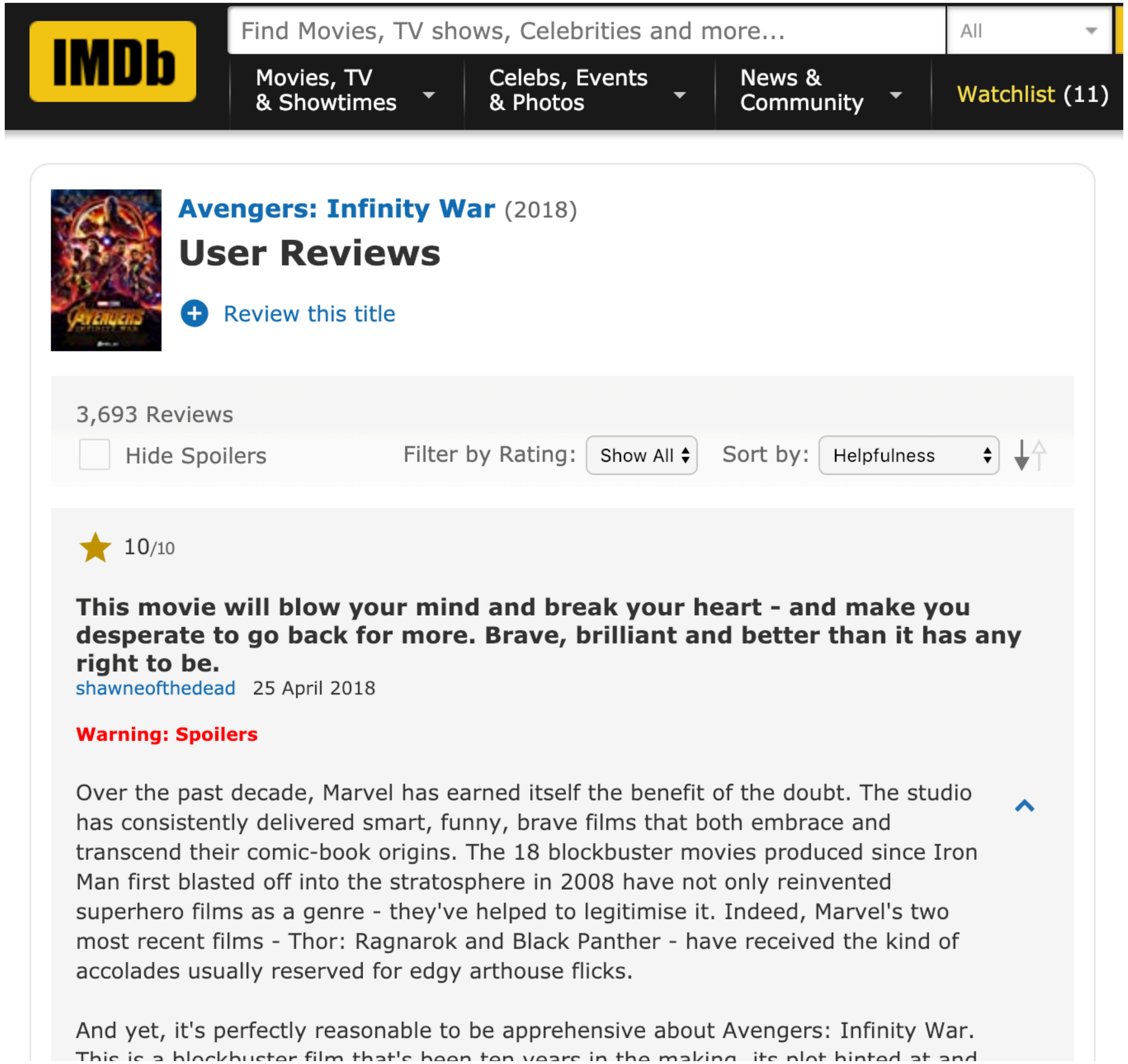
The IMDB Movie Review Dataset

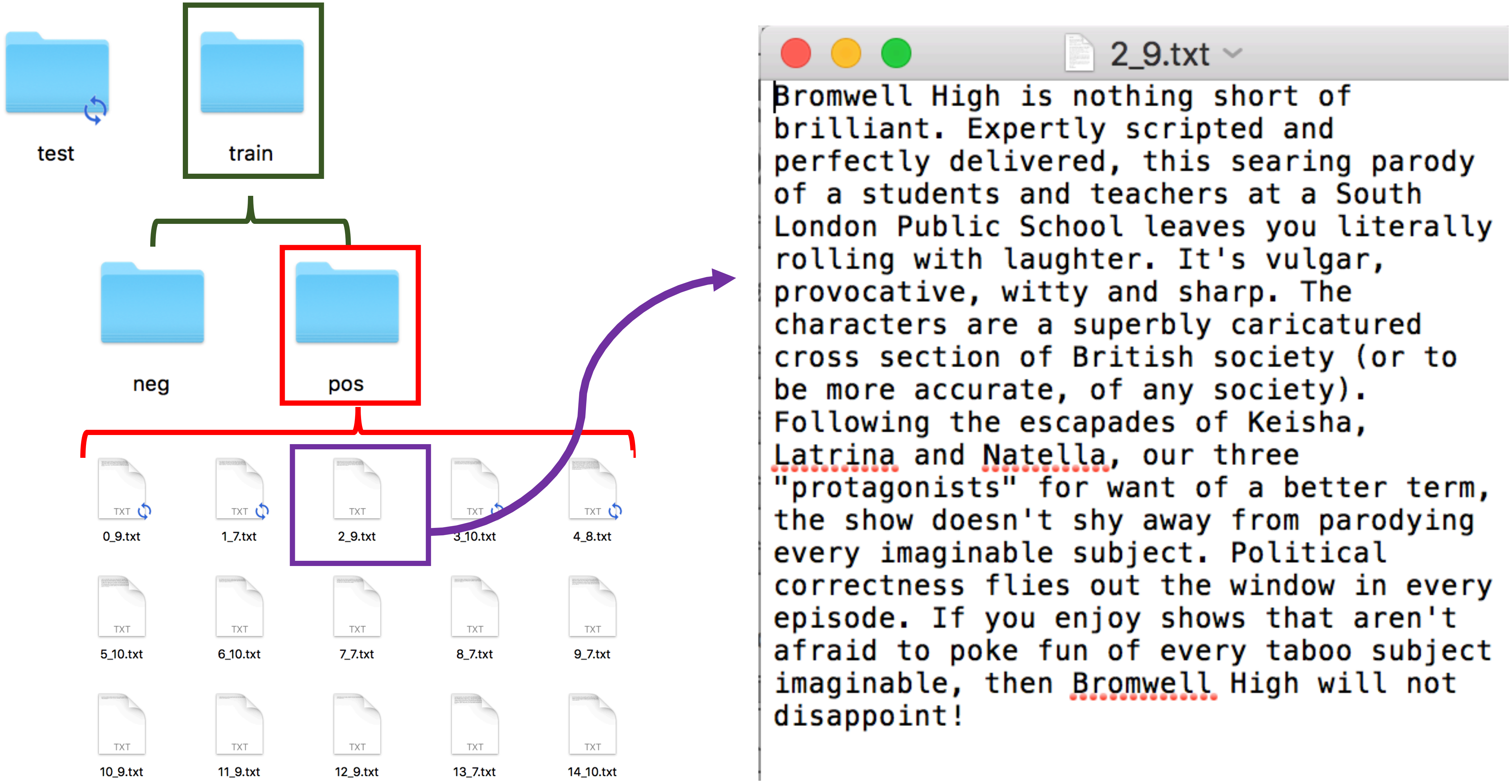


The IMDB Movie Review Dataset

- 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



Text to Sequence

Step 1: Tokenization

- Tokenization breaks a piece of text down into a list of tokens.
- Here, a token is a word. (A token can be a character in some applications.)

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

Tokenization

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

Step 1: Tokenization

- Tokenization breaks a piece of text down into a list of tokens.
- Here, a token is a word. (A token can be a character in some applications.)

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

↓ **Tokenization**

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

- Considerations in tokenization:
  - Upper case to lower case. ("Apple" to "apple"?)
  - Remove stop words, e.g., "the", "a", "of", etc.
  - Typo correction. ("goood" to "good".)

Step 2: Build Dictionary

- Use a dictionary (hash table) to count word frequencies.
- The dictionary maps word to index.

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

↓ **Tokenization**

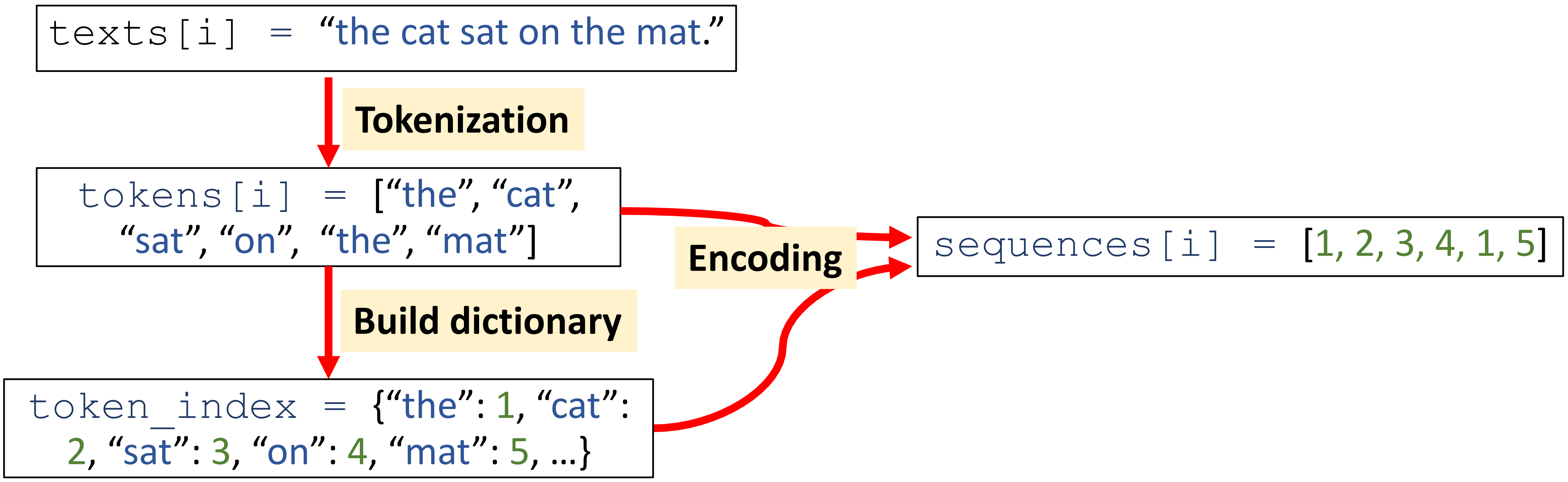
`tokens[i] =` [“the”, “cat”, “sat”, “on”, “the”, “mat”]

↓ **Build dictionary**

`token_index =` {“the”: 1, “cat”: 2, “sat”: 3, “on”: 4, “mat”: 5, …}

Step 3: One-Hot Encoding

- Use the dictionary to map words to indices (integers).
- A list of indices is called a sequence.



Step 3: One-Hot Encoding

`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. There 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]
```

Step 3: One-Hot Encoding

Result of encoding: $25K$ 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. There 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]`

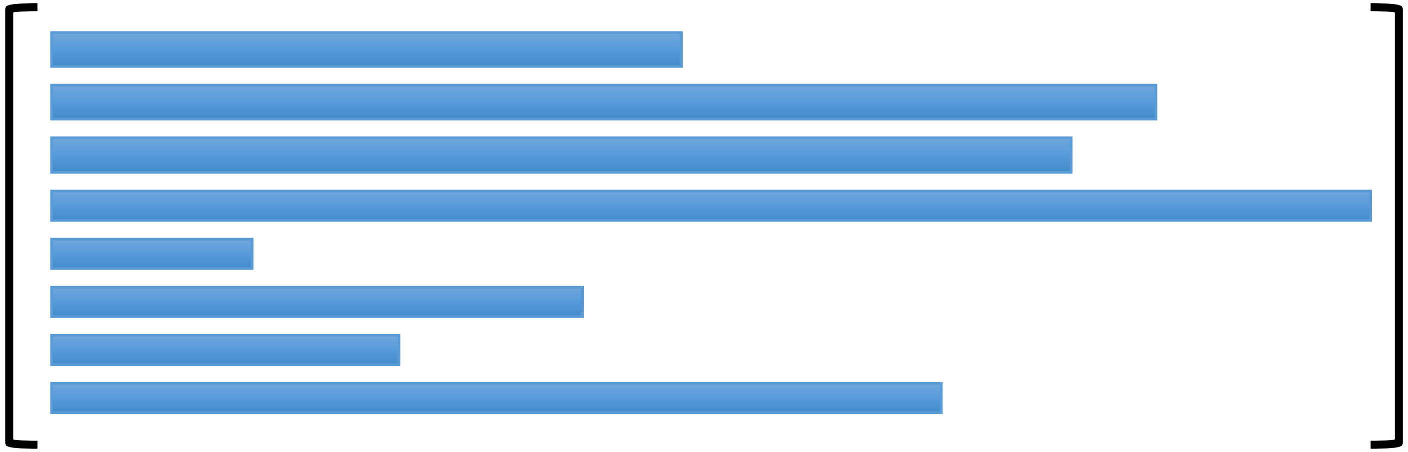
```
[178, 486, 29, 3, 46, 407, 487, 488, 489, 272,
160, 273, 40, 490, 40, 491, 178, 492, 272, 160,
493, 494, 193, 2, 3, 495, 496, 51, 488, 64,
385, 97, 497, 498, 8, 160, 273, 2, 499, 459,
178, 335, 29, 488, 293, 500, 272, 8, 196, 357,
501, 502, 29, 263, 110, 503, 263, 12, 504, 141,
391, 29, 505, 506, 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]
```

$w_5 = 90$ tokens

Step 4: Align Sequences

Result of encoding: $25K$ lists of integers; the i-th list has w_i elements.

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



Step 4: Align Sequences

Result of encoding: $25K$ lists of integers; the i-th list has w_i elements.

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

Solution:

- Cut off the text 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."

Step 4: Align Sequences

Result of encoding: $25K$ lists of integers; the i-th list has w_i elements.

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

Solution:

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

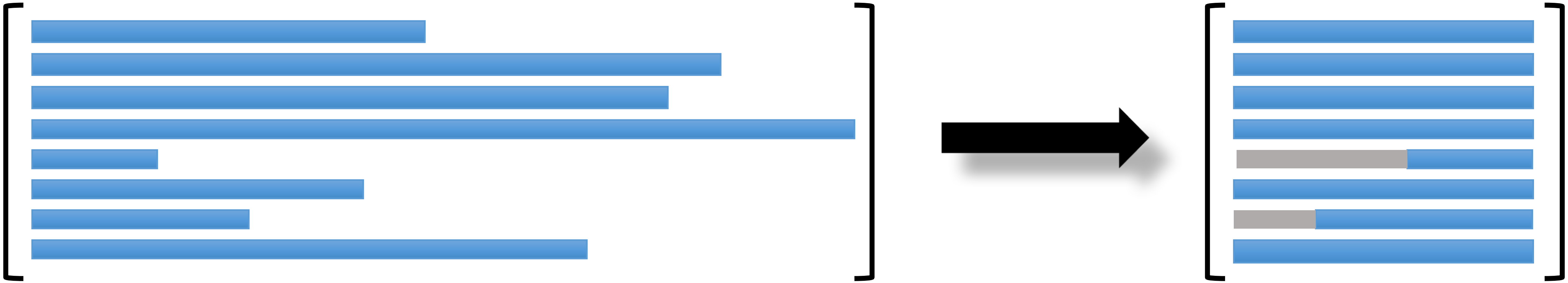
"the fat cat sat still on the big red mat." → "sat still on the big red mat."

"cat sat on the mat." → "null null cat sat on the mat."

Step 4: Align Sequences

Result of encoding: $25K$ lists of integers; the i-th list has w elements.

Aligned!



Text Processing in Keras

Steps 1, 2, & 3: Text to Sequence

```
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)
```

Steps 1, 2, & 3: Text to Sequence

```
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.

Steps 1, 2, & 3: Text to Sequence

```
from keras.preprocessing.text import Tokenizer

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

word_index = tokenizer.word_index
```

- word_index: dict[(string, int)]

Steps 1, 2, & 3: Text to Sequence

```
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)
```

- texts_train: list[string]
- sequence_train: list[list[int]]

Steps 1, 2, & 3: Text to Sequence

```
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]
```

Steps 4: Align Sequences

```
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)
```

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)

Do the same to test data

- Do the same operations to test data.
  - Step 1: tokenization.
  - Step 3: encoding.
  - Step 4: alignment.

Do the same to test data

- Do the same operations to test data.
  - Step 1: tokenization.
  - Step 3: encoding.
  - Step 4: alignment.
- Use the dictionary built on the training data.
  - Don't build a dictionary on the test data!
  - The dictionary for training and test must be the same!
- Otherwise, this may happen:
  - In training data, index 23 refers to "good".
  - In test data, index 23 refers to "mediocre".

Word Embedding: Word to Vector

How to map word to vector?

Word	Index	
“movie”	1	
“good”	2	
“fun”	3	
“boring”	4	
…	…	

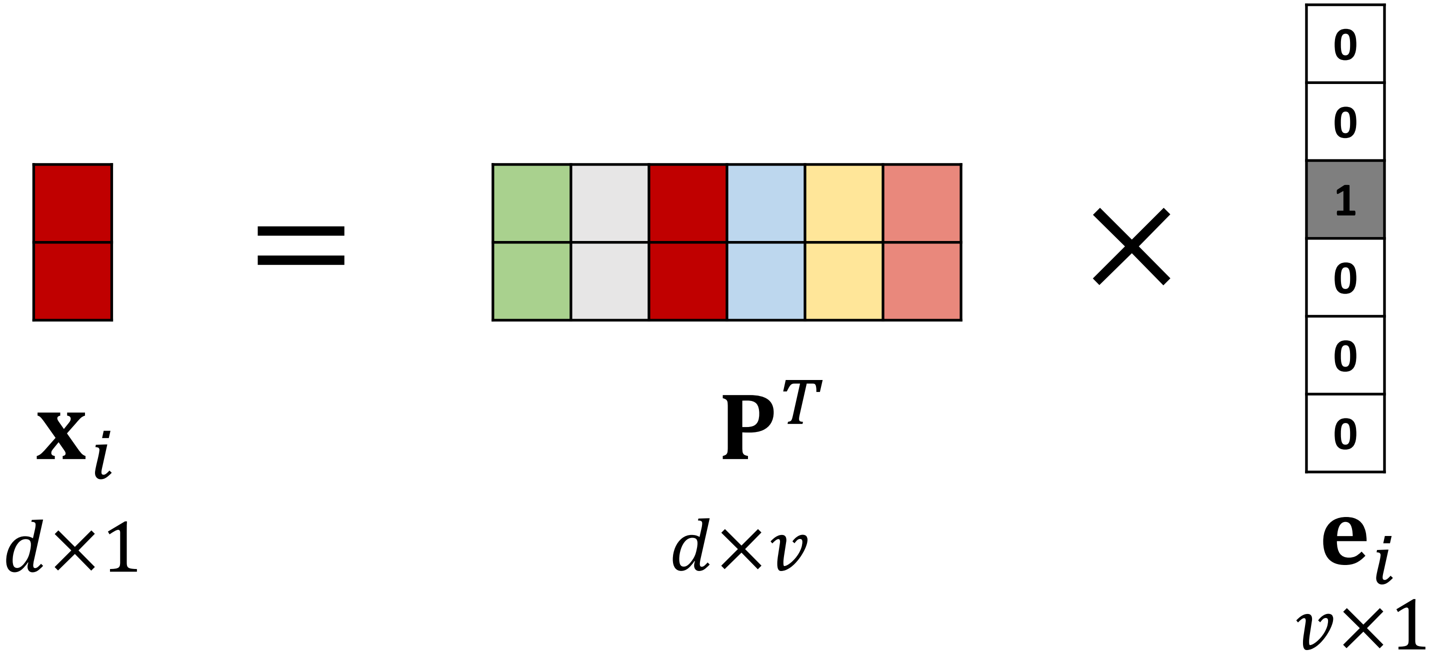
One-Hot Encoding

- First, represent words using one-hot vectors.
  - Suppose the dictionary contains v unique words (vocabulary= v).
  - Then the one-hot vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \cdots, \mathbf{e}_v$ are v-dimensional.

Word	Index	One-hot encoding
"movie"	1	$\mathbf{e}_1 = [1, 0, 0, 0, 0, \cdots, 0]$
"good"	2	$\mathbf{e}_2 = [0, 1, 0, 0, 0, \cdots, 0]$
"fun"	3	$\mathbf{e}_3 = [0, 0, 1, 0, 0, \cdots, 0]$
"boring"	4	$\mathbf{e}_4 = [0, 0, 0, 1, 0, \cdots, 0]$
…	…	…

Word Embedding

- Second, map the one-hot vectors to low-dimensional vectors by

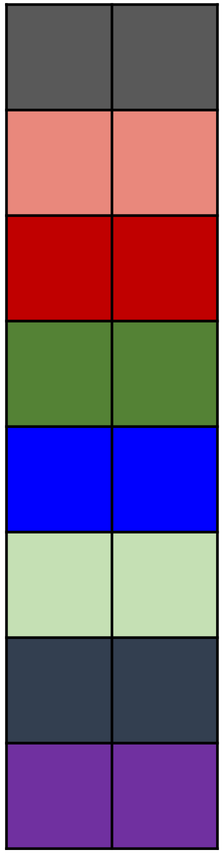


- $\mathbf{P}$ is parameter matrix which can be learned from training data.
- $\mathbf{e}_i$ is the one-hot vector of the i-th word in dictionary.

How to interpret the parameter matrix?

Parameter matrix

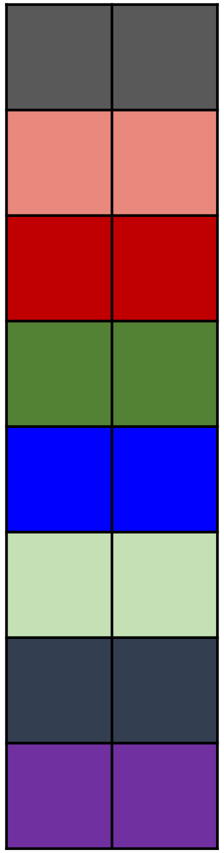
$$\mathbf{P} \in \mathbb{R}^{v \times d}$$



How to interpret the parameter matrix?

Parameter matrix

$$\mathbf{P} \in \mathbb{R}^{v \times d}$$



1: “movie”

2: “good”

3: “fun”

4: “boring”

5: “poor”

6: “mediocre”

7: “is”

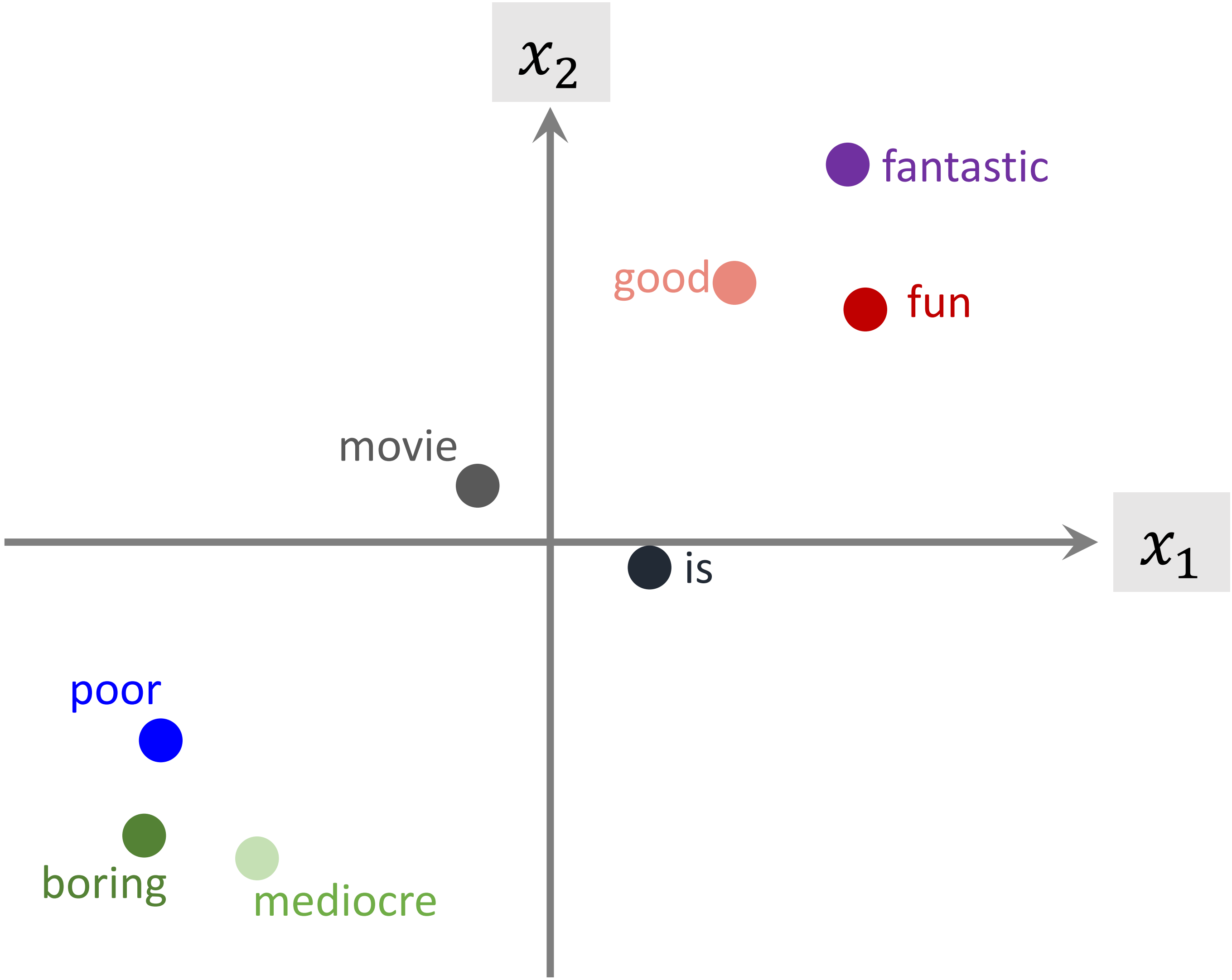
8: “fantastic”

How to interpret the parameter matrix?

Parameter matrix

$\mathbf{P} \in \mathbb{R}^{v \times d}$

		1: "movie"
		2: "good"
		3: "fun"
		4: "boring"
		5: "poor"
		6: "mediocre"
		7: "is"
		8: "fantastic"



```
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))
```

$v = 10K$ $\quad d = 8$ $\quad = 20$

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

word_num

embedding_dim

= vocabulary ×embedding_dim

Logistic Regression for Binary Classification

```python
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 #
=================================================================
embedding_1 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
```

```python
from keras import optimizers

epochs = 50

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

```python
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))
```

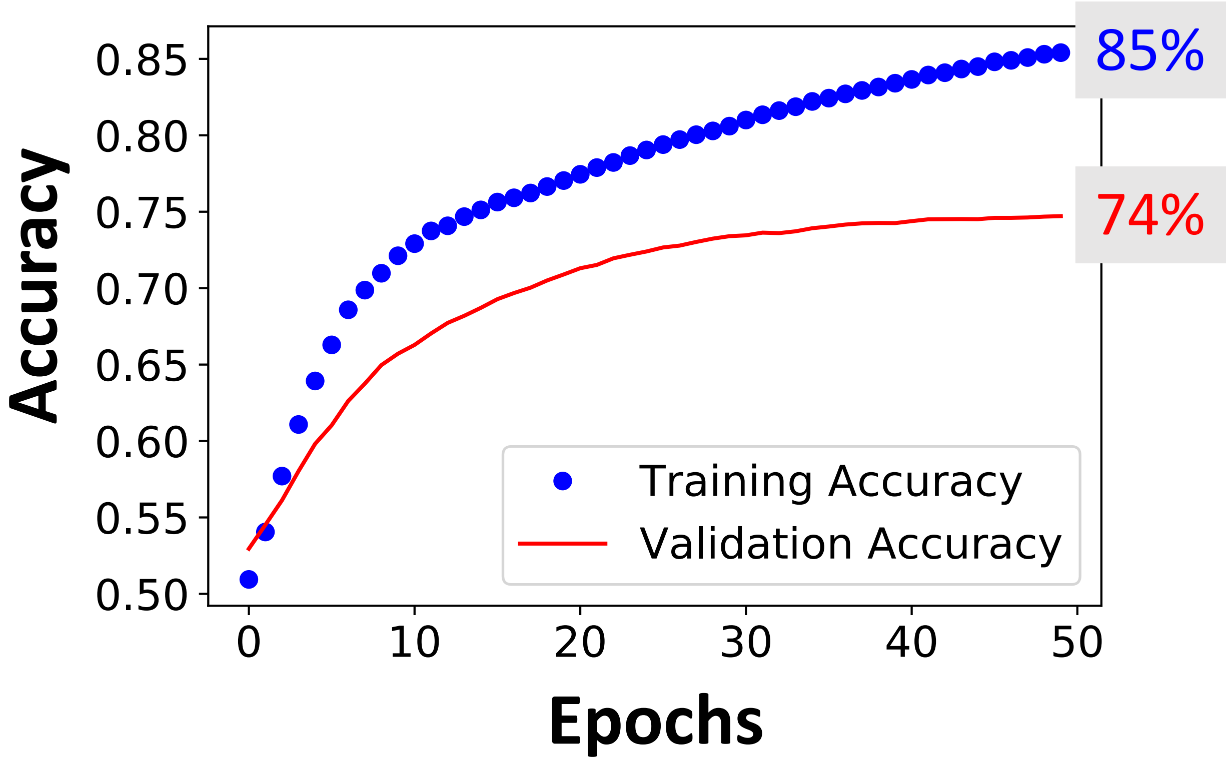
- The training set is randomly split to a training set and a validation set.
- 80% for training and 20% for validation.

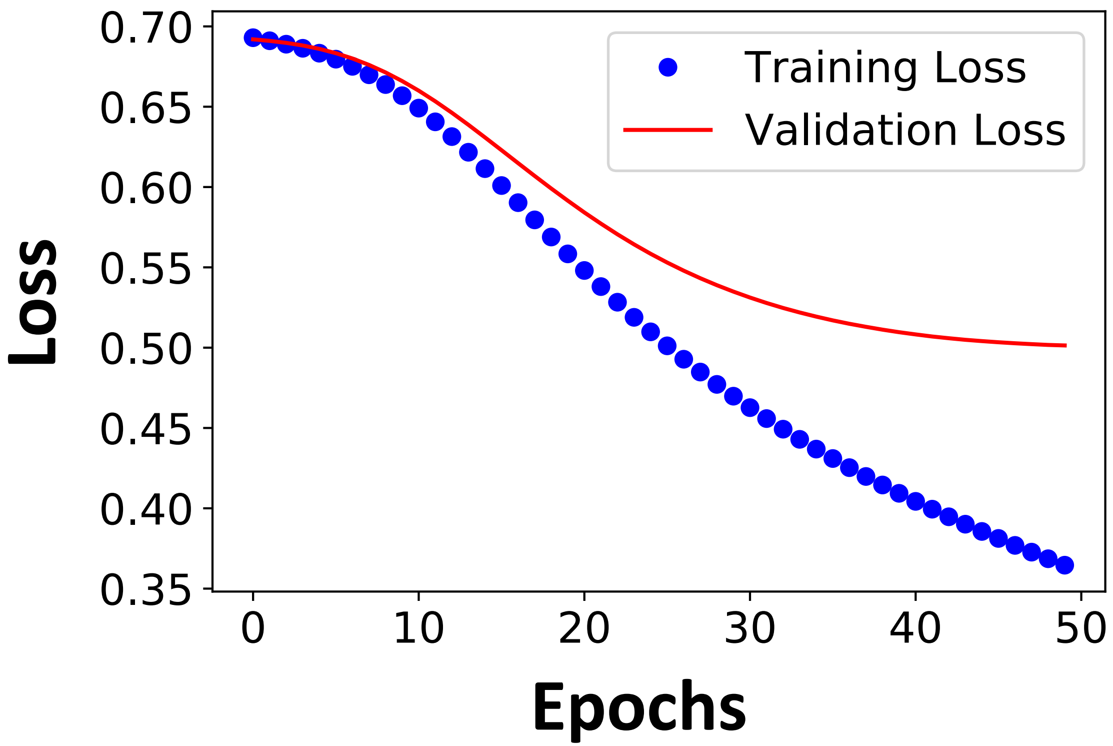
- x_train: 20,000×20 matrix
- X_valid: 5,000×20 matrix

```python
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
```

Performance on training and validation sets





Performance on 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 use only 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]`

↓ **Embedding Layer**

X[i]: word_num × embedding_dim (20×8) **matrix**

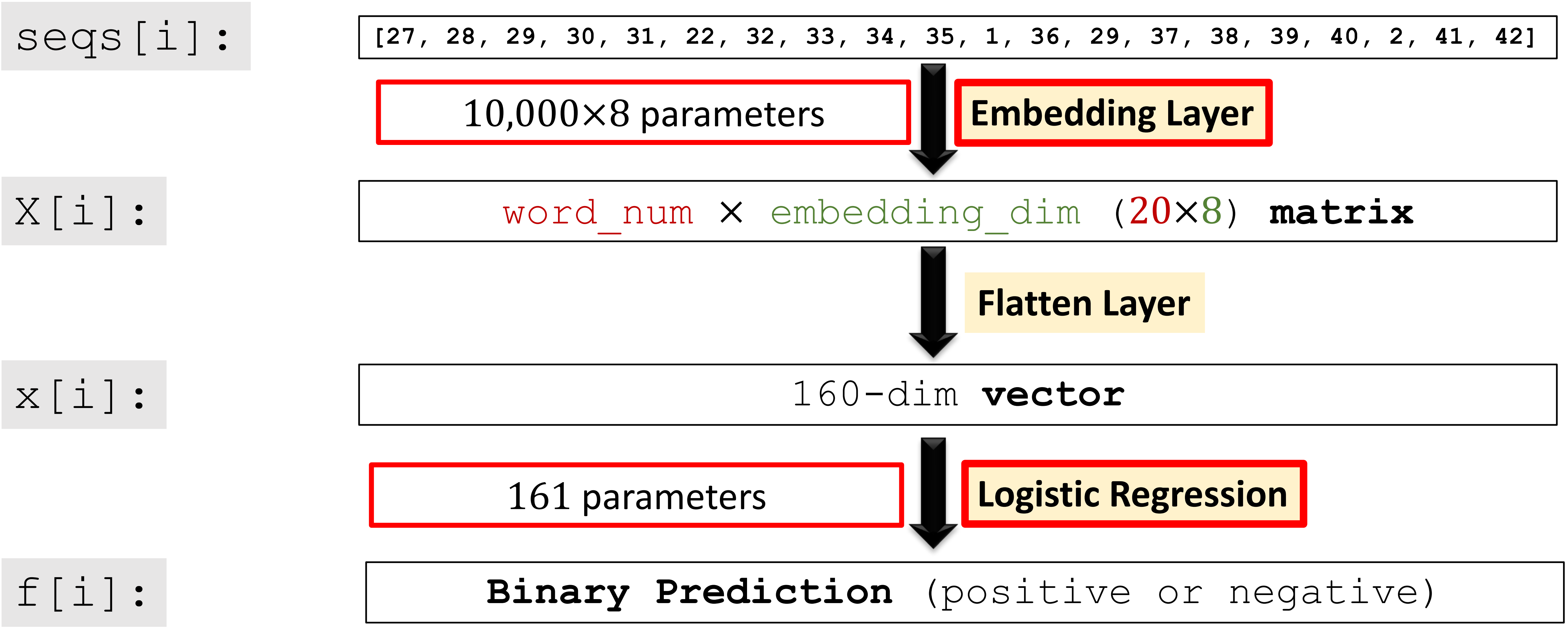
↓ **Flatten Layer**

x[i]: 160-dim **vector**

↓ **Logistic Regression**

f[i]: **Binary Prediction** (positive or negative)

Logistic Regression for Sentiment Analysis



Thank you!