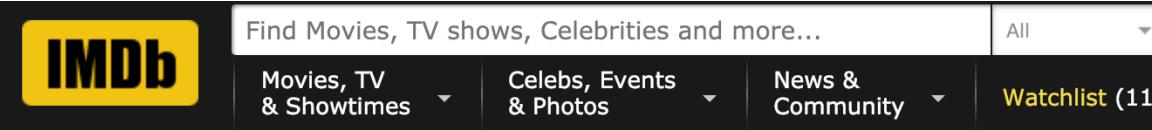


# Text Processing: Preparations for RNNs

Shusen Wang

# The IMDB Movie Review Dataset



Find Movies, TV shows, Celebrities and more... All  
Movies, TV & Showtimes Celebs, Events & Photos News & Community Watchlist (11)



**Avengers: Infinity War** (2018)

## User Reviews

+ Review this title

3,693 Reviews

Hide Spoilers

Filter by Rating: Show All

Sort by: Helpfulness



10/10

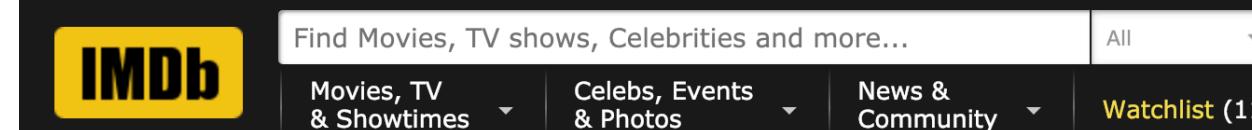
**This movie will blow your mind and break your heart - and make you desperate to go back for more. Brave, brilliant and better than it has any right to be.**

shawneofthedead 25 April 2018

### Warning: Spoilers

Over the past decade, Marvel has earned itself the benefit of the doubt. The studio has consistently delivered smart, funny, brave films that both embrace and transcend their comic-book origins. The 18 blockbuster movies produced since Iron Man first blasted off into the stratosphere in 2008 have not only reinvented superhero films as a genre - they've helped to legitimise it. Indeed, Marvel's two most recent films - Thor: Ragnarok and Black Panther - have received the kind of accolades usually reserved for edgy arthouse flicks.

And yet, it's perfectly reasonable to be apprehensive about Avengers: Infinity War. This is a blockbuster film that's been ten years in the making. Its plot hinted at and



Find Movies, TV shows, Celebrities and more... All  
Movies, TV & Showtimes Celebs, Events & Photos News & Community Watchlist (11)



**Star Wars: Episode VIII - The Last Jedi** (2017)

## User Reviews

+ Review this title

5,796 Reviews

Hide Spoilers

Filter by Rating: Show All

Sort by: Helpfulness



1/10

**The Last Jedi was just magical**

yupman 2 January 2018

### Warning: Spoilers

SPOILER: This movie was just magical.

The Force has become like Harry Potter magic, with a new spell every day or so.

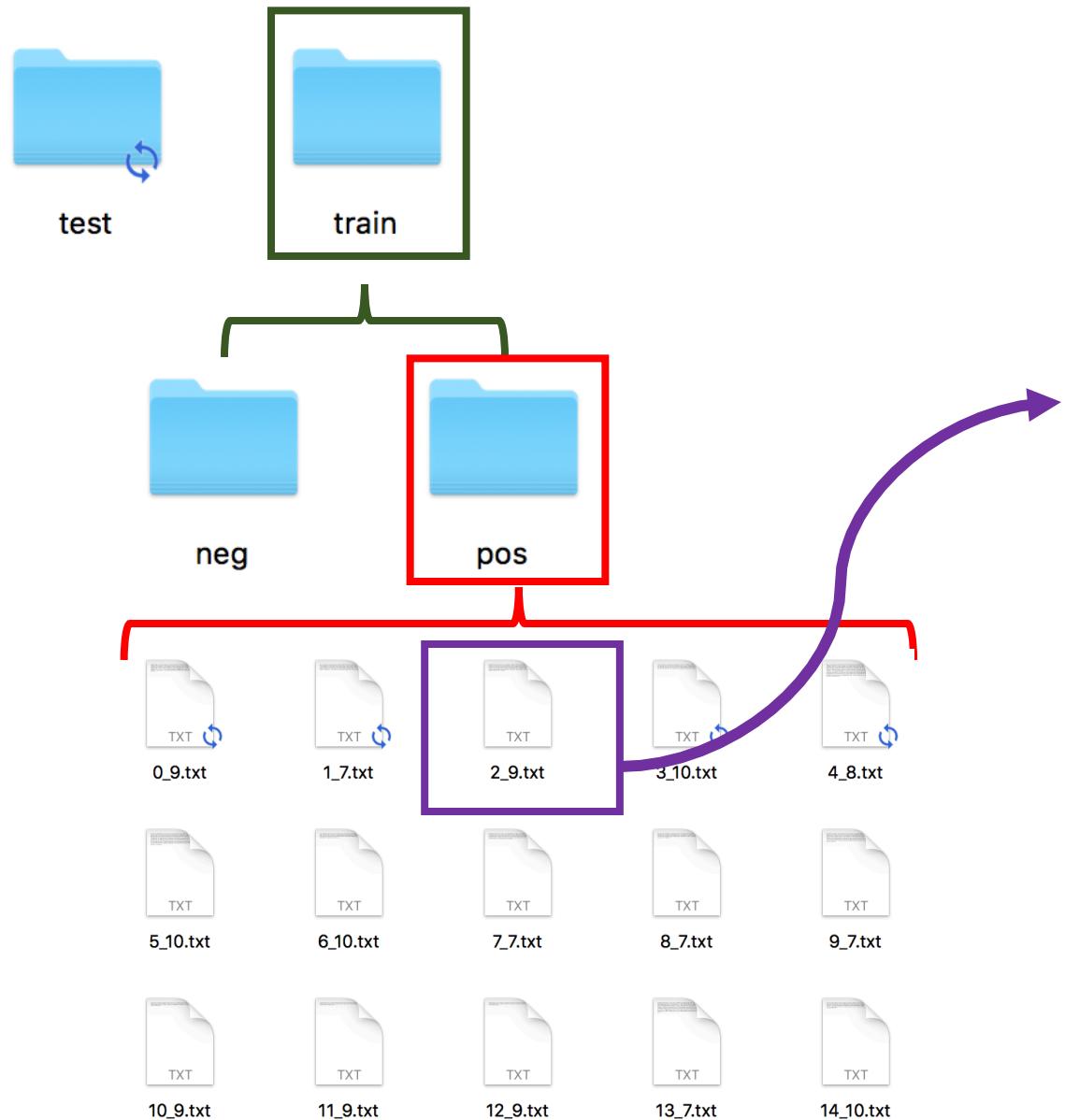
Rey is just magical. With practically training whatsoever she can take on Luke and Kylo and win.

Snoke is magical. He appeared out of nowhere in The Force Awakens and in The Last Jedi has such fantastic Force powers (never before seen in the entire Star

# The IMDB Movie Review Dataset

- 50K movie reviews (text).
- Each review is labeled with either “positive” or “negative”.
- It is a binary classification problem.
- 25K for training and 25K for test.
- Download from
  - <http://ai.stanford.edu/~amaas/data/sentiment/>
  - <http://s3.amazonaws.com/text-datasets/aclImdb.zip>

# The IMDB Movie Review Dataset



2\_9.txt

Bromwell High is nothing short of brilliant. Expertly scripted and perfectly delivered, this searing parody of a students and teachers at a South London Public School leaves you literally rolling with laughter. It's vulgar, provocative, witty and sharp. The characters are a superbly caricatured cross section of British society (or to be more accurate, of any society). Following the escapades of Keisha, Latrina and Natella, our three "protagonists" for want of a better term, the show doesn't shy away from parodying every imaginable subject. Political correctness flies out the window in every episode. If you enjoy shows that aren't afraid to poke fun of every taboo subject imaginable, then Bromwell High will not disappoint!

# Texts to Sequences

texts[i] :

"For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt."



Tokenization

tokens[i] :

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a',
'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'imagine', 'a',
'movie', 'where', 'joe', 'piscopo', 'is', 'actually', 'funny', 'maureen',
'stapleton', 'is', 'a', 'scene', 'stealer', 'the', 'moroni', 'character', 'is',
'an', 'absolute', 'scream', 'watch', 'for', 'alan', 'the', 'skipper', 'hale', 'jr',
'as', 'a', 'police', 'sgt']
```



Encoding

seqs[i] :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2, 3, 19, 20,
21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33, 34, 35, 1, 36, 29,
37, 38, 39, 40, 2, 41, 42]
```

# Preliminaries of Text Processing

# Represent Text by Vectors/Matrices

**Goal:** Build a binary classifier for predicting the sentiment of a movie review.

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

- Why representing text by numeric vectors/matrices?
  - ML models do not “*understand*” text.
  - The input must be numeric vectors, matrices, or tensors.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 1: Load the text and labels to memory.

```
import os
imdb_dir = './aclImdb'
train_dir = os.path.join(imdb_dir, 'train')
labels_train = []
texts_train = []
for label_type in ['pos', 'neg']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts_train.append(f.read())
            f.close()
            if label_type == 'neg':
                labels_train.append(0)
            else:
                labels_train.append(1)
```

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Results:**

- `texts_train`: `list[string]`.
  - Each `string` is a movie review.
- `labels_train`: `list[int]`.
  - “1” means positive, and “0” means negative.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Results:**

```
print('Number of training samples: ' + str(len(texts_train)))
print('Number of training labels: ' + str(len(labels_train)))
```

```
Number of training samples: 25000
Number of training labels: 25000
```

```
i = 0
print('label #' + str(i) + ': ' + str(labels_train[i]))
print('text #' + str(i) + ':')
print(texts_train[i])
```

```
label #0: 1
text #0:
For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt.
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

### 1. Split the text to tokens (word).

- `tokens_train`: `list[list[string]]`.
- Each `string` is a token (word).

```
tokens_train = []
for samples in texts_train:
    token_list = samples.split()
    token_list = [token.strip(',.!') for token in token_list]
    tokens_train.append(token_list)
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

### 1. Split the text to tokens (word).

- `tokens_train`: `list[list[string]]`.
- Each `string` is a token (word).

```
print(texts_train[i])
```

For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt.

```
print(tokens_train[i])
```

```
['For', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a', 'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'Imagine', 'a', 'movie', 'where', 'Joe', 'Piscopo', 'is', 'actually', 'funny', 'Maureen', 'Stapleton', 'is', 'a', 'scene', 'stealer', 'The', 'Moroni', 'character', 'is', 'an', 'absolute', 'scream', 'Watch', 'for', 'Alan', 'The', 'Skipper', 'Hale', 'jr', 'as', 'a', 'police', 'Sgt']
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

1. Split the text to tokens (word).
2. Build a **dictionary** of tokens.
  - `token_index: dict[(string, int)]`.
  - Each `string` is a token (word).
  - Each `int` is an index.

```
token_index = {}  
for token_list in tokens_train:  
    for token in token_list:  
        if token not in token_index:  
            token_index[token] = len(token_index) + 1
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

1. Split the text to tokens (word).

2. Build a **dictionary** of tokens.

- `token_index`: `dict[ (string, int) ]`.
- Each `string` is a token (word).
- Each `int` is an index.
- The vocabulary size (number of unique tokens) is 175.8K.

```
print(len(token_index))
```

175837

`token_index`

`Out[10]:`

```
{'for': 1,  
 'a': 2,  
 'movie': 3,  
 'that': 4,  
 'gets': 5,  
 'no': 6,  
 'respect': 7,  
 'there': 8,  
 'sure': 9,  
 'are': 10,
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

1. Split the text to tokens (word).
2. Build a dictionary of tokens.
3. Remove the **stop words** (optional).
  - The most frequent words such as “the”, “a”, “to”, etc.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

1. Split the text to tokens (word).
2. Build a dictionary of tokens.
3. Remove the stop words (optional).

All the training data we need:

- `tokens_train`: `list[list[string]]`.
- `labels_train`: `list[int]`.
- `token_index`: `dict[(string, int)]`.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

## Step 2: Tokenize the text.

1. Split the text to tokens (word).
2. Build a dictionary of tokens.
3. Remove the stop words (optional).

All the training data we need:

- `tokens_train`: `list[list[string]]`  `list[list[int]]`.
- `labels_train`: `list[int]`.
- `token_index`: `dict[(string, int)]`.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 3:** Encode the text: represent each token (word) by its index.

sequence\_train: list[list[int]].

```
sequences_train = []
for token_list in tokens_train:
    sequence = [token_index[token] for token in token_list]
    sequences_train.append(sequence)
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 3:** Encode the text: represent each token (word) by its index.

sequence\_train: list[list[int]].

```
print(tokens_train[i])
```

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure',
'are', 'a', 'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this',
'gem', 'imagine', 'a', 'movie', 'where', 'joe', 'piscopo', 'is', 'actuall
y', 'funny', 'maureen', 'stapleton', 'is', 'a', 'scene', 'stealer', 'th
e', 'moroni', 'character', 'is', 'an', 'absolute', 'scream', 'watch', 'fo
r', 'alan', 'the', 'skipper', 'hale', 'jr', 'as', 'a', 'police', 'sgt']
```

```
print(sequences_train[i])
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2,
3, 19, 20, 21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33,
34, 35, 1, 36, 29, 37, 38, 39, 40, 2, 41, 42]
```

token\_index

Out[10]:

```
{'for': 1,
'a': 2,
'movie': 3,
'that': 4,
'gets': 5,
'no': 6,
'respect': 7,
'there': 8,
'sure': 9,
'are': 10}
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 3:** Encode the text: represent each token (word) by its index.

sequence\_train: list[list[int]].

```
print(tokens_train[i])
```

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure',
'are', 'a', 'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this',
'gem', 'imagine', 'a', 'movie', 'where', 'joe', 'piscopo', 'is', 'actuall
y', 'funny', 'maureen', 'stapleton', 'is', 'a', 'scene', 'stealer', 'th
e', 'moroni', 'character', 'is', 'an', 'absolute', 'scream', 'watch', 'fo
r', 'alan', 'the', 'skipper', 'hale', 'jr', 'as', 'a', 'police', 'sgt']
```

```
print(sequences_train[i])
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2,
3, 19, 20, 21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33,
34, 35, 1, 36, 29, 37, 38, 39, 40, 2, 41, 42]
```

token\_index

Out[10]:

```
{ 'for': 1,
'a': 2,
'movie': 3,
'that': 4,
'gets': 5,
'no': 6,
'respect': 7,
'there': 8,
'sure': 9,
'are': 10,
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 3:** Encode the text: represent each token (word) by its index.

sequence\_train: list[list[int]].

```
print(tokens_train[i])
```

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure',
'are', 'a', 'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this',
'gem', 'imagine', 'a', 'movie', 'where', 'joe', 'piscopo', 'is', 'actuall
y', 'funny', 'maureen', 'stapleton', 'is', 'a', 'scene', 'stealer', 'th
e', 'moroni', 'character', 'is', 'an', 'absolute', 'scream', 'watch', 'fo
r', 'alan', 'the', 'skipper', 'hale', 'jr', 'as', 'a', 'police', 'sgt']
```

```
print(sequences_train[i])
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2,
3, 19, 20, 21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33,
34, 35, 1, 36, 29, 37, 38, 39, 40, 2, 41, 42]
```

token\_index

Out[10]:

```
{'for': 1,
'a': 2,
'movie': 3,
'that': 4,
'gets': 5,
'no': 6,
'respect': 7,
'there': 8,
'sure': 9,
'are': 10,
```

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Step 2:** Tokenize the text.

**Step 3:** Encode the text.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Step 2:** Tokenize the text.

**Step 3:** Encode the text.

```
texts[i] = "the cat sat on the mat."
```

**Tokenization**

```
tokens[i] = ["the", "cat",  
"sat", "on", "the", "mat"]
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Step 2:** Tokenize the text.

**Step 3:** Encode the text.

```
texts[i] = "the cat sat on the mat."
```

**Tokenization**

```
tokens[i] = ["the", "cat",  
"sat", "on", "the", "mat"]
```

```
token_index = {"the": 1, "cat":  
2, "sat": 3, "on": 4, "mat": 5, ...}
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Step 2:** Tokenize the text.

**Step 3:** Encode the text.

```
texts[i] = "the cat sat on the mat."
```

**Tokenization**

```
tokens[i] = ["the", "cat",  
"sat", "on", "the", "mat"]
```

**Encoding**

```
token_index = {"the": 1, "cat":  
2, "sat": 3, "on": 4, "mat": 5, ...}
```

```
sequences[i] = [1, 2, 3, 4, 1, 5]
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Step 1:** Load the text and labels to memory.

**Step 2:** Tokenize the text.

**Step 3:** Encode the text.

**Step 4:** Do the same for the test set.

- Use the **same dictionary** built on the **training data**.
- The dictionary for training and test must be **the same!**

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

**Result:** 50K lists of integers; the  $i$ -th list has  $w_i$  elements.

**texts [0]**

For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt.



**sequences [0]**

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13,  
14, 15, 1, 16, 17, 18, 2, 3, 19, 20, 21, 22,  
23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22,  
32, 33, 34, 35, 1, 36, 29, 37, 38, 39, 40, 2,  
41, 42]
```

**texts [5]**

I saw the movie with two grown children. Although it was not as clever as Shrek, I thought it was rather good. In a movie theatre surrounded by children who were on spring break, there was not a sound so I know the children all liked it. Their parents also seemed engaged. The death and apparent death of characters brought about the appropriate gasps and comments. Hopefully people realize this movie was made for kids. As such, it was successful although I liked it too. Personally I liked the Scrat!!



**sequences [5]**

```
[178, 486, 29, 3, 46, 407, 487, 488, 489, 272,  
160, 273, 40, 490, 40, 491, 178, 492, 272, 160,  
493, 494, 193, 2, 3, 495, 496, 51, 488, 64,  
385, 97, 497, 498, 8, 160, 273, 2, 499, 459,  
178, 335, 29, 488, 293, 500, 272, 8, 196, 357,  
501, 502, 29, 263, 110, 503, 263, 12, 504, 141,  
391, 29, 505, 506, 110, 507, 508, 509, 510, 16,  
3, 160, 511, 1, 199, 40, 377, 272, 160, 512,  
489, 178, 500, 272, 513, 514, 178, 500, 29,  
515]
```

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Result:**  $50K$  lists of integers; the  $i$ -th list has  $w_i$  elements.

**texts [0]**

For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt.



**sequences [0]**

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13,  
14, 15, 1, 16, 17, 18, 2, 3, 19, 20, 21, 22,  
23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22,  
32, 33, 34, 35, 1, 36, 29, 37, 38, 39, 40, 2,  
41, 42]
```

$w_0 = 52$  tokens

**texts [5]**

I saw the movie with two grown children. Although it was not as clever as Shrek, I thought it was rather good. In a movie theatre surrounded by children who were on spring break, there was not a sound so I know the children all liked it. Their parents also seemed engaged. The death and apparent death of characters brought about the appropriate gasps and comments. Hopefully people realize this movie was made for kids. As such, it was successful although I liked it too. Personally I liked the Scrat!!



**sequences [5]**

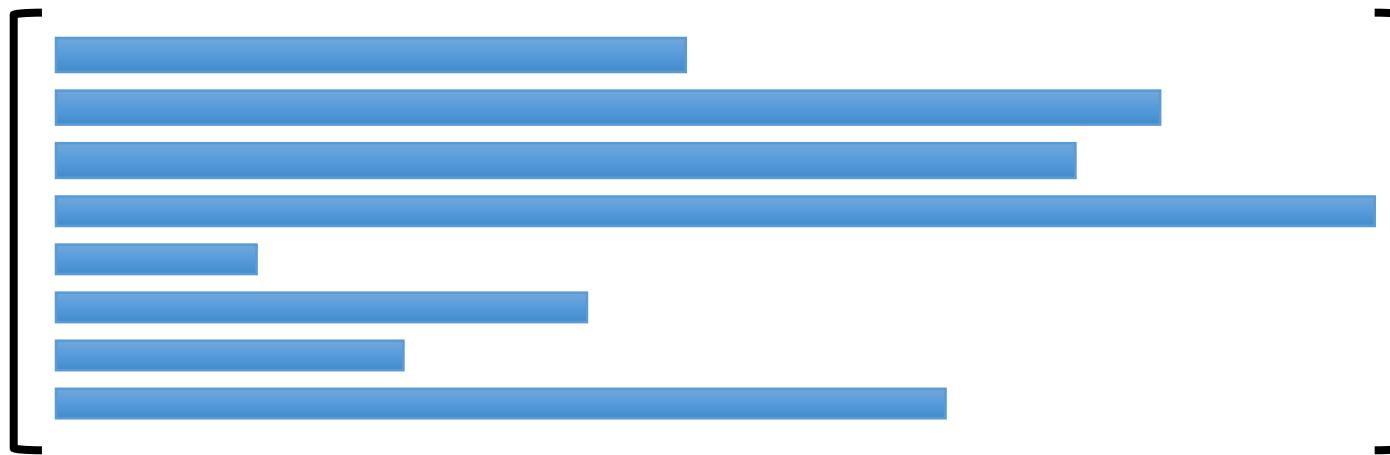
```
[178, 486, 29, 3, 46, 407, 487, 488, 489, 272,  
160, 273, 40, 490, 40, 491, 178, 492, 272, 160,  
493, 494, 193, 2, 3, 495, 496, 51, 488, 64,  
385, 97, 497, 498, 8, 160, 273, 2, 499, 459,  
178, 335, 29, 488, 293, 500, 272, 8, 196, 357,  
501, 502, 29, 263, 110, 503, 263, 12, 504, 141,  
391, 29, 505, 506, 110, 507, 508, 509, 510, 16,  
3, 160, 511, 1, 199, 40, 377, 272, 160, 512,  
489, 178, 500, 272, 513, 514, 178, 500, 29,
```

$w_5 = 90$  tokens

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Result:**  $50K$  lists of integers; the  $i$ -th list has  $w_i$  elements.

**Problem:** the training samples are not aligned (have different lengths,  $w_i$ ).



**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Result:**  $50K$  lists of integers; the  $i$ -th list has  $w_i$  elements.

**Problem:** the training samples are not aligned (have different lengths,  $w_i$ ).

**Solution:**

- Cut off the texts to keep  $w$  words, e.g.,  $w = 7$ .

“the fat cat sat still on the big red mat.”



“sat still on the big red mat.”

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Result:**  $50K$  lists of integers; the  $i$ -th list has  $w_i$  elements.

**Problem:** the training samples are not aligned (have different lengths,  $w_i$ ).

**Solution:**

- Cut off the texts to keep  $w$  words, e.g.,  $w = 7$ .
- If a text is shorter than  $w$ , pad it with zeros.

“the fat cat sat still on the big red mat.”



“sat still on the big red mat.”

“cat sat on the mat.”

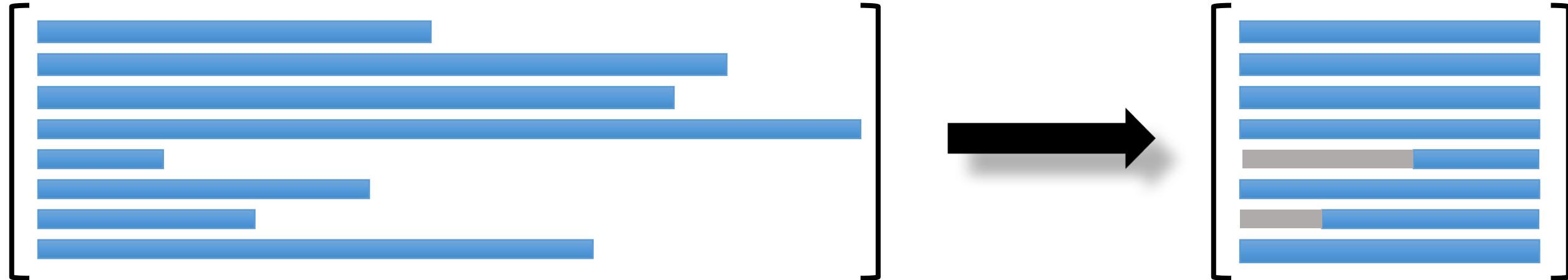


“null null cat sat on the mat.”

**Task:** Convert the  $50K$  movie reviews (text) to  $50K$  vectors (sequences).

**Result:**  $50K$  lists of integers; every list has  $w$  elements.

Aligned!



# Text Processing in Keras

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

Convert texts to sequences by Keras.

```
from keras.preprocessing.text import Tokenizer  
  
vocabulary = 10000  
tokenizer = Tokenizer(num_words=vocabulary)  
tokenizer.fit_on_texts(texts_train)  
  
word_index = tokenizer.word_index  
sequences_train = tokenizer.texts_to_sequences(texts_train)
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

Convert texts to sequences by Keras.

```
from keras.preprocessing.text import Tokenizer  
  
vocabulary = 10000  
tokenizer = Tokenizer(num_words=vocabulary)  
tokenizer.fit_on_texts(texts_train)
```

1. Tokenize the movie reviews
2. Build a dictionary.

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

Convert texts to sequences by Keras.

```
from keras.preprocessing.text import Tokenizer

vocabulary = 10000
tokenizer = Tokenizer(num_words=vocabulary)
tokenizer.fit_on_texts(texts_train)

word_index = tokenizer.word_index
sequences_train = tokenizer.texts_to_sequences(texts_train)
```

- word\_index: dict[(string, int)]
- sequence\_train: list[list[int]]

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

Convert texts to sequences by Keras.

```
from keras.preprocessing.text import Tokenizer

vocabulary = 10000
tokenizer = Tokenizer(num_words=vocabulary)
tokenizer.fit_on_texts(texts_train)

word_index = tokenizer.word_index
sequences_train = tokenizer.texts_to_sequences(texts_train)

print(sequences_train[0])

[15, 3, 17, 12, 211, 54, 1158, 47, 249, 23, 3, 173, 4, 903, 4381, 3559, 15, 11, 1525, 835, 3,
17, 118, 911, 6, 162, 160, 7262, 6, 3, 133, 1, 106, 6, 32, 1552, 2032, 103, 15, 1605, 1, 859
5, 1789, 14, 3, 565, 6259]
```

**Task:** Convert the 50K movie reviews (text) to 50K vectors (sequences).

Convert texts to sequences by Keras.

```
from keras import preprocessing

word_num = 20
x_train = preprocessing.sequence.pad_sequences(sequences_train, maxlen=word_num)
```

```
x_train.shape
```

```
(25000, 20)
```

```
x_train[0]
```

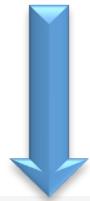
```
array([7262,      6,      3,    133,      1,    106,      6,      32,   1552,   2032,     103,
       15,  1605,      1,  8595,  1789,     14,      3,    565,   6259], dtype=int32)
```

**Task:** Convert the **50K** movie reviews (text) to **50K** vectors (sequences).

**Result:** A matrix of size **25K × word\_num** (training). (The test is analogous.)

```
texts_train[0]
```

'For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt.'



```
x_train[0]
```

```
array([7262,      6,      3,    133,      1,   106,      6,     32,  1552, 2032,    103,
       15, 1605,      1,  8595, 1789,     14,      3,    565,  6259], dtype=int32)
```

# Texts to Sequences: Summary

texts[0] :

"For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt."



Tokenization

tokens[0] :

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a',
'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'imagine', 'a',
'movie', 'where', 'joe', 'piscopo', 'is', 'actually', 'funny', 'maureen',
'stapleton', 'is', 'a', 'scene', 'stealer', 'the', 'moroni', 'character', 'is',
'an', 'absolute', 'scream', 'watch', 'for', 'alan', 'the', 'skipper', 'hale', 'jr',
'as', 'a', 'police', 'sgt']
```

# Texts to Sequences: Summary

texts [0] :

"For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt."



**Tokenization**

tokens [0] :

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a',
'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'imagine', 'a',
'movie', 'where', 'joe', 'piscopo', 'is', 'actually', 'funny', 'maureen',
'stapleton', 'is', 'a', 'scene', 'stealer', 'the', 'moroni', 'character', 'is',
'an', 'absolute', 'scream', 'watch', 'for', 'alan', 'the', 'skipper', 'hale', 'jr',
'as', 'a', 'police', 'sgt']
```



**Encoding**

seqs [0] :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2, 3, 19, 20,
21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33, 34, 35, 1, 36, 29,
37, 38, 39, 40, 2, 41, 42]
```

# Texts to Sequences: Summary

texts [0] :

"For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt."



Tokenization

tokens [0] :

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a',
'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'imagine', 'a',
'movie', 'where', 'joe', 'piscopo', 'is', 'actually', 'funny', 'maureen',
'stapleton', 'is', 'a', 'scene', 'stealer', 'the', 'moroni', 'character', 'is',
'an', 'absolute', 'scream', 'watch', 'for', 'alan', 'the', 'skipper', 'hale', 'jr',
'as', 'a', 'police', 'sgt']
```



Encoding

seqs [0] :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2, 3, 19, 20,
21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33, 34, 35, 1, 36, 29,
37, 38, 39, 40, 2, 41, 42]
```

Alignment (keep the last  $w = 20$  tokens)

# Word Embedding: Word to Vector

# Represent Text by Vectors/Matrices

**Goal:** Build a binary classifier for predicting the sentiment of a movie review.

**Task 1:** Convert the  $50K$  movie reviews (text) to  $50K$  sequences.

**Texts:**  $50K$  movie reviews



**Sequences:**  $50K \times \text{word\_num}$  integer matrix

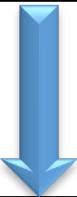
# Represent Text by Vectors/Matrices

**Goal:** Build a binary classifier for predicting the sentiment of a movie review.

**Task 1:** Convert the  $50K$  movie reviews (text) to  $50K$  sequences.

**Task 2:** Word embedding: an index (of word) to a vector.

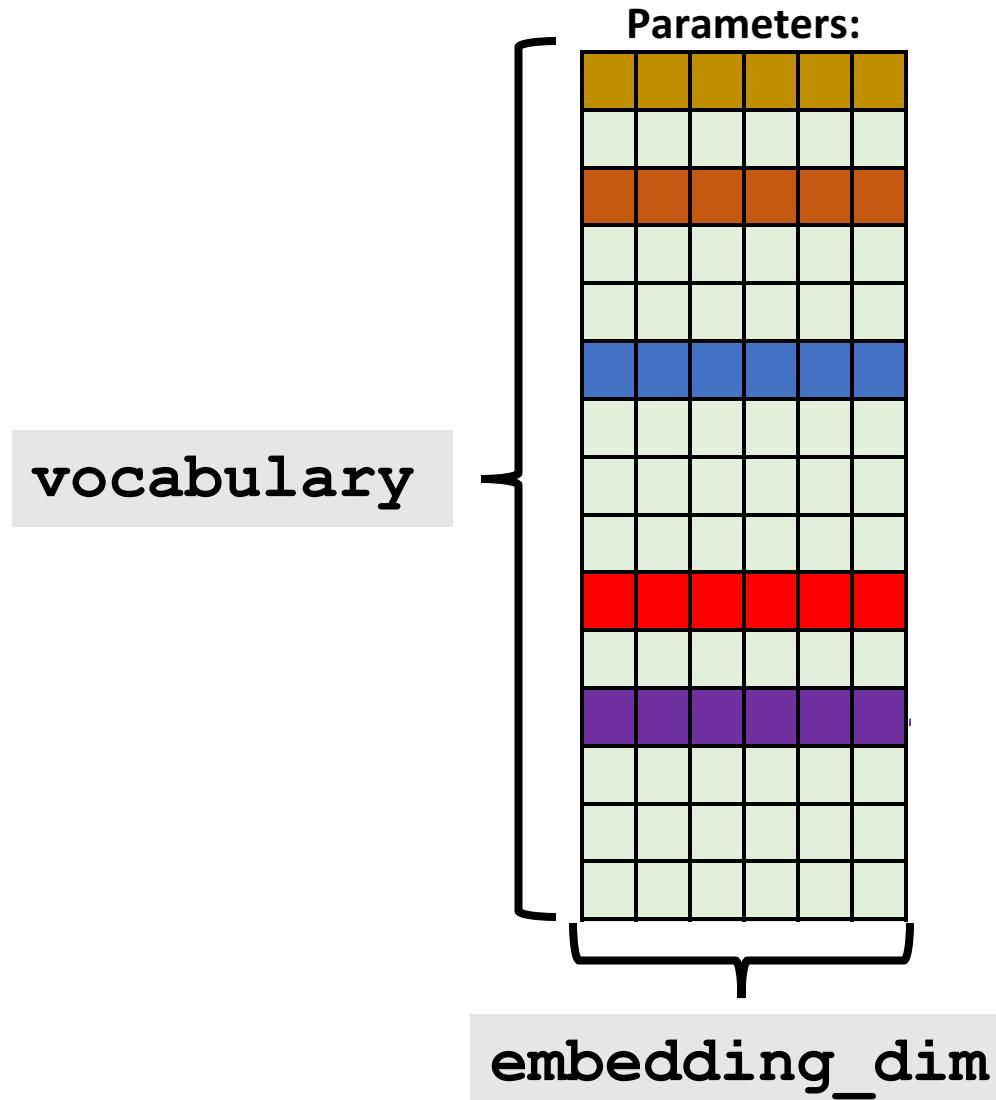
**Sequences:**  $50K \times \text{word\_num}$  integer matrix



**Embeddings:**  $50K \times \text{word\_num} \times \text{embedding\_dim}$  real tensor

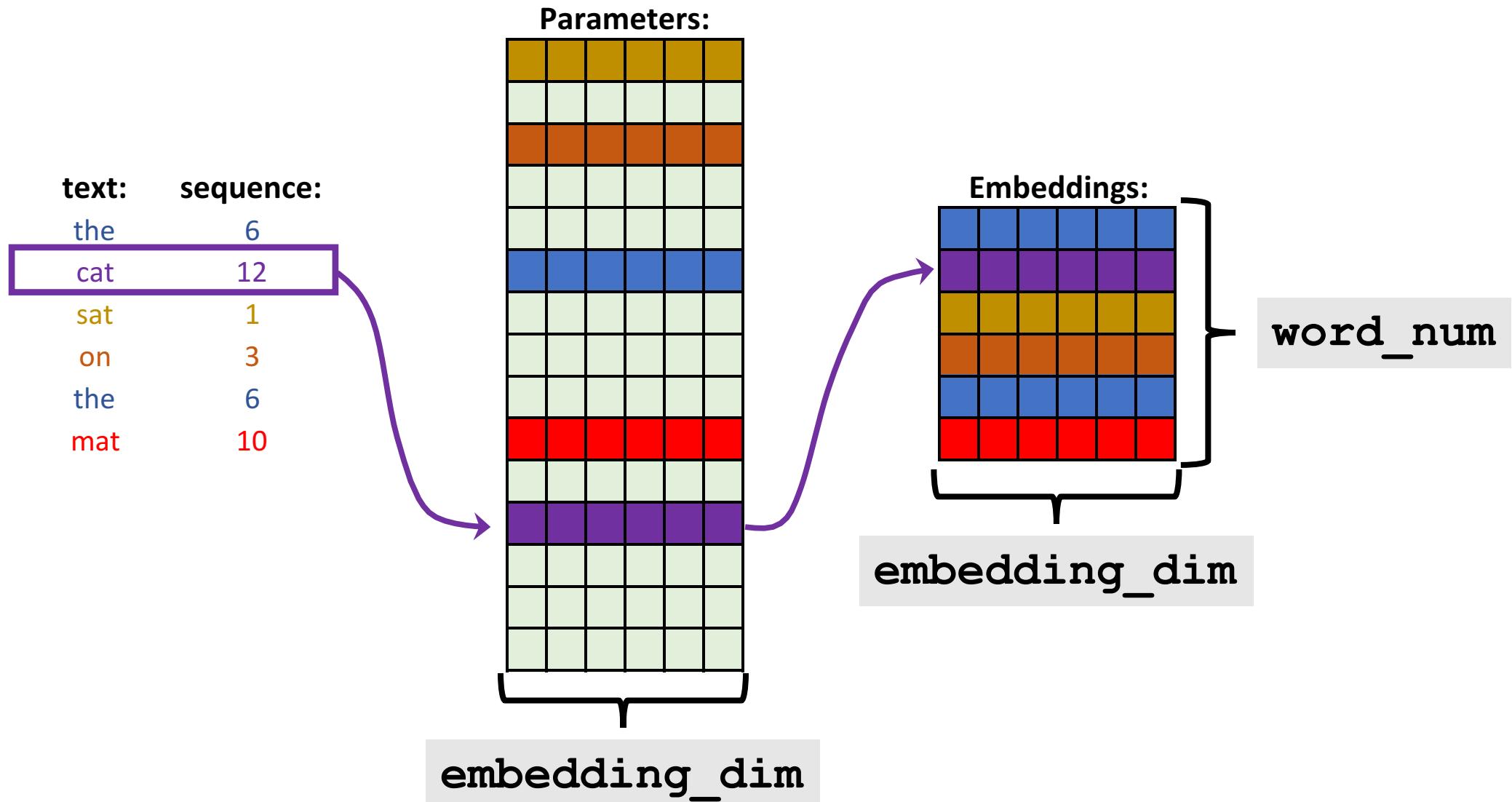
## Task: Word embedding: an index (of word) to a vector.

- Learn a parameter matrix of size `vocabulary`  $\times$  `embedding_dim`.



## Task: Word embedding: an index (of word) to a vector.

- Learn a parameter matrix of size `vocabulary` × `embedding_dim`.



# Task: Word embedding: an index (of word) to a vector.

```
from keras.models import Sequential  
from keras.layers import Flatten, Dense, Embedding
```

```
embedding_dim = 8
```

```
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
```

= 10K

= 8

= 20

Layer (type)	Output Shape	Param #	
embedding_1 (Embedding)	(None, 20, 8)	80000	= vocabulary × embedding_dim

word\_num      embedding\_dim

# Logistic Regression for Binary Classification

# Represent Text by Vectors/Matrices

**Goal:** Build a binary classifier for predicting the sentiment of a movie review.

**Task 1:** Convert the  $50K$  movie reviews (text) to  $50K$  sequences (vectors).

**Task 2:** Word embedding: an index (of word) to a vector.

**Task 3:** Binary classification using a neural network.

# Task: Word embedding: an index (of word) to a vector.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

embedding_dim = 8

model = Sequential()
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
embedding_1 (Embedding)	(None, 20, 8)	80000
<hr/>		
flatten_1 (Flatten)	(None, 160)	0
<hr/>		
dense_1 (Dense)	(None, 1)	161
<hr/>		

Total params: 80,161

Trainable params: 80,161

Non-trainable params: 0

## Task: Binary classification using a neural network.

```
from keras import optimizers  
  
epochs = 50  
  
model.compile(optimizer=optimizers.RMSprop(lr=0.0001),  
              loss='binary_crossentropy', metrics=['acc'])
```

## Task: Binary classification using a neural network.

```
from keras import optimizers  
  
epochs = 50  
  
model.compile(optimizer=optimizers.RMSprop(lr=0.0001),  
              loss='binary_crossentropy', metrics=['acc'])  
history = model.fit(x_train, y_train, epochs=epochs,  
                      batch_size=32, validation_data=(x_valid, y_valid))
```

- The training set is randomly split to a training set and a validation set.
- 80% for training and 20% for validation.

# Task: Binary classification using a neural network.

```
from keras import optimizers

epochs = 50

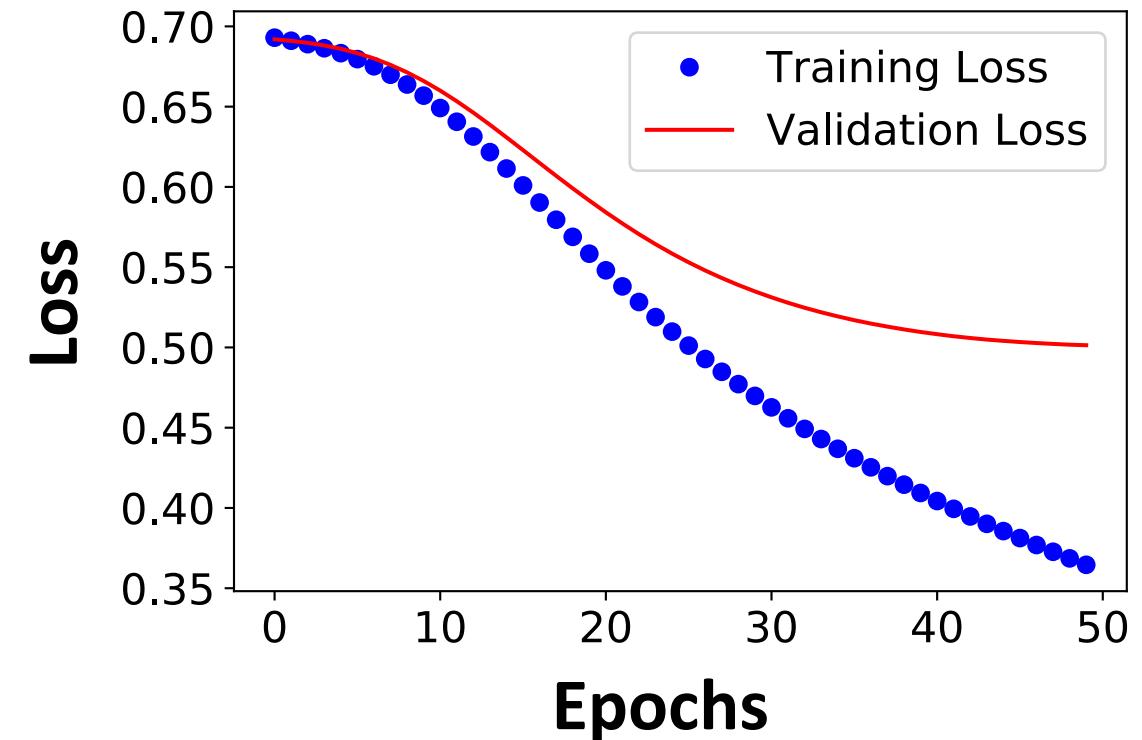
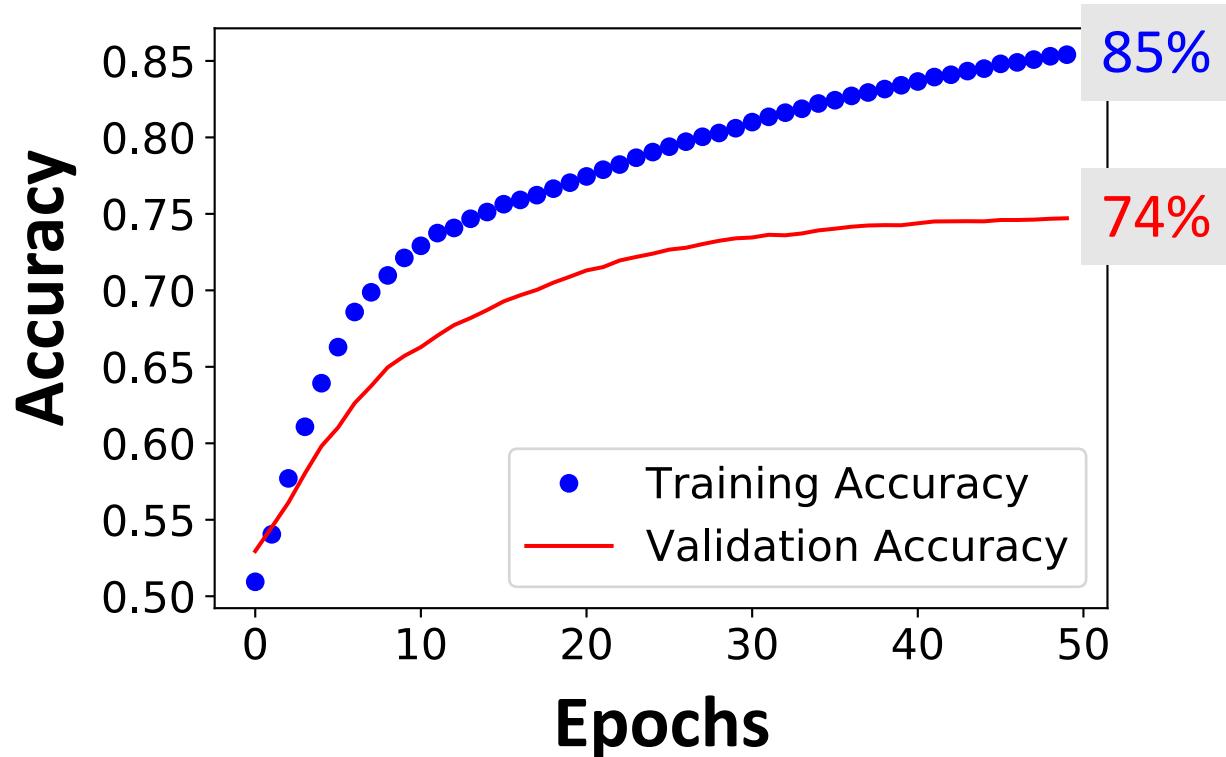
model.compile(optimizer=optimizers.RMSprop(lr=0.0001),
              loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=epochs,
                      batch_size=32, validation_data=(x_valid, y_valid))
```

```
Epoch 1/50
12500/12500 [=====] - 1s 74us/step - loss: 0.6930 - acc: 0.5094 - val_loss: 0.6919 - val_acc: 0.5295
Epoch 2/50
12500/12500 [=====] - 1s 60us/step - loss: 0.6911 - acc: 0.5405 - val_loss: 0.6910 - val_acc: 0.5452
Epoch 3/50
12500/12500 [=====] - 1s 57us/step - loss: 0.6889 - acc: 0.5770 - val_loss: 0.6898 - val_acc: 0.5612
.
.
.

●
```

```
Epoch 49/50
12500/12500 [=====] - 1s 56us/step - loss: 0.3686 - acc: 0.8530 - val_loss: 0.5017 - val_acc: 0.7468
Epoch 50/50
12500/12500 [=====] - 1s 56us/step - loss: 0.3646 - acc: 0.8541 - val_loss: 0.5014 - val_acc: 0.7471
```

# Task: Binary classification using a neural network.



## Task: Binary classification using a neural network.

Evaluate the model on the test set.

```
loss_and_acc = model.evaluate(x_test, labels_test)
print('loss = ' + str(loss_and_acc[0]))
print('acc = ' + str(loss_and_acc[1]))
```

```
25000/25000 [=====] - 0s 18us/step
loss = 0.5025235502243042
acc = 0.74928
```

- About 75% accuracy on the test set.
- Not bad, because we just use the last 20 words in each movie review. (`word_num=20`)

# Summary

# Texts to Sequences

texts[i] :

"For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt."



Tokenization

tokens[i] :

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a',
'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'imagine', 'a',
'movie', 'where', 'joe', 'piscopo', 'is', 'actually', 'funny', 'maureen',
'stapleton', 'is', 'a', 'scene', 'stealer', 'the', 'moroni', 'character', 'is',
'an', 'absolute', 'scream', 'watch', 'for', 'alan', 'the', 'skipper', 'hale', 'jr',
'as', 'a', 'police', 'sgt']
```



Encoding

seqs[i] :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2, 3, 19, 20,
21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33, 34, 35, 1, 36, 29,
37, 38, 39, 40, 2, 41, 42]
```

# Texts to Sequences

texts[i] :

“For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni character is an absolute scream. Watch for Alan "The Skipper" Hale jr. as a police Sgt.”



Tokenization

tokens[i] :

```
['for', 'a', 'movie', 'that', 'gets', 'no', 'respect', 'there', 'sure', 'are', 'a',
'lot', 'of', 'memorable', 'quotes', 'listed', 'for', 'this', 'gem', 'imagine', 'a',
'movie', 'where', 'joe', 'piscopo', 'is', 'actually', 'funny', 'maureen',
'stapleton', 'is', 'a', 'scene', 'stealer', 'the', 'moroni', 'character', 'is',
'an', 'absolute', 'scream', 'watch', 'for', 'alan', 'the', 'skipper', 'hale', 'jr',
'as', 'a', 'police', 'sgt']
```



Encoding

seqs[i] :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 13, 14, 15, 1, 16, 17, 18, 2, 3, 19, 20,
21, 22, 23, 24, 25, 26, 22, 2, 27, 28, 29, 30, 31, 22, 32, 33, 34, 35, 1, 36, 29,
37, 38, 39, 40, 2, 41, 42]
```

Alignment (keep the last  $w = 20$  tokens)

# Logistic Regression for Sentiment Analysis

seqs[i] :

```
[27, 28, 29, 30, 31, 22, 32, 33, 34, 35, 1, 36, 29, 37, 38, 39, 40, 2, 41, 42]
```



X[i] :

word\_num × embedding\_dim ( $20 \times 8$ ) matrix



x[i] :

160-dim vector



f[i] :

Binary Prediction (positive or negative)

# Logistic Regression for Sentiment Analysis

