IMDB Movie Review Sentiment Analysis

Import Statements

```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Is this notebook running on Colab or Kaggle?
IS_COLAB = "google.colab" in sys.modules
IS_KAGGLE = "kaggle_secrets" in sys.modules

if IS_COLAB:
    !pip install -q -U tensorflow-addons
    !pip install -q -U transformers

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# TensorFlow ≥2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected. LSTMs and CNNs can be very slow without a GPU.")
    if IS_COLAB:
        print("Go to Runtime > Change runtime and select a GPU hardware accelerator.")
    if IS_KAGGLE:
        print("Go to Settings > Accelerator and select GPU.")

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "nlp"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 611.8/611.8 kB 14.0 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
inflect 7.5.0 requires typeguard>=4.0.1, but you have typeguard 2.13.3 which is incompatible.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 119.0 MB/s eta 0:00:00
```

```
tf.random.set_seed(42) # Sets the global random seed for TensorFlow's random number generators
```

```
(X_train, y_train), (X_test, y_test) = keras.datasets.imdb.load_data() # Loads the IMDB movie reviews dataset from Keras into training and t
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
X_train[0][:10] # Inspect first 10 words (dataset is preprocessed, therefore each index is a unique word in the database)
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
```

```
word_index = keras.datasets.imdb.get_word_index() # Instantiate the imdb word to index dictionary
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
word_index
```


```
{'fawn': 34701,
 'tsukino': 52006,
 'nunnery': 52007,
 'sonja': 16816,
 'vani': 63951,
 'woods': 1408,
 'spiders': 16115,
 'hanging': 2345,
 'woody': 2289,
 'trawling': 52008,
 "hold's": 52009,
 'comically': 11307,
 'localized': 40830,
 'disobeying': 30568,
 "'royale": 52010,
 "harpo's": 40831,
 'canet': 52011,
 'aileen': 19313,
 'acurately': 52012,
 "diplomat's": 52013,
 'rickman': 25242,
 'arranged': 6746,
 'rumbustious': 52014,
 'familiarness': 52015,
 "spider'": 52016,
 'hahahah': 68804,
 "wood'": 52017,
 'transvestism': 40833,
 "hangin'": 34702,
 'bringing': 2338,
 'seamier': 40834,
 'wooded': 34703,
 'bravora': 52018,
 'grueling': 16817,
 'wooden': 1636,
 'wednesday': 16818,
 "'prix": 52019,
 'altagracia': 34704,
 'circuitry': 52020,
 'crotch': 11585,
 'busybody': 57766,
 "tart'n'tangy": 52021,
 'burgade': 14129,
 'thrace': 52023,
 "tom's": 11038,
 'snuggles': 52025,
 'francesco': 29114,
 'complainers': 52027,
 'templarios': 52125,
 '272': 40835,
 '273': 52028,
 'zaniacs': 52130,
 '275': 34706,
 'consenting': 27631,
 'snuggled': 40836,
 'inanimate': 15492,
 'uality': 52030,
 'bronte': 11926,
```


```
# Create an ID-to-word dictionary to map token indices back to words
# Offset IDs by +3 to reserve indices 0-2 for special tokens (<pad>, <sos>, <unk>)
id_to_word = {id_+3:word for word, id_ in word_index.items()}

# Add the special tokens manually
# 0 → <pad> (used to pad shorter sequences)
# 1 → <sos> (start of sequence)
```

```
# 2 → <unk> (unknown or rare word)
for id_, token in enumerate(("<pad>", "<sos>", "<unk>" )):
    id_to_word[id_] = token

# Use the mapping to join the words (seperated by a space) in our 10 word sample
" ".join([id_to_word[id_] for id_ in X_train[0][:10]] )
```

```
'<sos> this film was just brilliant casting location scenery story'
```

```
word_index["noodle"] # Find the index of a specific word
```

```
14351
```

```
id_to_word[14354] # Remember the +3 to find the corresponding word
```

```
'noodle'
```

```
# Display the first 10 token IDs and their corresponding words, if available
[(i, id_to_word[i]) for i in range(10) if i in id_to_word] # just make sure all words are mapped to a unique number
```

```
[(0, '<pad>'),
 (1, '<sos>'),
 (2, '<unk>'),
 (4, 'the'),
 (5, 'and'),
 (6, 'a'),
 (7, 'of'),
 (8, 'to'),
 (9, 'is')]
```

```
pip install tensorflow_datasets
```

```
Requirement already satisfied: tensorflow_datasets in /usr/local/lib/python3.11/dist-packages (4.9.9)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (1.4.0)
Requirement already satisfied: array_record>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.7.2)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.1.9)
Requirement already satisfied: etils>=1.9.1 in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; pyth
Requirement already satisfied: immutabledict in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (4.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (2.0.2)
Requirement already satisfied: promise in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (2.3)
Requirement already satisfied: protobuf>=3.20 in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (5.29.5)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (5.9.5)
Requirement already satisfied: pyarrow in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (18.1.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (2.32.3)
Requirement already satisfied: simple_parsing in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.1.7)
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (1.17.2)
Requirement already satisfied: termcolor in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (3.1.0)
Requirement already satisfied: toml in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.10.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (4.67.1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (1.17.2)
Requirement already satisfied: einops in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_ver
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_ver
Requirement already satisfied: importlib_resources in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1;
Requirement already satisfied: zipp in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_versi
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_da
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_datasets) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_datasets
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_datasets
Requirement already satisfied: attrs>=18.2.0 in /usr/local/lib/python3.11/dist-packages (from dm-tree->tensorflow_datasets) (25.3.0)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from promise->tensorflow_datasets) (1.17.0)
Requirement already satisfied: docstring-parser<1.0,>=0.15 in /usr/local/lib/python3.11/dist-packages (from simple_parsing->tensorflow_d
Requirement already satisfied: googleapis-common-protos<2,>=1.56.4 in /usr/local/lib/python3.11/dist-packages (from tensorflow-metadata-
```

```
import tensorflow_datasets as tfds
datasets, info = tfds.load(
    'imdb_reviews', # specifices the dataset to load
    as_supervised=True, # returns labeled data in format (text, label) for supervised learning
    with_info=True) # returns: datasets (dict with train, test, and unsupervised splits), info (metadata about dataset)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tensorflc
```

Dl Completed...: 100% 1/1 [00:03<00:00, 3.43s/ url]

Dl Size...: 100% 80/80 [00:03<00:00, 25.03 MiB/s]

```
datasets
```

```
{Split('train'): <_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>,
 Split('test'): <_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>,
 Split('unsupervised'): <_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>}
```

```
datasets.keys()
```

```
dict_keys([Split('train'), Split('test'), Split('unsupervised')])
```

```
info
```

```
tfds.core.DatasetInfo(
    name='imdb_reviews',
    full_name='imdb_reviews/plain_text/1.0.0',
    description="""
    Large Movie Review Dataset. This is a dataset for binary sentiment
    classification containing substantially more data than previous benchmark
    datasets. We provide a set of 25,000 highly polar movie reviews for training,
    and 25,000 for testing. There is additional unlabeled data for use as well.
    """,
    config_description="""
    Plain text
    """,
    homepage='http://ai.stanford.edu/~amaas/data/sentiment/',
    data_dir='/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0',
    file_format=tfrecord,
    download_size=80.23 MiB,
    dataset_size=129.83 MiB,
    features=FeaturesDict({
        'label': ClassLabel(shape=(), dtype=int64, num_classes=2),
        'text': Text(shape=(), dtype=string),
    }),
    supervised_keys=('text', 'label'),
    disable_shuffling=False,
    nondeterministic_order=False,
    splits={
        'test': <SplitInfo num_examples=25000, num_shards=1>,
        'train': <SplitInfo num_examples=25000, num_shards=1>,
        'unsupervised': <SplitInfo num_examples=50000, num_shards=1>,
    },
    citation="""@InProceedings{maas-EtAl:2011:ACL-HLT2011,
      author    = {Maas, Andrew L.  and  Daly, Raymond E.  and  Pham, Peter T.  and  Huang, Dan  and  Ng, Andrew Y.  and  Potts, Christopher},
      title     = {Learning Word Vectors for Sentiment Analysis},
      booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies},
      month     = {June},
      year      = {2011},
      address   = {Portland, Oregon, USA},
      publisher = {Association for Computational Linguistics},
      pages     = {142--150},
      url       = {http://www.aclweb.org/anthology/P11-1015}
    }""",
)
```

```
train_size = info.splits["train"].num_examples # extract the number of training examples from the IMDB dataset
```

```
train_size
```

```
25000
```

```
test_size = info.splits['test'].num_examples # extract the number of test examples from the IMDB dataset
```

```
test_size
```

```
25000
```

```
# Group the dataset into batches of 2 reviews (batch size = 2)
# Take only the *first batch* (so we don’t loop through the whole dataset)
for X_batch, y_batch in datasets['train'].batch(2).take(1):

    # Print a tensor containing 2 raw text reviews.
    # Each review is stored as a byte string (notice the `b""` prefix).
    # A byte string means the data is encoded in bytes — it's not yet a Python string (str).
    # In NLP, this is common for storage and performance.
    # Even though it *looks* like readable text, it’s not decoded yet.
    print(X_batch)
    print('-------------------')

    # Convert the Tensor to a NumPy array.
    # This step allows us to manipulate the data more easily using standard Python and NumPy functions.
    # Output will look like: [b'Review1...', b'Review2...']
    print(X_batch.numpy())
    print('-------------------')

    # Decode the first review (still in byte form) into a proper UTF-8 string.
    # This step converts it into a regular Python string so we can read and process it.
    # UTF-8 is the standard encoding for human-readable text.
    print(X_batch.numpy()[0].decode('utf-8'))
    print('-------------------')

    # Print the labels tensor for the two reviews.
    # These are sentiment labels: 0 = Negative, 1 = Positive.
    # Example output: tf.Tensor([0 0], shape=(2,), dtype=int64)
    # So in this case, both reviews are classified as Negative.
    print(y_batch)
```

```
tf.Tensor(
[b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this m
 b'I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm
-------------------
[b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this m
 b'I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm
-------------------
This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must
-------------------
tf.Tensor([0 0], shape=(2,), dtype=int64)
```

```
# Group the dataset into batches of 2 reviews (batch size = 2)
# Take only the *first batch* (so we don’t loop through the whole dataset)
for X_batch, y_batch in datasets['train'].batch(2).take(1):

    # Convert both the review and label tensors into NumPy arrays
    # Then iterate through them in parallel using zip()
    # Each `review` is still in byte string format and must be decoded
    for review, label in zip(X_batch.numpy(), y_batch.numpy()):

        # Decode the byte string review into readable UTF-8 text
        # Truncate to the first 200 characters for cleaner display
        print("review: ", review.decode('utf-8')[:200], "...")

        # Print the label (0 or 1), and translate it to a human-readable sentiment
        # 0 = Negative, 1 = Positive
        print("label: ", label, '= Positive' if label else '= Negative')

        # Add a blank line between samples for visual separation
        print()
```

```
review:  This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but
label:  0 = Negative

review:  I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, bein
label:  0 = Negative
```

Preprocessing

- It starts by truncating the reviews, keeping only the first 300 characters of each: this will speed up training, and it won't impact performance too much because you can generally tell whether a review is positive or not in the first sentence or two.
- Then it uses regular expressions to replace `<br />` tags with spaces, and
- replace any characters other than letters and quotes with spaces. For example, the text "Well, I can't `<br />`" will become "Well I can't ".
- Finally, the preprocess() function splits the reviews by the spaces, which returns a ragged tensor, and it converts this ragged tensor to a dense tensor, padding all reviews with the padding token "" so that they all have the same length.

Example

```
# replace all <br />, <br>, or <br/> HTML tags with a single space
temp = tf.strings.regex_replace(["Well, I can't<br />", "Hi, there!"], rb"<br\s*/?>", b" ")

# Print the result — Tensor of byte strings with <br> replaced by spaces
print(temp)
```

```
tf.Tensor([b"Well, I can't " b'Hi, there!'], shape=(2,), dtype=string)
```

```
# replace anything that is not a letter (ex: punctuation, numbers, etc...)
temp = tf.strings.regex_replace(temp, b"[^a-zA-Z']", b" ")

# Print the cleaned output — only letters and apostrophes should remain
print(temp)
```

```
tf.Tensor([b"Well  I can't " b'Hi  there '], shape=(2,), dtype=string)
```

```
# split each tensor byte string into words
temp = tf.strings.split(temp)
print(temp)
```

```
<tf.RaggedTensor [[b'Well', b'I', b"can't"], [b'Hi', b'there']]>
```

```
# Add a <pad> to ensure that our tensor shape is consistent (not a RaggedTensor)
temp = temp.to_tensor(default_value = b"<pad>")


print(temp)
```

```
tf.Tensor(
[[b'Well' b'I' b"can't"]
 [b'Hi' b'there' b'<pad>']], shape=(2, 3), dtype=string)
```

Preprocessing

```
def preprocess(X_batch, y_batch):
    # Truncate each review to the first 300 characters -> better for training and can still capture sentiment well
    X_batch = tf.strings.substr(X_batch, 0, 300)
    print(type(X_batch))
    print(X_batch)
    print()

    # Replace HTML <br>, <br/>, and <br /> tags with a space
    X_batch = tf.strings.regex_replace(X_batch, rb"<br\s*/?>", b" ")
    print(type(X_batch))
    print(X_batch)
    print()

    # Remove all non-letter characters (except apostrophes)
    X_batch = tf.strings.regex_replace(X_batch, b"[^a-zA-Z']", b" ")
    print(type(X_batch))
    print(X_batch)
```

```
    print()

    # Split each review into tokens (words) based on whitespace
    # Returns a RaggedTensor — sequences of varying word counts
    X_batch = tf.strings.split(X_batch)
    print(type(X_batch))
    print(X_batch)
    print()

    return X_batch.to_tensor(default_value=b"<pad>"), y_batch # add padding to turn RaggedTensor back into Tensor of consistent shape


preprocess(X_batch, y_batch)
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
tf.Tensor(
[b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this m
 b'I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm

<class 'tensorflow.python.framework.ops.EagerTensor'>
tf.Tensor(
[b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this m
 b'I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm

<class 'tensorflow.python.framework.ops.EagerTensor'>
tf.Tensor(
[b"This was an absolutely terrible movie  Don't be lured in by Christopher Walken or Michael Ironside  Both are great actors  but this m
 b'I have been known to fall asleep during films  but this is usually due to a combination of things including  really tired  being warm

<class 'tensorflow.python.ops.ragged.ragged_tensor.RaggedTensor'>
<tf.RaggedTensor [[b'This', b'was', b'an', b'absolutely', b'terrible', b'movie', b"Don't",
  b'be', b'lured', b'in', b'by', b'Christopher', b'Walken', b'or',
  b'Michael', b'Ironside', b'Both', b'are', b'great', b'actors', b'but',
  b'this', b'must', b'simply', b'be', b'their', b'worst', b'role', b'in',
  b'history', b'Even', b'their', b'great', b'acting', b'could', b'not',
  b'redeem', b'this', b"movie's", b'ridiculous', b'storyline', b'This',
  b'movie', b'is', b'an', b'early', b'nineties', b'US', b'propaganda',
  b'pi']                                                                  ,
 [b'I', b'have', b'been', b'known', b'to', b'fall', b'asleep', b'during',
  b'films', b'but', b'this', b'is', b'usually', b'due', b'to', b'a',
  b'combination', b'of', b'things', b'including', b'really', b'tired',
  b'being', b'warm', b'and', b'comfortable', b'on', b'the', b'sette',
  b'and', b'having', b'just', b'eaten', b'a', b'lot', b'However', b'on',
  b'this', b'occasion', b'I', b'fell', b'asleep', b'because', b'the',
  b'film', b'was', b'rubbish', b'The', b'plot', b'development', b'was',
  b'constant', b'Cons']                                                   ]>

(<tf.Tensor: shape=(2, 53), dtype=string, numpy=
 array([[b'This', b'was', b'an', b'absolutely', b'terrible', b'movie',
         b"Don't", b'be', b'lured', b'in', b'by', b'Christopher',
         b'Walken', b'or', b'Michael', b'Ironside', b'Both', b'are',
         b'great', b'actors', b'but', b'this', b'must', b'simply', b'be',
         b'their', b'worst', b'role', b'in', b'history', b'Even',
         b'their', b'great', b'acting', b'could', b'not', b'redeem',
         b'this', b"movie's", b'ridiculous', b'storyline', b'This',
         b'movie', b'is', b'an', b'early', b'nineties', b'US',
         b'propaganda', b'pi', b'<pad>', b'<pad>', b'<pad>'],
        [b'I', b'have', b'been', b'known', b'to', b'fall', b'asleep',
         b'during', b'films', b'but', b'this', b'is', b'usually', b'due',
         b'to', b'a', b'combination', b'of', b'things', b'including',
         b'really', b'tired', b'being', b'warm', b'and', b'comfortable',
         b'on', b'the', b'sette', b'and', b'having', b'just', b'eaten',
         b'a', b'lot', b'However', b'on', b'this', b'occasion', b'I',
         b'fell', b'asleep', b'because', b'the', b'film', b'was',
         b'rubbish', b'The', b'plot', b'development', b'was', b'constant',
         b'Cons']], dtype=object)>,
 <tf.Tensor: shape=(2,), dtype=int64, numpy=array([0, 0])>)
```

Construct the vocabulary

```
from collections import Counter

vocabulary = Counter() # Initialize a Counter to keep track of word frequencies

# Iterate through the training dataset in batches of 32
# Each batch is preprocessed using the preprocess() function defined earlier
for X_batch, y_batch in datasets['train'].batch(32).map(preprocess):

    # For each review (sequence of tokens) in the batch
    for review in X_batch:
```

```
    # Convert the tensor of byte-string tokens to a list of bytes (e.g., [b'word1', b'word2', ...])
    # Update the vocabulary counter with token occurrences
    vocabulary.update(list(review.numpy()))
```

```
<class 'tensorflow.python.framework.ops.SymbolicTensor'>
Tensor("Substr:0", shape=(None,), dtype=string)

<class 'tensorflow.python.framework.ops.SymbolicTensor'>
Tensor("StaticRegexReplace:0", shape=(None,), dtype=string)

<class 'tensorflow.python.framework.ops.SymbolicTensor'>
Tensor("StaticRegexReplace_1:0", shape=(None,), dtype=string)

<class 'tensorflow.python.ops.ragged.ragged_tensor.RaggedTensor'>
tf.RaggedTensor(values=Tensor("StringSplit/StringSplitV2:1", shape=(None,), dtype=string), row_splits=Tensor("StringSplit/RaggedFromValu
```

```
# Get the 10 most common tokens in the vocabulary
vocabulary.most_common()[:10]
```

```
[(b'<pad>', 214309),
 (b'the', 61137),
 (b'a', 38564),
 (b'of', 33983),
 (b'and', 33431),
 (b'to', 27707),
 (b'I', 27019),
 (b'is', 25719),
 (b'in', 18966),
 (b'this', 18490)]
```

```
# Get number of tokens... Do we really need to know all of these words?
len(vocabulary)
```

```
53893
```

```
vocab_size = 10000 # We can use 10,000 tokens and still get decent results (good enough)
truncated_vocabulary = [word for word, count in vocabulary.most_common()[:vocab_size]] # keep the 10,000 most common tokens
```

```
len(truncated_vocabulary)
```

```
10000
```

```
truncated_vocabulary # most common tokens
```

```
[b'<pad>',
 b'the',
 b'a',
 b'of',
 b'and',
 b'to',
 b'I',
 b'is',
 b'in',
 b'this',
 b'it',
 b'was',
 b'movie',
 b'that',
 b'The',
 b'film',
 b'with',
 b'for',
 b'as',
 b'on',
 b'but',
 b'have',
 b'This',
 b'one',
 b'not',
 b'be',
 b'are',
 b'you',
 b'an',
 b'at',
 b'about',
 b'by',
```

```
      b'all',
      b'his',
      b'so',
      b'like',
      b'from',
      b'who',
      b'has',
      b'It',
      b'good',
      b'my',
      b'just',
      b'very',
      b'out',
      b'or',
      b'story',
      b'some',
      b'time',
      b'had',
      b'he',
      b'they',
      b'really',
      b'me',
      b'when',
      b'what',
      b'first',
      b'movies',
```

```
word_to_id = {word: index for index, word in enumerate(truncated_vocabulary)} # map indices to words (making a dictionary)
```

```
# Example: given a sentence, show its indices... words with index 10000 are == vocab size, meaning the word is not common and unknown (oov =
for word in b"This movie was faaaaaaaantastic akljfglkajglk".split():
    print(word_to_id.get(word) or vocab_size)
```

```
22
12
11
10000
10000
```

oov: out of vocab.

We can't just use the same index for these words with very different meaning.

```
# Reserve 1000 unique IDs for out-of-vocabulary (OOV) words
num_oov_buckets = 1000

# Convert the truncated vocabulary (list of top 10,000 byte-string tokens) into a constant Tensor
words = tf.constant(truncated_vocabulary)

# Create a range of integer IDs [0, 1, 2, ..., 9999] matching the vocabulary size
word_ids = tf.range(len(truncated_vocabulary), dtype=tf.int64)

# Create a key-value initializer that maps each word to its corresponding ID
vocab_init = tf.lookup.KeyValueTensorInitializer(words, word_ids)

# Build a static lookup table with OOV handling:
# - Known words are mapped to their ID from the vocabulary
# - Unknown words are mapped to one of the 1000 OOV buckets using a hash function
table = tf.lookup.StaticVocabularyTable(vocab_init, num_oov_buckets)
```

```
# Look up the word IDs for a sample list of tokens using the vocabulary lookup table
# The input is a list of byte-string tokens (split from the sentence)
# Known words will be mapped to their assigned IDs from the vocabulary
# Unknown words (e.g., "faaaantastic", "alkjflkjafs") will be hashed into one of the OOV bucket IDs

table.lookup(tf.constant([b"This movie was faaaantastic alkjflkjafs".split()]))
```

```
<tf.Tensor: shape=(1, 5), dtype=int64, numpy=array([[   22,    12,    11, 10771, 10014]])>
```

```
# Define a function to encode word tokens into integer IDs using the lookup table
# Input: X_batch is a batch of tokenized reviews (RaggedTensor or dense Tensor of byte-string tokens)
# Output: A batch where each token is replaced by its corresponding vocabulary ID (or OOV bucket ID)
def encode_words(X_batch, y_batch):
    return table.lookup(X_batch), y_batch  # Leave labels unchanged

# Preprocess the training dataset:
```

```
# - Tokenize, clean, and pad each batch of reviews using the preprocess() function
train_set = datasets['train'].batch(32).map(preprocess)

# Encode the tokenized reviews using the vocabulary lookup table
train_set = train_set.map(encode_words).prefetch(1)  # Prefetch improves training performance by pipelining
```

```
<class 'tensorflow.python.framework.ops.SymbolicTensor'>
Tensor("Substr:0", shape=(None,), dtype=string)

<class 'tensorflow.python.framework.ops.SymbolicTensor'>
Tensor("StaticRegexReplace:0", shape=(None,), dtype=string)

<class 'tensorflow.python.framework.ops.SymbolicTensor'>
Tensor("StaticRegexReplace_1:0", shape=(None,), dtype=string)

<class 'tensorflow.python.ops.ragged.ragged_tensor.RaggedTensor'>
tf.RaggedTensor(values=Tensor("StringSplit/StringSplitV2:1", shape=(None,), dtype=string), row_splits=Tensor("StringSplit/RaggedFromValu
```

```
# Take the first batch of encoded training data to inspect it
# Each review is now a tensor of integer token IDs
# Each label is still 0 (negative) or 1 (positive)

for X_batch, y_batch in train_set.take(1):

    print(X_batch)
    # Prints a dense Tensor of shape (32, N), where N is the max review length in this batch
    # Each element is an integer ID from the vocabulary or one of the OOV bucket IDs

    print(y_batch)
    # Prints a Tensor of 32 labels (0 or 1), corresponding to the sentiment of each review
```

```
tf.Tensor(
[[  22   11   28 ...    0    0    0]
 [   6   21   70 ...    0    0    0]
 [4099 6881    1 ...    0    0    0]
 ...
 [  22   12  118 ...  331 1047    0]
 [1757 4101  451 ...    0    0    0]
 [3365 4392    6 ...    0    0    0]], shape=(32, 60), dtype=int64)
tf.Tensor([0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0], shape=(32,), dtype=int64)
```

```
embed_size = 128  # Dimensionality of the word embedding vectors

# Define a Sequential model for binary sentiment classification
model = keras.models.Sequential([

    # Embedding layer:
    # - Inputs are word IDs (including OOV buckets)
    # - Outputs 128-dimensional dense vectors for each word
    # - mask_zero=True tells the model to ignore padding tokens (ID 0) during training
    # - input_shape=[None] allows for variable-length input sequences
    keras.layers.Embedding(vocab_size + num_oov_buckets, embed_size, mask_zero=True, input_shape=[None]),

    # First GRU layer (return_sequences=True to feed into the next GRU layer)
    keras.layers.GRU(128, return_sequences=True),

    # Second GRU layer (final hidden state becomes the input to the Dense layer)
    keras.layers.GRU(128),

    # Output layer:
    # - Dense layer with 1 neuron and sigmoid activation for binary classification
    keras.layers.Dense(1, activation="sigmoid")
])

# Compile the model:
# - Loss function: binary_crossentropy (for 0/1 classification)
# - Optimizer: Adam (adaptive learning rate)
# - Metric: accuracy
model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(**kwargs)
```

```
history = model.fit(train_set, epochs=5)
```

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 11ms/step - accuracy: 0.6403 - loss: 0.6121
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - accuracy: 0.8404 - loss: 0.3736
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - accuracy: 0.9205 - loss: 0.2129
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9524 - loss: 0.1341
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.9611 - loss: 0.1068
```

```
K = keras.backend  # Import Keras backend functions (for Tensor operations like masking)

embed_size = 128  # Dimensionality of word embeddings

# Define the model using the Functional API for more control (e.g., custom masking)
inputs = keras.layers.Input(shape=[None])  # Input layer: sequences of variable length

# Manually create a mask: 1 where input != 0 (non-padding), 0 where input == 0 (padding)
# This replaces mask_zero=True for cases where custom mask logic is needed
mask = keras.layers.Lambda(lambda inputs: K.not_equal(inputs, 0))(inputs)

# Embedding layer: maps token IDs to dense 128-dimensional vectors
# Does NOT apply masking automatically — we'll pass our custom mask to the RNN layers
z = keras.layers.Embedding(vocab_size + num_oov_buckets, embed_size)(inputs)

# First GRU layer: returns sequences so it can feed into the next GRU
z = keras.layers.GRU(128, return_sequences=True)(z, mask=mask)

# Second GRU layer: returns the final hidden state (compressed representation of the review)
z = keras.layers.GRU(128)(z, mask=mask)

# Output layer: sigmoid activation for binary classification (positive vs. negative review)
outputs = keras.layers.Dense(1, activation="sigmoid")(z)

# Create the full model from inputs to outputs
model = keras.models.Model(inputs=[inputs], outputs=[outputs])

# Compile the model with binary crossentropy loss and accuracy metric
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model on the preprocessed and encoded training dataset
history = model.fit(train_set, epochs=5)
```

```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of `inputs` doesn't match the exp
Expected: ['keras_tensor_6']
Received: inputs=Tensor(shape=(None, None))
  warnings.warn(msg)
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 17ms/step - accuracy: 0.6458 - loss: 0.6072
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.8375 - loss: 0.3809
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.9213 - loss: 0.2113
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.9451 - loss: 0.1451
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9528 - loss: 0.1280
```