-the-art deep learning systems
  - Word embeddings encode semantic structure—words with similar meaning are mapped to nearby Euclidean points
  - CNNs encode semantic structure for images—images that are "similar" are mapped to nearby Euclidean points
- An RNN tracks how what's stored in memory changes over time — **an RNN's job is made easier if the memory is a semantically meaningful representation**

A brief glimpse at word embeddings


We used spaCy/CountVectorizer/
TfidfVectorizer
PCA
(e.g., 100-dim)
“learn”
Either TF
or TF-IDF
vector
100-dim
PCA vector
“study”
Either TF
or TF-IDF
vector
100-dim
PCA vector
“car”
Either TF
or TF-IDF
vector
100-dim
PCA vector


Tokens/words
Neural net model
“learn”
. . .
word embedding
“study”
. . .
word embedding
“car”
. . .
word embedding

Word Embeddings: Even without labels, we can set up a prediction problem!

Hide part of training data and try to predict what you've hid!

This is commonly referred to as *self-supervised learning*

We're setting up a prediction task in an *unsupervised* setting!

Word Embeddings: word2vec (2013)

Can solve tasks like the following:

Man is to King as Woman is to ???

Word Embeddings: word2vec (2013)

Can solve tasks like the following:

Man is to King as Woman is to Queen

Word Embeddings: word2vec (2013)

Can solve tasks like the following:

Man is to King as Woman is to Queen

Which word doesn't belong?

blue, red, green, crimson, transparent

Word Embeddings: word2vec (2013)

Can solve tasks like the following:

Man is to King as Woman is to <u>Queen</u>

Which word doesn't belong?

blue, red, green, crimson, <u>transparent</u>

Word Embeddings: word2vec (2013)

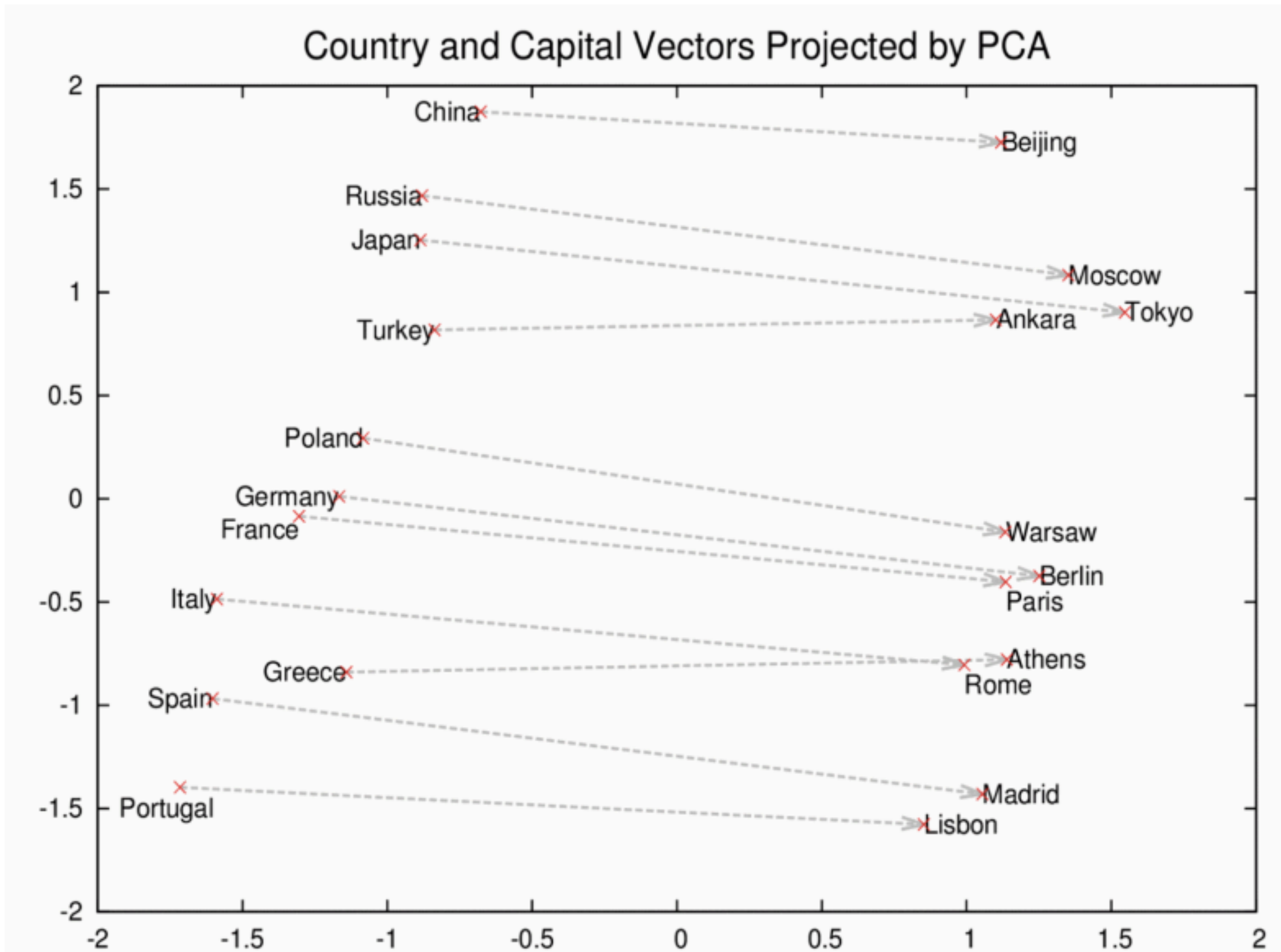


Image source: https://deeplearning4j.org/img/countries_capitals.png

Word Embeddings: word2vec (2013)

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Predict context of each word!

Training data point: epidemic

“Training labels”: the, opioid, or, opioid

Word Embeddings: word2vec (2013)

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Predict context of each word!

Training data point: or

"Training labels": opioid, epidemic, opioid, crisis

Word Embeddings: word2vec (2013)

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Predict context of each word!

These are "positive" (correct) examples of what context words are for "opioid"

Training data point: opioid

"Training labels": epidemic, or, crisis, is

Also provide "negative" examples of words that are *not* likely to be context words (by randomly sampling words elsewhere in document)

Word Embeddings: word2vec (2013)

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

randomly sampled word

Predict context of each word!

Training data point: opioid

"Negative training label": 2010s

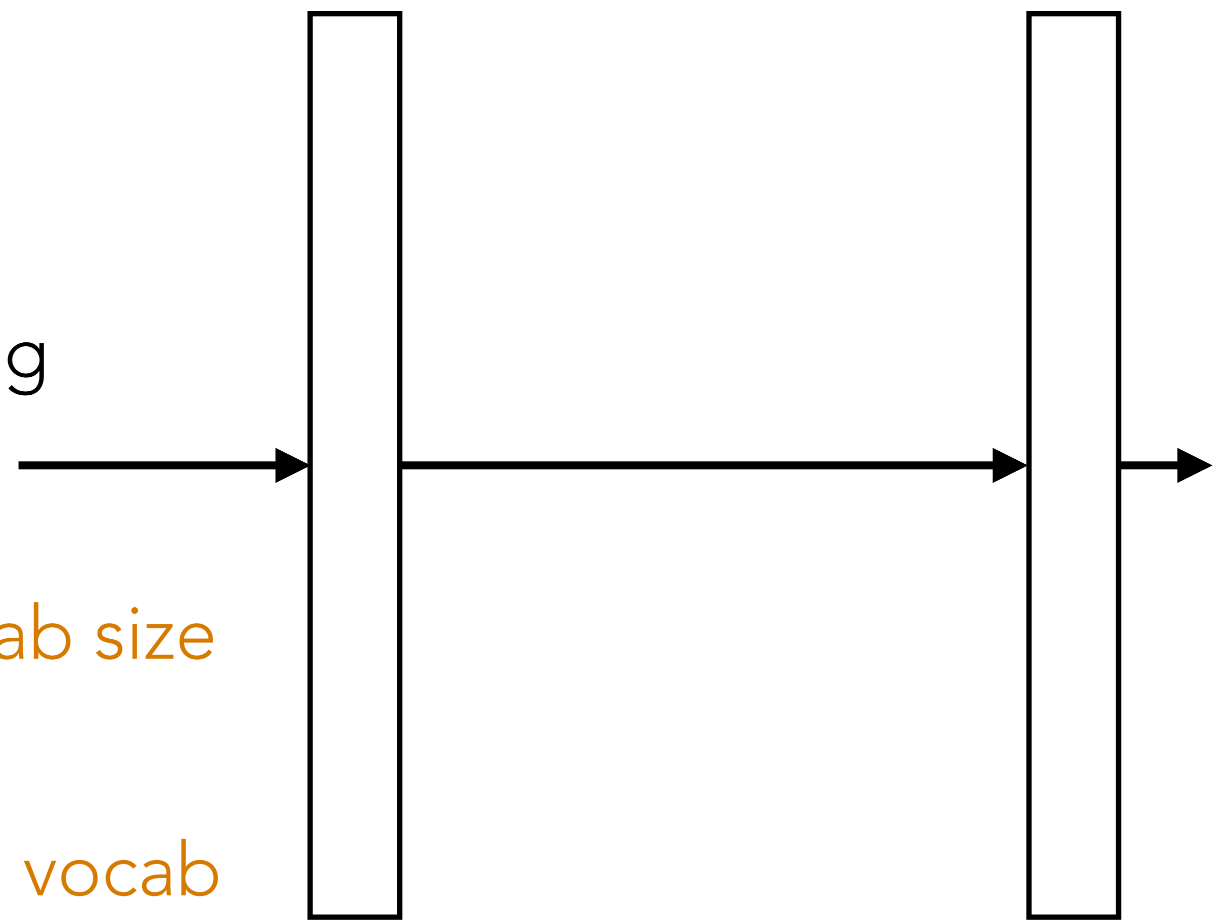
Also provide "negative" examples of words that are *not* likely to be context words (by randomly sampling words elsewhere in document)

Word2vec Neural Net

"opioid"
Use one-hot encoding
[0, 0, …, 1, …, 0]

vector length = vocab size

index of "opioid" in vocab



Want real context words (e.g., "epidemic", "crisis") to have high probability

Linear, no bias vector
(100 nodes)

Learned weight matrix used as word embedding!

Linear
(# nodes = vocab size),
Softmax

(Treat *i*-th col of weight matrix as word embedding for *i*-th word)

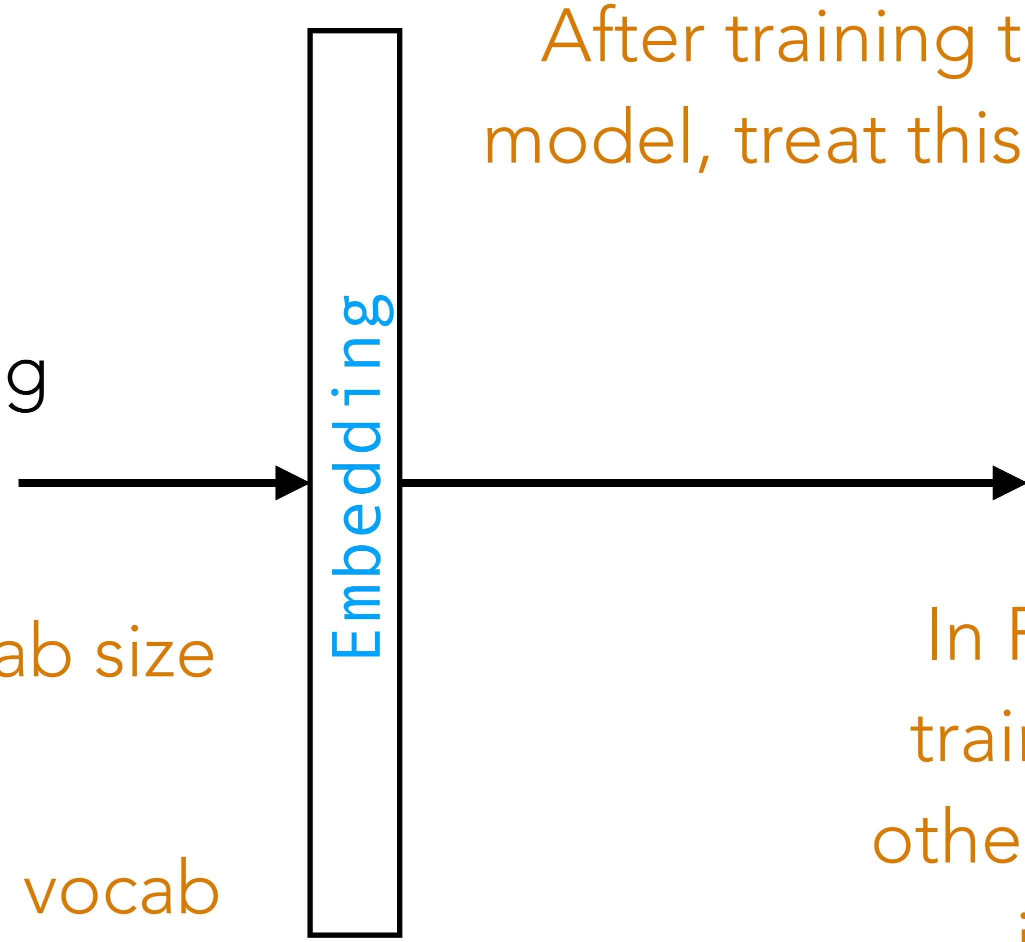
Word2vec Neural Net

After training the word2vec model, treat this layer as fixed!

"opioid"
Use one-hot encoding
[0, 0, …, 1, …, 0]

vector length = vocab size

index of "opioid" in vocab

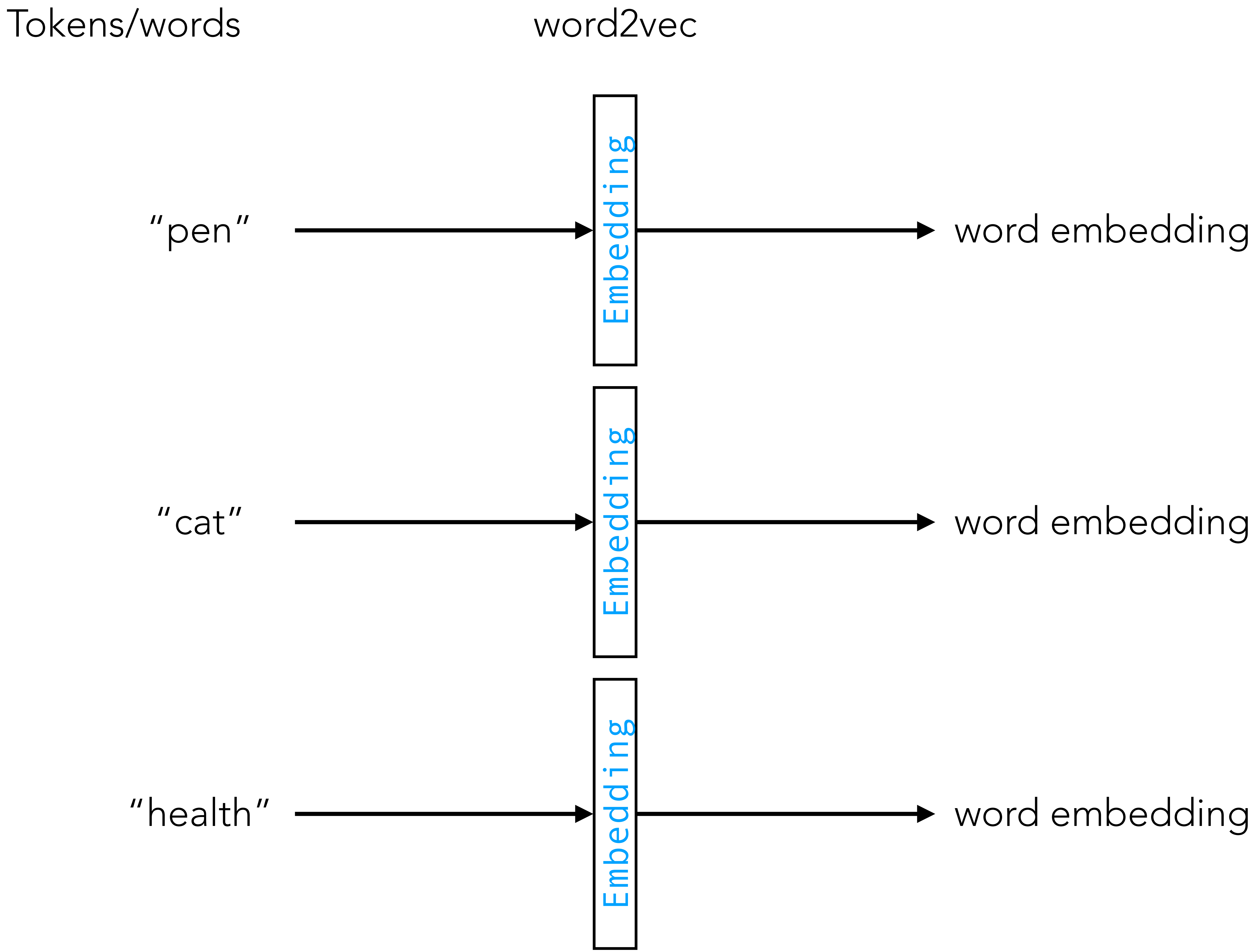


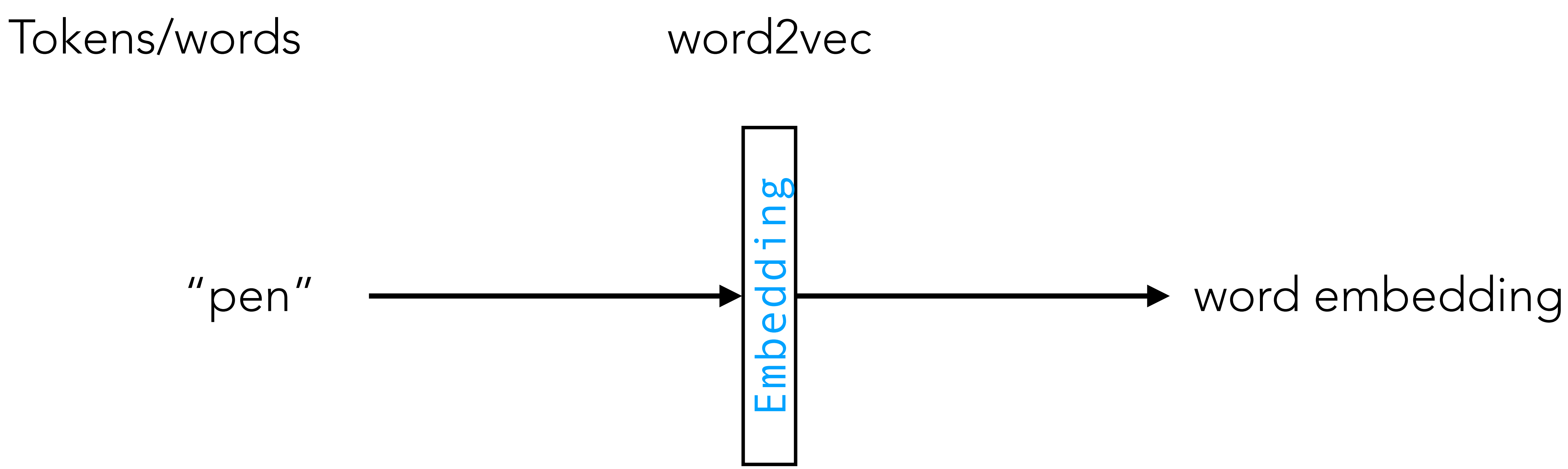
In PyTorch, can store already trained word2vec model (and other similar models like GloVe) in the `Embedding` layer

Linear, no bias vector
(100 nodes)

Learned weight matrix used as word embedding!

(Treat *i*-th col of weight matrix as word embedding for *i*-th word)


Tokens/words
word2vec
"pen"
Embedding
word embedding
"cat"
Embedding
word embedding
"health"
Embedding
word embedding



Even though "pen" has multiple meanings
(e.g., what you write with vs a play pen),
word2vec would produce the same word embedding for "pen"

(Flashback)

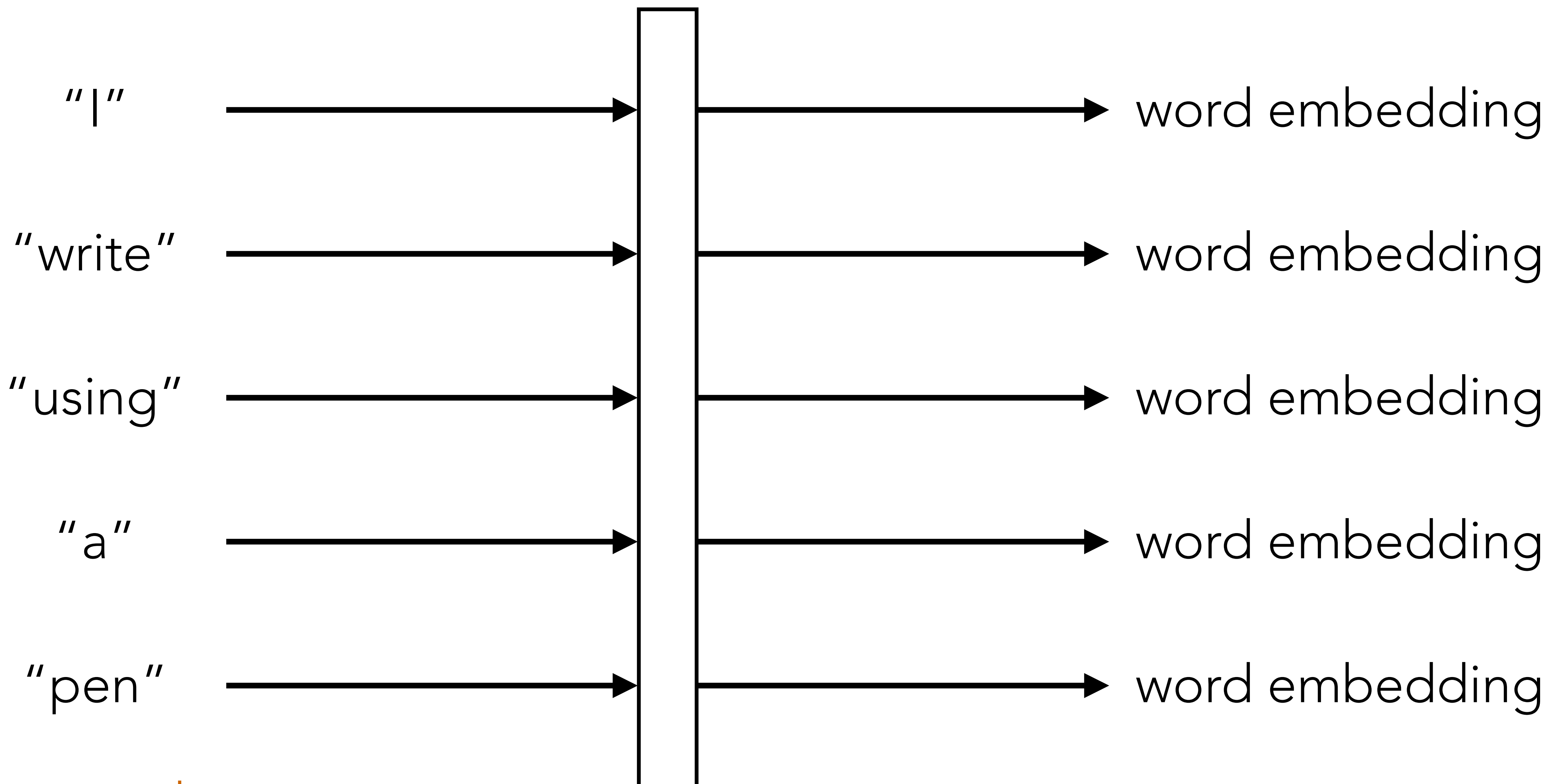
What about a word that has multiple meanings?

Challenging: try to split up word into multiple words depending on meaning (requires inferring meaning from context)

This problem is called word sense disambiguation (WSD)

Modern Word Embeddings Use Context

(such as BERT, which came out in 2018)

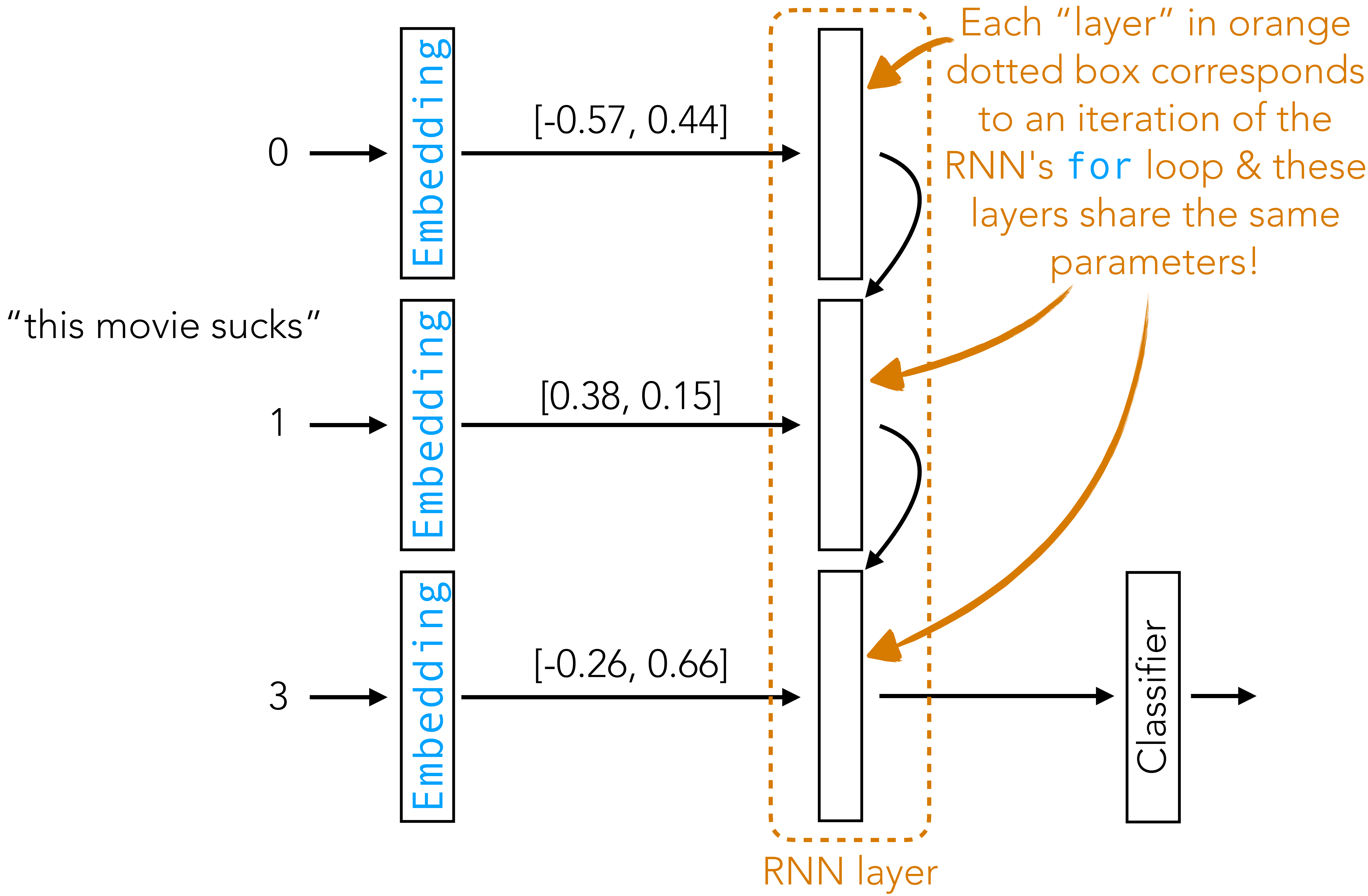


You provide a whole sentence (or a longer document)

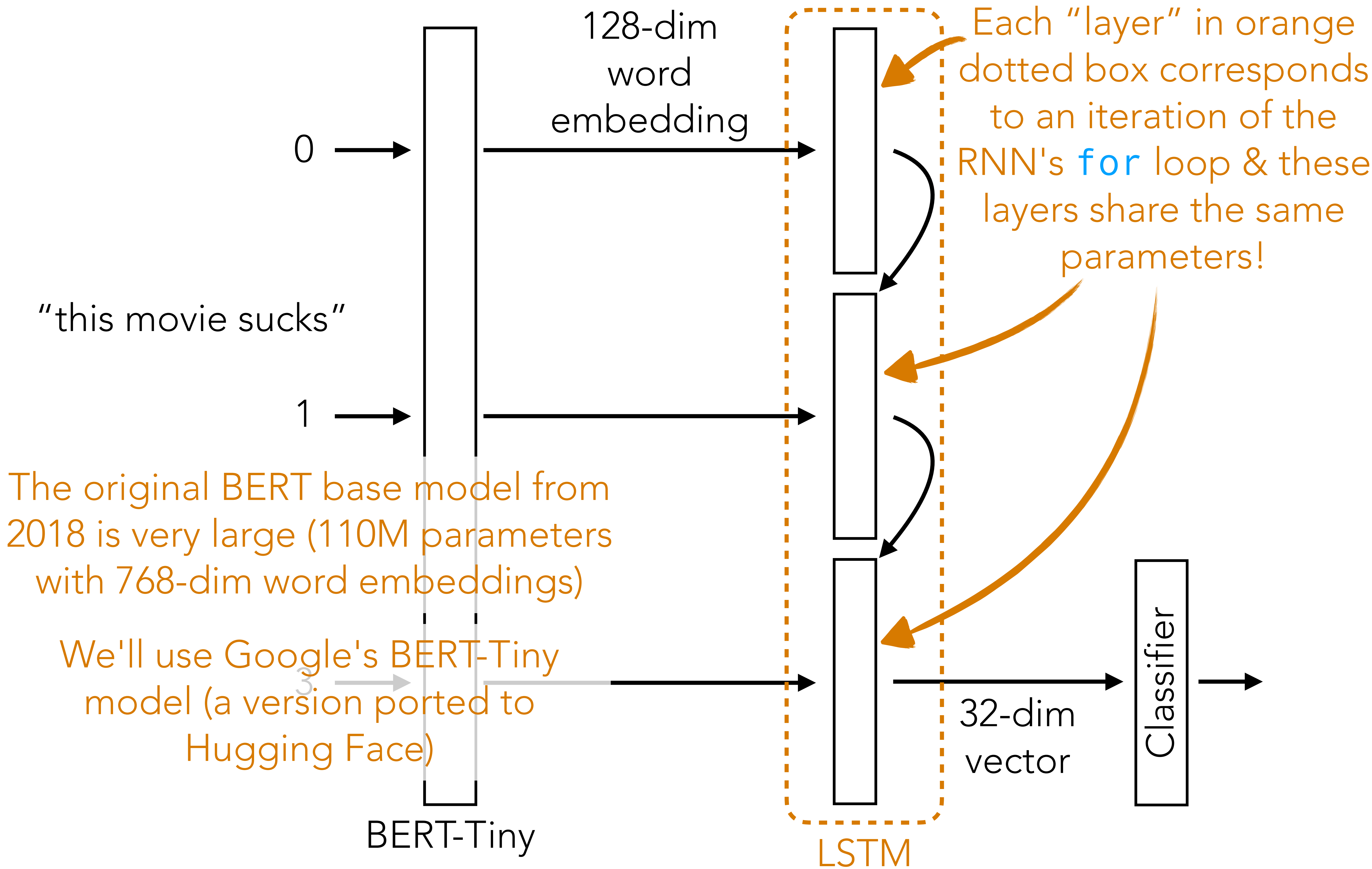
More complicated neural net (compared to applying `Embedding` separately to each word)

Time-permitting, we'll talk more about high level ideas of what happens in this neural net later this week

(Flashback) Sentiment Analysis with IMDb Reviews



What the Demo Will Actually Do



We do not store these embeddings and will instead compute them as needed (they depend on context anyways when using BERT/BERT-Tiny!)

Sentiment Analysis with IMDb Reviews

In the demo, use the vocabulary from a pre-trained BERT-Tiny

Training reviews

BERT/BERT-Tiny uses tokens that can be smaller than a word (specifically, unknown words get split into subwords)

Step 1: Tokenize & build vocabulary

Token ID ~~Word index~~	Token ~~Word~~	~~2D Embedding~~
0	this	~~[-0.57, 0.44]~~
1	movie	~~[0.38, 0.15]~~
2	rocks	~~[-0.85, 0.70]~~
3	sucks	~~[-0.26, 0.66]~~

Step 2: Encode each review as a sequence of ~~word indices into the vocab~~ token IDs

"this movie sucks" → 0 1 3

Step 3: Use word embeddings to represent each ~~word~~ token

~~[-0.57, 0.44]~~
~~[0.38, 0.15]~~
~~[-0.26, 0.66]~~

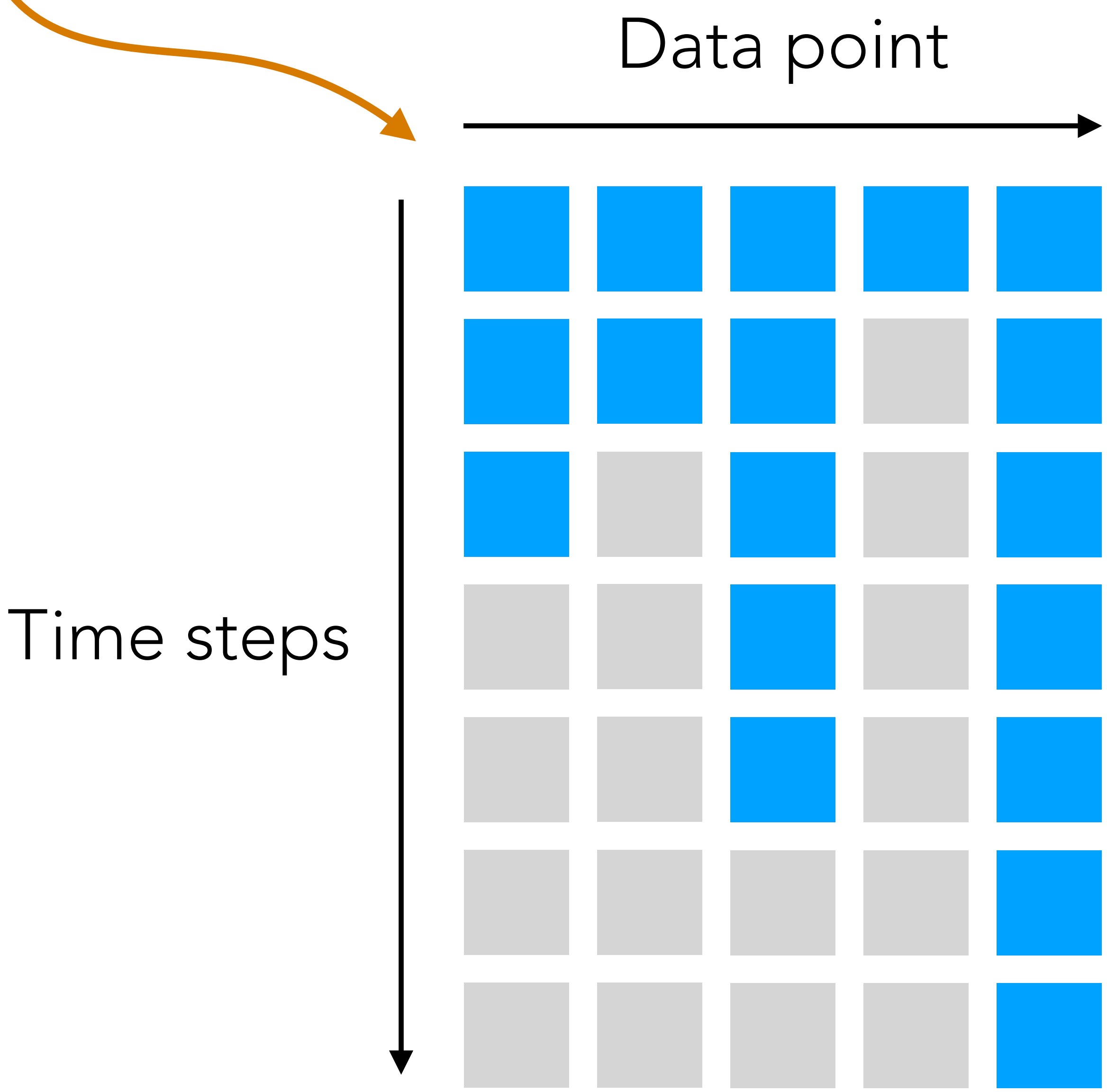
Each token represented as a 128-dim BERT-Tiny word embedding

Variable-Length Time Series in PyTorch

In PyTorch, how do we specify a batch of time series of varying lengths?

Common way: give a 2D table with all time series padded to the max length, and also give a 1D table specifying the lengths

Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7



[3, 2, 5, 1, 7]

Blue entries contain actual values from the 5 time series

Gray entries contain padded values (e.g., zeros)

This shows up in the demo when we specify an example input to the neural net

Sentiment Analysis with IMDb Reviews Demo

The next series of slides provide a “cheatsheet” explaining what the sentiment analysis demo is doing

I will <u>not</u> go over the demo in detail in class and will expect you to read it fully (I will go over the cheatsheet with you)

The demo does not use a vanilla ReLU RNN and instead uses an LSTM (you are not expected to know details of what’s under the hood for an LSTM)

Sentiment Analysis Demo Cheatsheet

Important: we do not build a vocabulary from scratch since we just use BERT-Tiny's vocabulary!

list of length-2 tuples each containing (review, label 0 or 1)

1. Load in training data (25000 IMDb reviews) `train_dataset`
2. Do a 80/20 split of the training data into:
   - proper training data (20000 reviews) `proper_train_dataset`
   - validation data (5000 reviews) `val_dataset`
3. Convert each <u>proper training</u> review into token IDs using BERT-Tiny's `encode` method

```
"Master cinéaste Alain Resnais likes to work with those actors"
```

↓

```
['master', 'ci', '##eas', '##te", 'alain', 'res', '##nais',
'likes', 'to', 'work', 'with', 'those', 'actors']
```

↓

```
[3040, 25022, 26737, 2618, 15654, 24501, 28020, 7777,
2000, 2147, 2007, 2216, 5889]
```

Important: we do not build a vocabulary from scratch since we just use BERT-Tiny's vocabulary!

list of length-2 tuples each containing (review, label 0 or 1)

1. Load in training data (25000 IMDb reviews) `train_dataset`

2. Do a 80/20 split of the training data into:
   - proper training data (20000 reviews) `proper_train_dataset`
   - validation data (5000 reviews) `val_dataset`

3. Convert each <u>proper training</u> review into token IDs using BERT-Tiny's `encode` method

```
"Master cinéaste Alain Resnais likes to work with those actors"
```

```
['master', 'ci', '##eas', '##te", 'alain', 'res', '##nais',
'likes', 'to', 'work', 'with', 'those', 'actors']
```

```
[3040, 25022, 26737, 2618, 15654, 24501, 28020, 7777,
2000, 2147, 2007, 2216, 5889]
```

`proper_train_dataset_encoded`

`val_dataset_encoded`

list of length-2 tuples each containing (encoded review, label 0 or 1) `proper_train_dataset_encoded`

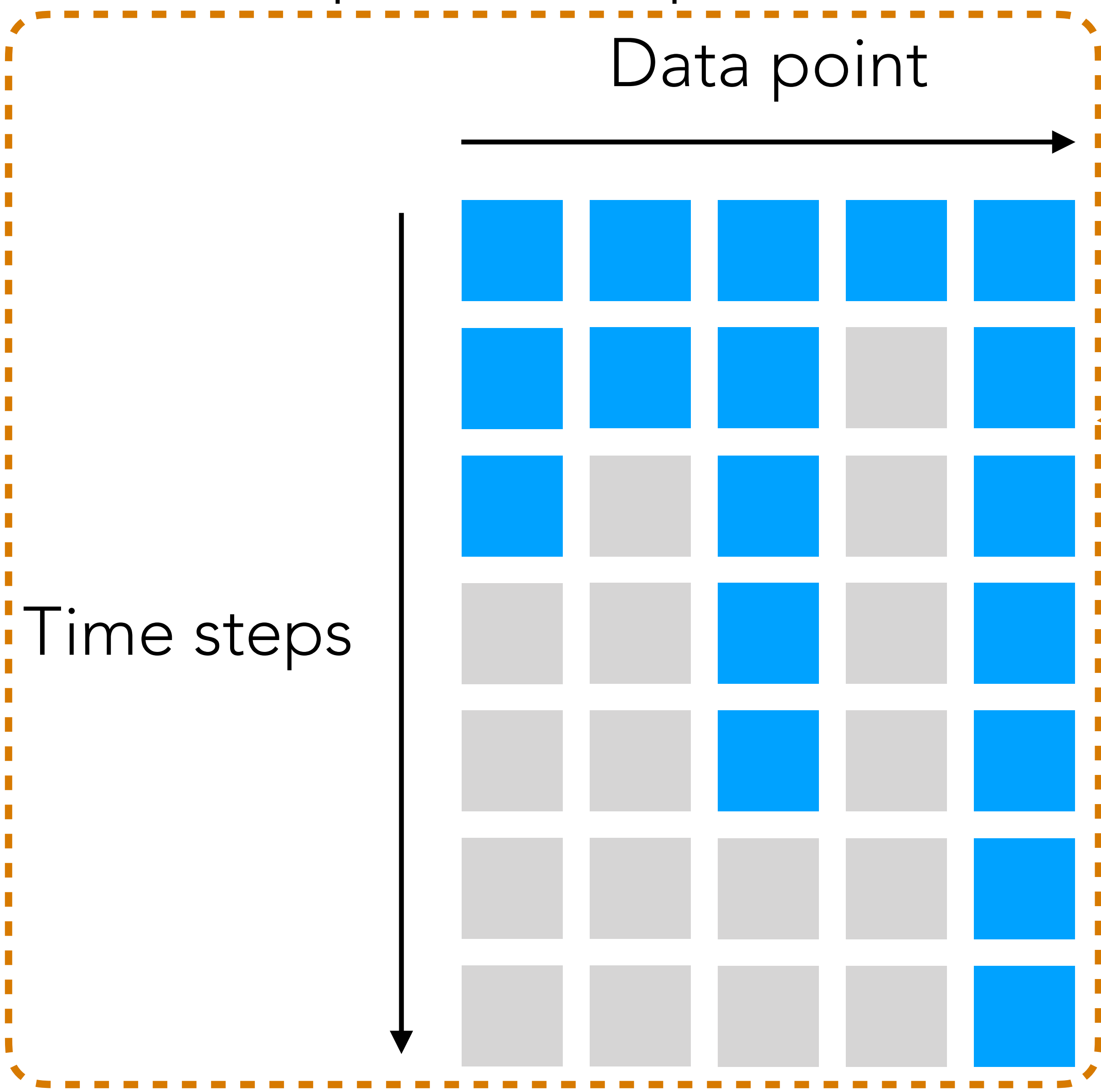
`val_dataset_encoded`

list of length-2 tuples each containing (encoded review, label 0 or 1)

4. Construct neural net (instead of `nn.Sequential`, we make a class that inherits from `nn.module`)

PyTorch convention: the `forward` function specifies how a neural net actually processes a batch of input data

Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7

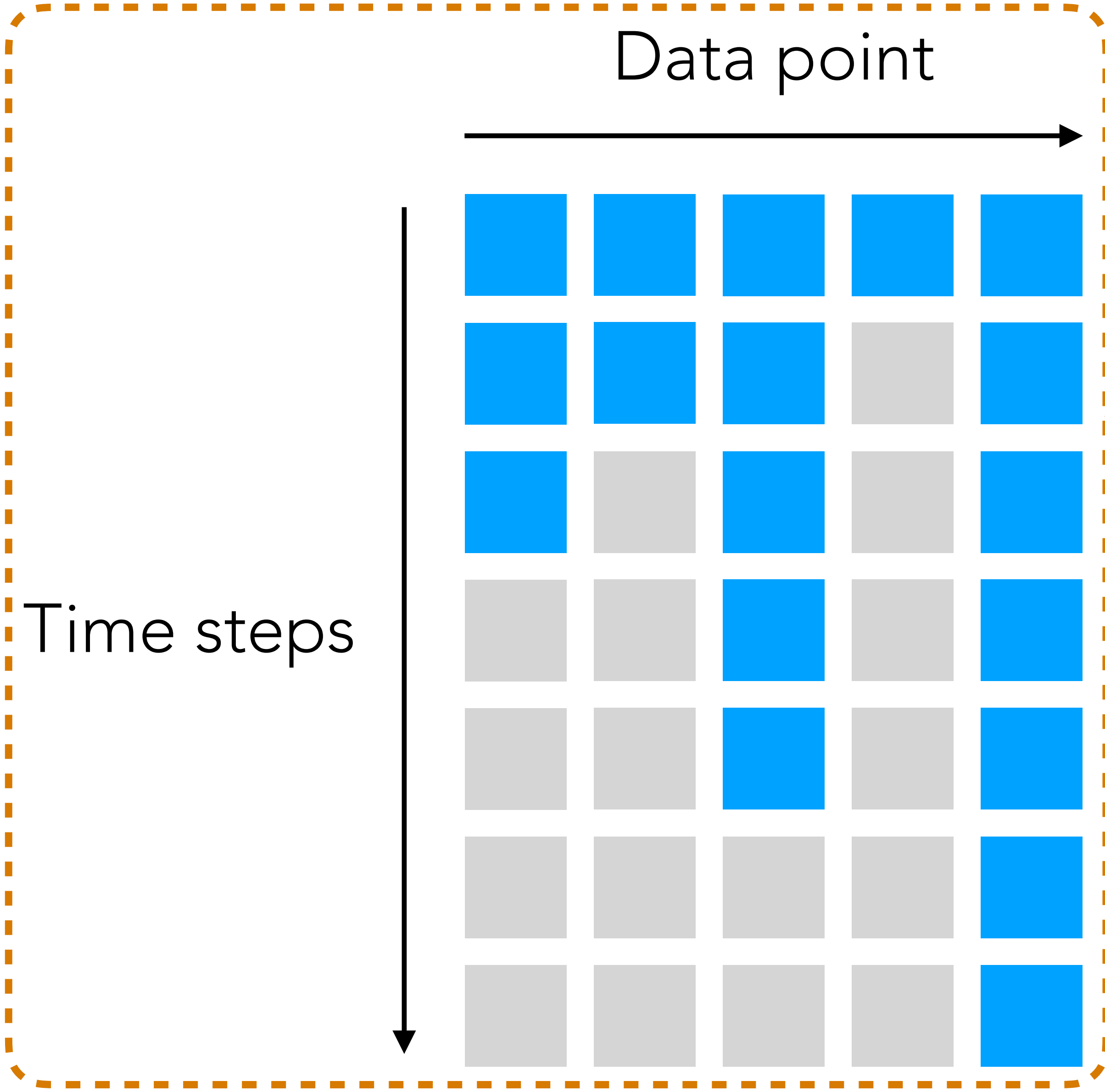


The neural net we constructed has a `forward` function with two inputs:
- a 2D table (each column is for 1 data point)
- a 1D table (specifies length for each time series)

Blue entries contain actual values from the 5 time series

Gray entries contain padded values (e.g., zeros)

Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7



The neural net we constructed has a `forward` function with two inputs:

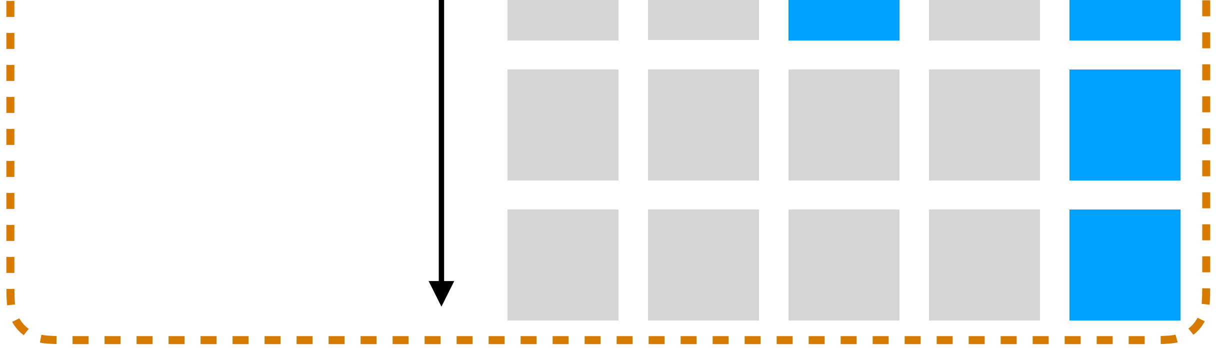
- a 2D table (each column is for 1 data point)
- a 1D table (specifies length for each time series)

Blue entries contain actual values from the 5 time series

Gray entries contain padded values (e.g., zeros)

```
# example where there are 5 input time series of lengths 3, 2, 5, 1, 7;
# we specify these time series using a 2D table that is padded and a
# 1D table of lengths (see lecture slides for details)
summary(word_embedding_lstm_linear_model,
        input_data=[torch.zeros((7, 5), dtype=torch.long),
                    torch.tensor([3, 2, 5, 1, 7], dtype=torch.long)])
```

Data types matter in PyTorch (`torch.long` means these tables store integers)



values from the 5 time series

Gray entries contain padded values (e.g., zeros)

```
# example where there are 5 input time series of lengths 3, 2, 5, 1, 7;
# we specify these time series using a 2D table that is padded and a
# 1D table of lengths (see lecture slides for details)
summary(word_embedding_lstm_linear_model,
        input_data=[torch.zeros((7, 5), dtype=torch.long),
                    torch.tensor([3, 2, 5, 1, 7], dtype=torch.long)])
```

Data types matter in PyTorch (`torch.long` means these tables store integers)

5. Train the neural net for some user-specified max number of epochs

6. Automatically tune on one hyperparameter:
   choose # of epochs to be the one achieving highest validation accuracy

7. Load in the saved neural net from the best # of epochs

8. Finally load in test data, tokenize and convert each test review into a list of integers, and use the trained neural net to predict

Two Demos

First demo (very short): How to use word embedding models from Hugging Face's `transformers` package

Second demo (long): sentiment analysis demo
(again, please actually read it carefully including the comments after class)

Text generation as a prediction problem

Just like the word2vec prediction problem:
we set up a self-supervised prediction problem

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters (so no need to use spaCy)

Given `['T']`, predict next character `'h'`

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Let's treat this string as a <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters
(so no need to use spaCy)

Given `['T']`, predict next character `'h'`

Given `['T', 'h']`, predict next character `'e'`

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters
(so no need to use spaCy)

Given `['T']`, predict next character `'h'`

Given `['T', 'h']`, predict next character `'e'`

Given `['T', 'h', 'e']`, predict next character `' '`

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Let's treat this string as a <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters
(so no need to use spaCy)

Given `['T']`, predict next character `'h'`

Given `['T', 'h']`, predict next character `'e'`

Given `['T', 'h', 'e']`, predict next character `' '`

Given `['T', 'h', 'e', ' ']`, predict next character `'o'`

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Let's treat this string as a <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters
(so no need to use spaCy)

Given `['T']`, predict next character `'h'`

Given `['T', 'h']`, predict next character `'e'`

Given `['T', 'h', 'e']`, predict next character `' '`

Given `['T', 'h', 'e', ' ']`, predict next character `'o'`

…

If the string has $L + 1$ characters total, then there are L such prediction tasks

How to solve this prediction task with an RNN

We will now keep track of outputs at every time step of the RNN

(Previously for sentiment analysis, we only kept the output at the final time step)

Vocabulary

First, let's agree on a vocabulary to use
(e.g., pick the unique ones seen in the dataset)

```
vocabulary
```

```
array(['\n', ' ', '"', '$', '%', '&', "'", '(', ')', ',', '-', '.', '/',
       '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '?',
       'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
       'N', 'O', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', '[', ']',
       '^', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
       'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
       'z', '|', '—', '–', '‘', '’', '“', '”'], dtype='<U1')
```

```
len(vocabulary)
```

```
86
```

```
token_to_id = {token: idx for idx, token in enumerate(vocabulary)}
```

```
def encode(s):
    assert type(s) == str
    return torch.tensor([token_to_id[character] for character in s], dtype=torch.long)
```

```
encode('The opi')
```

```
tensor([44, 60, 57,  1, 67, 68, 61])
```

RNN Language Model

```
['T', 'h', 'e', ' ', 'o', 'p', 'i']
```

length = $L + 1$

$L = 6$ in this example

RNN Language Model