

# Chapter 14

## Workshop#5 Sentiment Analysis and Text Classification with LSTM Using spaCy (Hour 9–10)

### 14.1 Introduction

NLTK and spaCy are two major NLP Python implementation tools for basic text processing, N-gram modeling, POS tagging, and semantic analysis introduced in last four workshops. Workshop 5 will explore how to position these NLP implementation techniques into two important NLP applications: text classification and sentiment analysis. TensorFlow and Kera are two vital components to implement Long-Short Term Memory networks (LSTM networks), a commonly used Recurrent Neural Networks (RNN) on machine learning especially in NLP applications.

This workshop will:

1. study text classification concepts in NLP and how spaCy NLP pipeline works on text classifier training.
2. use movie reviews as a problem domain to demonstrate how to implement sentiment analysis with spaCy.
3. introduce Artificial Neural Networks (ANN) concepts, TensorFlow, and Kera technologies.
4. introduce sequential modeling scheme with LSTM technology using movie reviews domain as example to integrate these technologies for text classification and movie sentiment analysis.

### 14.2 Text Classification with spaCy and LSTM Technology

Text classification is a vital component in sentiment analysis application.

*TextCategorizer* is a spaCy's text classifier component applied in dataset for sentiment analysis to perform text classification with two vital Python frameworks: (1) TensorFlow Keras API and (2) spaCy technology.

Sequential data modelling with LSTM technology is used to process text for machine learning tasks with Keras's text preprocessing module and implement a neural network with tf.keras.

This workshop will cover the following key topics:

- Basic concept and knowledge of text classification.
- Model training of spaCy text classifier.
- Sentiment Analysis with spaCy.
- Sequential modeling with LSTM Technology.

## 14.3 Technical Requirements

Codes for training spaCy text classifier and sentiment analysis are spaCy v3.0 compatible. Text classification with spaCy and Keras requires Python libraries as follows:

- TensorFlow (version 2.3 or above)
- NumPy
- pandas
- Matplotlib

If these packages are not installed into PC/notebook, use *pip install xxx* command.

## 14.4 Text Classification in a Nutshell

### 14.4.1 What Is Text Classification?

Text Classification (Albrecht et al. 2020; Bird et al. 2009; George 2022; Sarkar 2019; Siahaan and Sianipar 2022; Srinivasa-Desikan 2018) is the task of assigning a set of predefined labels to text.

They are classified by manual tagging, but machine learning techniques are applied progressively to train classification system with known examples, or train samples to classify unseen cases. It is a fundamental task of NLP (Perkins 2014; Sarkar 2019) using various machine learning method such as LSTM technology (Arumugam and Shanmugamani 2018; Géron 2019; Kedia and Rasu 2020).

Text classification types are (Agarwal 2020; George 2022; Pozzi et al. 2016):

- Language detection is the first step of many NLP systems, i.e. machine translation.
- Topic generation and detection are the process of summarization, or classification of a batch of sentences, paragraphs, or texts into certain Topic of Interest (TOI) or topic titles, e.g. customers' email request refund or complaints about products or services.

- Sentiment analysis to classify or analyze users’ responses, comments, and messages on a particular topic attribute to positive, neutral, or negative sentiments. It is an essential task in e-commerce and social media platforms.

Text classifiers can emphasize on overall text sentiments, text language detection, and words levels, i.e. verbs. A text classifier of a customer service automation system is shown in Fig. 14.1.

14.4.2 Text Classification as AI Applications

Text classification is considered as Supervised Learning (SL) task in AI which means that the classifier can predict class label of a text based on sample input text-class label pairs. It must require sufficient input (text)-output (classified labels) pairs databank for network training, testing, and validation. Hence, a labeled dataset is a list of text-label pairs required to train a text classifier. An example dataset of five training sentences with sentiment labels is shown in Fig. 14.2.

When a classifier encounters a new text which is not in the training text, it predicts a class label of this unseen text based on examples during training phase to induce a text classifier output is always a class label.



Fig. 14.1 Example of top detection for customer complaint in CSAS (Customer Service Automation System)

This TV has brought me so much joy.	Pos
This is the best soccer game I have ever seen.	Pos
This dress is so ordinary not worth this expensive selling price.	Neg
Mom makes the best dinners.	Pos
Shut up, you can't talk to me like that.	Neg

Fig. 14.2 Sample input texts and their corresponding output class labels

Text classification can also be divided into (1) binary, (2) multi-class, and (3) multi-label categories:

1. Binary text classification refers to categorize text with two classes.
2. Multi-class text classification refers to categorize texts with more than two classes. Each class is mutually exclusive where one text is associated with single class, e.g. rating customer reviews are represented by 1–5 stars category single class label.
3. Multi-label text classification system is to generalize its multi-class counterpart assigned to each example text e.g. *toxic*, *severe toxic*, *insult*, *threat*, *obscenity* levels of negative sentiment. What are Labels in Text Classification?

Labels are class names for output. A class label can be categorical (string) or numerical (a number).

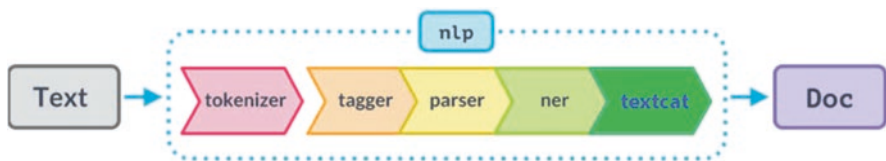
Text classification has the following class labels:

- Sentiment analysis has positive and negative class labels abbreviated by pos and neg where 0 represents negative sentiment and 1 represents positive sentiment. Binary class labels are popular as well.
- The identical numeric representation applies to binary classification problems, i.e. use 0–1 for class labels.
- Class labeled with a meaningful name for multi-class and multi-label problems, e.g. movie genre classifier has labels *action*, *scifi*, *weekend*, *Sunday movie*, etc. Numbers are labels for a five-class classification problem, i.e. 1–5.

## 14.5 Text Classifier with spaCy NLP Pipeline

*TextCategorizer* (*tCategorizer*) is spaCy’s text classifier component (Altinok 2021; SpaCy 2022; Vasiliev 2020). It required class labels and examples in NLP pipeline to perform training procedure as shown in Fig. 14.3.

*TextCategorizer* provides user-friendly and end-to-end approaches to train classifier so that it does not need to deal with neural network architecture directly.



**Fig. 14.3** *TextCategorizer* in the spaCy NLP pipeline

### 14.5.1 *TextCategorizer* Class

Import spaCy and load nlp component from "en\_core\_web\_md":

```
In[1] ▶ # Load and import spacy package
import spacy
# Load the en_core_web_md module
nlp = spacy.load( "en_core_web_md" )
```

Import *TextCategorizer* from spaCy pipeline components:

```
In[2] ▶ # Import the Single Text Categorizer Model
from spacy.pipeline.textcat import
DEFAULT_SINGLE_TEXTCAT_MODEL
```

*TextCategorizer* consists of (1) single-label and (2) multi-label classifiers.

A multi-label classifier can predict more than single class. A single-label classifier predicts a single class for each example and classes are mutually exclusive.

The preceding import line imports single-label classifier, and the following code imports multi-label classifier:

```
In[3] ▶ # Import the Multiple Text Categorizer Model
from spacy.pipeline.textcat_multilabel import
DEFAULT_MULTI_TEXTCAT_MODEL
```

There are two parameters (1) a threshold value and (2) a model name (either Single or Multi depends on classification task) required for a *TextCategorizer* component configuration.

*TextCategorizer* generates a probability for each class and a class is assigned to text if the probability of this class is higher than the threshold value.

A traditional threshold value for text classification is 0.5; however, if prediction is required for a higher confidence, it can adjust threshold to 0.6–0.8.

A single-label *TextCategorizer* (*tCategorizer*) component is added to nlp pipeline as follows:

```
In[4] ▶ # Import the Single Text Categorizer Model
# Define the model parameters: threshold and model
from spacy.pipeline.textcat import
DEFAULT_SINGLE_TEXTCAT_MODEL
config = {
    "threshold": 0.5,
    "model": DEFAULT_SINGLE_TEXTCAT_MODEL
}
```

```
In[5] # Define the Text Categorizer object (tCategorizer)
tCategorizer = nlp.add_pipe("textcat", config=config)
```

Let's look at Text Categorizer object (tCategorizer):

```
In[6] tCategorizer
Out[6] <spacy.pipeline.textcat.TextCategorizer at 0x1bf406cedc0>
```

Add a multilabel component to nlp pipeline:

```
In[7] # Import the Multiple Text Categorizer Model
# Define the model parameters: threshold and model
from spacy.pipeline.textcat_multilabel import
DEFAULT_MULTI_TEXTCAT_MODEL
config = {
    "threshold": 0.5,
    "model": DEFAULT_MULTI_TEXTCAT_MODEL
}
```

```
In[8] tCategorizer = nlp.add_pipe("textcat_multilabel", config=config)
```

```
In[9] tCategorizer
Out[9] <spacy.pipeline.textcat_multilabel.MultiLabel_TextCategorizer at 0x262412efe20>
```



Add a *TextCategorizer* pipeline component to nlp pipeline object at the last line of each preceding code blocks. The newly created *TextCategorizer* component is captured by textcat variable and set for training

## 14.5.2 Formatting Training Data for the TextCategorizer

Let us prepare a customer sentiment dataset for binary text classification.

The label (category) will be called sentiment to obtain two possible values, 0 and 1 corresponding to negative and positive sentiments.

There are six examples from IMDB with three each of positive and negative as below:

```
In[10] movie_comment1 = [
    ("This movie is perfect and worth watching. ",
     {"cats": {"Positive Sentiment": 1}}),
    ("This movie is great, the performance of Al Pacino is brilliant.",
     {"cats": {"Positive Sentiment": 1}}),
    ("A very good and funny movie. It should be the best this year!",
     {"cats": {"Positive Sentiment": 1}}),
    ("This movie is so bad that I really want to leave after the first hour
     watching.", {"cats": {"Positive Sentiment": 0}}),
    ("Even free I won't see this movie again. Totally failure!",
     {"cats": {"Positive Sentiment": 0}}),
    ("I think it is the worst movie I saw so far this year.",
     {"cats": {"Positive Sentiment": 0}})
]
```

Check on any movie comment1 element:

```
In[11] movie_comment1 [1]
Out[11] ('This movie is great, the performance of Al Pacino is brilliant.',
         {'cats': {'Positive Sentiment': 1}})
```



- Each training example (movie\_coment1) is a tuple object consists of a text and a nested dictionary
- The dictionary contains a class category in a format recognized by spaCy
- The *cats* field means categories
- Include class category sentiment and its value. The value should always be a floating-point number

The code will introduce a class category selected for *TextCategorizer* component.

```
In[12] import random
import spacy
from spacy.training import Example
from spacy.pipeline.textcat import
DEFAULT_SINGLE_TEXTCAT_MODEL
```



- Import a built-in library *random* to shuffle dataset
- Import *spaCy* as usual, then import *Example* to prepare training samples in *spaCy* format
- Import *TextCategorizer* model in final statement

Initialize pipeline and *TextCategorizer* component.

When a new *TextCategorizer* component *tCategorizer* is created, use calling *add\_label* method to introduce category sentiment to *TextCategorizer* component with examples.

The following code adds label to *TextCategorizer* component and initialize *TextCategorizer* model's weights with training samples:

```
In[13] ▶ import random
import spacy
from spacy.training import Example
from spacy.pipeline.textcat import
DEFAULT_SINGLE_TEXTCAT_MODEL

# Load the spaCy NLP model
nlp = spacy.load('en_core_web_md')

# Set the threshold and model
config = {
    "threshold": 0.5,
    "model": DEFAULT_SINGLE_TEXTCAT_MODEL
}

# Define TextCategorizer object (tCategorizer)
tCategorizer = nlp.add_pipe("textcat", config=config)
```


Let's look at *pipe\_names*:

```
In[14] ▶ nlp.pipe_names
Out[14] ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer',
'ner', 'textcat']
```

When a new *TextCategorizer* component *textcat* is created, use calling *add\_label* method to introduce label sentiment to the *TextCategorizer* component and initialize this component with examples.

The following code adds label to *TextCategorizer* component and initializes *TextCategorizer* model's weights with training samples (*movie\_comment\_exp*):



```
In[15]  # Create the two sentiment categories
tCategorizer.add_label("Positive Sentiment")
tCategorizer.add_label("Negative Sentiment")

# Create the movie comment samples
movie_comment_exp = [Example.from_dict(nlp.make_doc(comments),
category) for comments,category in movie_comment1]
tCategorizer.initialize(lambda: movie_comment_exp, nlp=nlp)
```

Let's look at `movie_comment_exp`:

[illegible]

### 14.5.3 System Training

Training loop is all set to be defined.

First, disable other pipe components to allow only textcat can be trained.

Second, create an optimizer object by calling *resume\_training* to keep the weights of existing statistical models.

Examine each epoch training example one by one and update the weights of textcat. Examine data for 20 epochs.

Try the whole program with training loop:

In[17]



movie\_comment1

Out[17]

```
[('This movie is perfect and worth watching. ',
 {'cats': {'Positive Sentiment': 1}}),
 ('This movie is great, the performance of Al Pacino is brilliant.',
 {'cats': {'Positive Sentiment': 1}}),
 ('A very good and funny movie. It should be the best this year!',
 {'cats': {'Positive Sentiment': 1}}),
 ('This movie is so bad that I really want to leave after the first hour watch-
ing.',
 {'cats': {'Positive Sentiment': 0}}),
 ('Even free I won't see this movie again. Totally failure!',
 {'cats': {'Positive Sentiment': 0}}),
 ('I think it is the worst movie I saw so far this year.',
 {'cats': {'Positive Sentiment': 0}})]
```

In[18] ▶

```

# Full implementation of the Movie Sentiment Analysis System
import random
import spacy
from spacy.training import Example
from spacy.pipeline.textcat import
DEFAULT_SINGLE_TEXTCAT_MODEL

# Load the spaCy NLP model
nlp = spacy.load('en_core_web_md')

# Set the threshold and model
config = {
    "threshold": 0.5,
    "model": DEFAULT_SINGLE_TEXTCAT_MODEL
}

# Create the TextCategorizer object (tCategorizer)
tCategorizer = nlp.add_pipe("textcat", config=config)

# Add the two movie sentiment categories
tCategorizer.add_label("Positive Sentiment")
tCategorizer.add_label("Negative Sentiment")

# Create the movie sample comments
movie_comment_exp = [Example.from_dict(nlp.make_doc(comments),
category) for comments,category in movie_comment1]
tCategorizer.initialize(lambda: movie_comment_exp, nlp=nlp)

# Set the training epochs and loss values
epochs=20
losses = {}

# Main program loop
with nlp.select_pipes(enable="textcat"):
    optimizer = nlp.resume_training()
    for i in range(epochs):
        random.shuffle(movie_comment1)
        for comments, category in movie_comment1:
            mdoc = nlp.make_doc(comments)
            exp = Example.from_dict(mdoc, category)
            nlp.update([exp], sgd=optimizer, losses=losses)
        print("Epoch #",i, "Losses: ",losses)

```

Out[18]

Epoch # 0 Losses: {'textcat': 1.5183179080486298}  
Epoch # 1 Losses: {'textcat': 2.7817106544971466}  
Epoch # 2 Losses: {'textcat': 3.5506987050175667}  
Epoch # 3 Losses: {'textcat': 3.8420143127441406}  
Epoch # 4 Losses: {'textcat': 3.9003750435076654}  
Epoch # 5 Losses: {'textcat': 3.9074860664550215}  
Epoch # 6 Losses: {'textcat': 3.908426207563025}  
Epoch # 7 Losses: {'textcat': 3.908603171435061}  
Epoch # 8 Losses: {'textcat': 3.9086502377413126}  
Epoch # 9 Losses: {'textcat': 3.908672676368724}  
Epoch # 10 Losses: {'textcat': 3.9086846519847427}  
Epoch # 11 Losses: {'textcat': 3.908692961549093}  
Epoch # 12 Losses: {'textcat': 3.9086987949742706}  
Epoch # 13 Losses: {'textcat': 3.9087034759107553}  
Epoch # 14 Losses: {'textcat': 3.908707513363254}  
Epoch # 15 Losses: {'textcat': 3.90871090364098}  
Epoch # 16 Losses: {'textcat': 3.9087138364217537}  
Epoch # 17 Losses: {'textcat': 3.9087164432144874}  
Epoch # 18 Losses: {'textcat': 3.9087187692100116}  
Epoch # 19 Losses: {'textcat': 3.908720835553922}

14.5.4 System Testing


Let us test a new textcategorizer component, doc.cats property holds the class labels:

In[19] ▶ # Test 1: This movie sucks  
test1 = nlp("This movie sucks and the worst I ever saw.")  
test1.cats

Out[19] {'Positive Sentiment': 0.05381184443831444,  
'Negative Sentiment': 0.9461881518363953}

In[20] ▶ # Test 2: I'll watch it again, how amazing.  
test2 = nlp("This movie really very great!")  
test2.cats

Out[20] {'Positive Sentiment': 0.8159973621368408,  
'Negative Sentiment': 0.1840025931596756}



The small dataset trained spaCy text classifier successfully for a binary text classification problem to perform correct sentiment analysis. Now, let us perform multi-label classification

### 14.5.5 Training *TextCategorizer* for Multi-Label Classification

Multi-label classification means the classifier can predict more than single label for an example text. Naturally, the classes are not mutually exclusive.

Provide training samples with at least more than 2 categories to train a multiple labelled classifier.

Construct a small training set to train spaCy's *TextCategorizer* for multi-label classification. This time will form a set of movie reviews, where the multi-category is:

- ACTION.
- SCIFI.
- WEEKEND.

Here is a small sample dataset (*movie\_comment2*):

```
In[21] movie_comment2 = [
    ("This movie is great for weekend watching.",
     {"cats": {"WEEKEND": True}}),
    ("This a 100% action movie, I enjoy it.",
     {"cats": {"ACTION": True}}),
    ("Avatar is the best Scifi movie I ever seen!" ,
     {"cats": {"SCIFI": True}}),
    ("Such a good Scifi movie to watch during the weekend!",
     {"cats": {"WEEKEND": True, "SCIFI": True}}),
    ("Matrix a great Scifi movie with a lot of action. Pure action, great!",
     {"cats": {"SCIFI": True, "ACTION": True}})
]
```

Check dataset first:

```
In[22] movie_comment2
Out[22] [('This movie is great for weekend watching.', {'cats': {'WEEKEND':
True}}),
('This a 100% action movie, I enjoy it.', {'cats': {'ACTION': True}}),
('Avatar is the best Scifi movie I ever seen!', {'cats': {'SCIFI': True}}),
('Such a good Scifi movie to watch during the weekend!',
 {'cats': {'WEEKEND': True, 'SCIFI': True}}),
('Matrix a great Scifi movie with a lot of action. Pure action, great!',
 {'cats': {'SCIFI': True, 'ACTION': True}})]
```

In[23] ▶	movie_comment2[1]
Out[23]	('This a 100% action movie, I enjoy it.', {'cats': {'ACTION': True}})

Provide examples with a single label, such as first example (the first sentence of *movie\_comment2*, the second line of preceding code block), and examples with more than single label, such as fourth example of *movie\_comment2*.

Import after the training set is formed.

In[24] ▶	<pre>import random import spacy from spacy.training import Example from spacy.pipeline.textcat_multilabel import DEFAULT_MULTI_TEXTCAT_MODEL  # Load spaCy NLP model nlp = spacy.load('en_core_web_md')</pre>
----------	---

Note that the last line has different code than previous section. Import multi-label model instead of single-label model.

Next, add multi-label classifier component to nlp pipeline.

Also note that pipeline component name is *textcat\_multilabel* as compared with previous section's *textcat*:

In[25] ▶	<pre># Set the threshold and model config = {     "threshold": 0.5,     "model": DEFAULT_MULTI_TEXTCAT_MODEL }  # Create the TextCategorizer object (tCategorizer) tCategorizer = nlp.add_pipe("textcat_multilabel", config=config)</pre>
----------	---

Add categories to *TextCategorizer* component and initialize model like previous text classifier section.

Add three labels instead of one:

```
In[26] ▶ # Create the categorizer object with 3 categories
categories = ["SCIFI", "ACTION", "WEEKEND"]

# Using For Loop to add the 3 categories
for category in categories:
    tCategorizer.add_label(category)

# Create the movie comment sample for training
movie_comment_exp = [Example.from_dict(nlp.make_doc(comments), category) for comments, category in movie_comment2]

# Initialize the tCategorizer
tCategorizer.initialize(lambda: movie_comment_exp, nlp=nlp)
```

Training loop is all set to be defined.

Code functions are like previous section's code, the only difference is component name `textcat_multilabel` in the first line:

```
In[27] ▶ # Set the training epochs and loss values
epochs=20
losses = {}

# Main Loop of the program
with nlp.select_pipes(enable="textcat_multilabel"):
    optimizer = nlp.resume_training()
    for i in range(epochs):
        random.shuffle(movie_comment2)
        for comments, category in movie_comment2:
            mdoc = nlp.make_doc(comments)
            exp = Example.from_dict(mdoc, category)
            nlp.update([exp], sgd=optimizer, losses=losses)
        print(losses)
```

```
Out[27] {'textcat_multilabel': 6.85278207868123e-06}
{'textcat_multilabel': 1.3591816482971808e-05}
{'textcat_multilabel': 2.003331736943892e-05}
{'textcat_multilabel': 2.6209833507095937e-05}
{'textcat_multilabel': 3.211208475306648e-05}
{'textcat_multilabel': 3.780755992011109e-05}
{'textcat_multilabel': 4.3277938615915446e-05}
{'textcat_multilabel': 4.857392603696553e-05}
{'textcat_multilabel': 5.372697206951216e-05}
{'textcat_multilabel': 5.867910277856936e-05}
{'textcat_multilabel': 6.350277087108225e-05}
{'textcat_multilabel': 6.81268817857017e-05}
{'textcat_multilabel': 7.271952523524305e-05}
{'textcat_multilabel': 7.709734516936351e-05}
{'textcat_multilabel': 8.136703193883932e-05}
{'textcat_multilabel': 8.552625510560574e-05}
{'textcat_multilabel': 8.953699784797209e-05}
{'textcat_multilabel': 9.34374557175488e-05}
{'textcat_multilabel': 9.724928190735227e-05}
{'textcat_multilabel': 0.00010092528287941605}
```


The output should look like output of previous section but use multiple category for system training. Let’s test the new multi-label classifier:

```
In[28] test3 = nlp("Definitely in my weekend scifi movie night list")
test3.cats

Out[28] {'SCIFI': 0.9995421171188354,
'ACTION': 0.5897207260131836,
'WEEKEND': 0.9992324113845825}

In[29] test4 = nlp("Go to watch action scifi movie this weekend.")
test4.cats

Out[29] {'SCIFI': 0.994023859500885,
'ACTION': 0.9914324283599854,
'WEEKEND': 0.9998989105224609}
```



Although sample size is small, but the multiple textcategorizer can classify two IMDB user comments correctly into three categories: SCIFI, ACTION, and WEEKEND. Note that over thousands of IMDB user comments are required to perform a satisfactory sentiment analysis in real situations



This section has learnt how to train a spaCy's *TextCategorizer* component for binary and multilabel text classifications.

Now, *TextCategorizer* will be trained on a real-world dataset for a sentiment analysis using IMDB user comments dataset.



#### Workshop 5.1 Movie comments from IMDB.com

Movie comments is a significant classification in social media. This workshop constructs a simple movie comment classification with millions of user comments from IMDB.com, the world biggest movie social media platform.

1. Try to collect 900 comments with 300 *Good*, 300 *Average* and 300 *Bad* comments to train system. Make sure they make sense or system won't function
2. Construct a Multi-label Classification System to create three movie comments: *Good*, *Average* or *Bad*
3. Train system with at least 100 epochs
4. Use 10 examples to test and see whether it works

## 14.6 Sentiment Analysis with spaCy

### 14.6.1 IMDB Large Movie Review Dataset

This section will work on a real-world dataset using IMDB Large Movie Reviews Dataset from Kaggle (2022).

The original *imdb\_sup.csv* dataset has 50,000 rows. They need to down-size and select first 5000 records into datafile *imdb\_5000.csv* to speed up training. This movie reviews dataset consists of movie reviews, reviews sizes, IMDB Ratings (1–10), and Sentiment Ratings (0 or 1).

The dataset can be downloaded from workshop directory namely: *imdb\_sup.csv* (complete dataset) or *imdb\_5000.csv* (5000 records).


### 14.6.2 Explore the Dataset


Let us have some understanding from dataset prior sentiment analysis.

1. First, import to read and visualize dataset:

```
In[30] ▶ import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

2. Read imdb\_5000.csv datafile into a pandas DataFrame (mcommentDF) and output the shape of DataFrame:


In[31]  mcommentDF=pd.read\_csv( 'imdb\_5000.csv' )

In[32]  mcommentDF.shape

Out[32] (5000, 3)

Note: This IMDB movie reviews dataset contains 5000 records, each record has 3 fields attributes: Review, Rating, and Sentiment

3. Examine rows and columns of dataset by printing the first few rows using head() method:

In[33]  mcommentDF.head()

Out[33]

	Review	Rating	Sentiment
0	**Possible Spoilers**	1	0
1	Read the book, forget the movie!	2	0
2	**Possible Spoilers Ahead**	2	0
3	What a script, what a story, what a mess!	2	0
4	I hope this group of film-makers never re-unites.	1	0

4. Use Review and Sentiment columns only in this workshop. Hence, drop other columns that won't use, and call dropna() method to drop the rows with missing values:

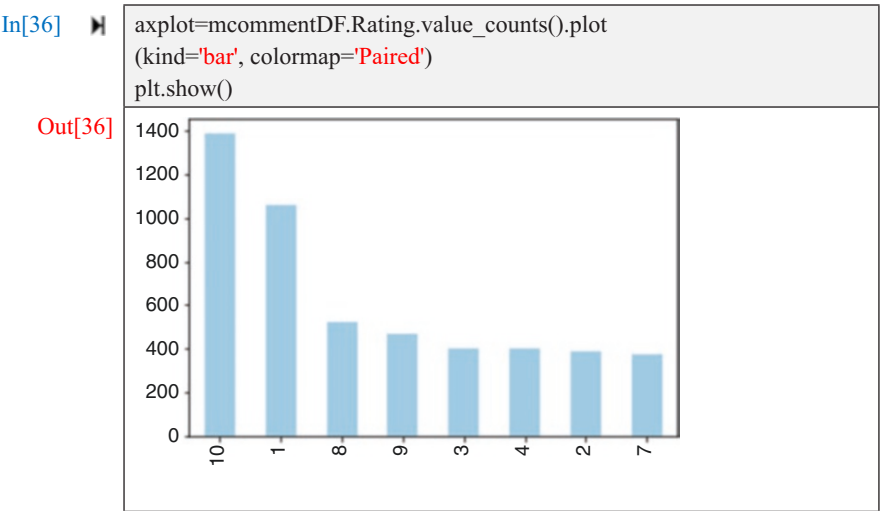
In[34]  mcommentDF\_clean = mcommentDF[[ 'Review', 'Sentiment' ]].dropna()


In[35] ▶ mcommentDF\_clean.head()

Out[35]

	Review	Sentiment
0	<b>**Possible Spoilers**</b>	0
1	Read the book, forget the movie!	0
2	<b>**Possible Spoilers Ahead**</b>	0
3	What a script, what a story, what a mess!	0
4	I hope this group of film-makers never re-unites.	0

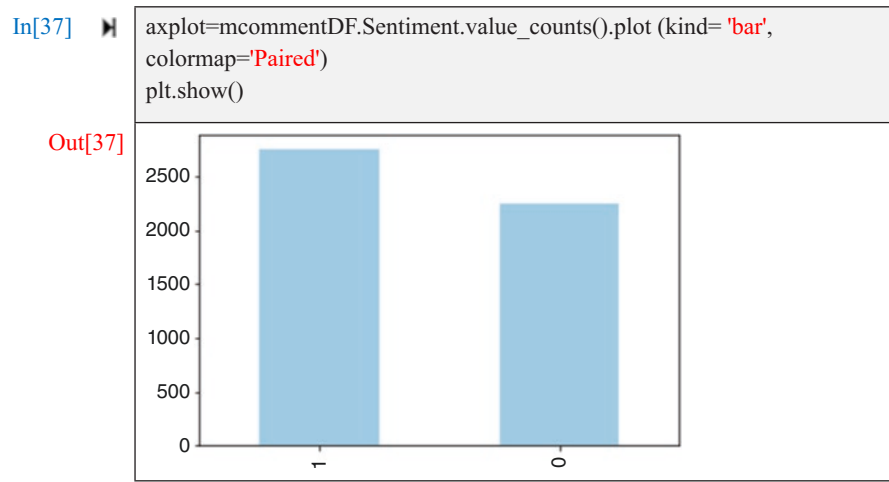
5. Let us look at how review scores are distributed:




- 
1. Users prefer to give high rating, i.e. 8 or above, and 10 is the highest as shown
  2. It is better to select sample set with even distribution to balance sample data rating
  3. Check system performance first. If it is not as good as predicted, can use fine-tune sampling method to improve system performance

Here use the sentiments already labeled.

6. Plot ratings distribution:





Note that rating distribution has better results than the previous one, it has higher number of positive reviews, but negative reviews are also significant as shown


After the dataset is processed, it can be reduced to a two-column dataset with negative and positive ratings. So, call `mcommentDF.head()` again and the following result is obtained:

In[38]

mcommentDF.head()

Out[38]

	Review	Rating	Sentiment
0	<b>**Possible Spoilers**</b>	1	0
1	Read the book, forget the movie!	2	0
2	<b>**Possible Spoilers Ahead**</b>	2	0
3	What a script, what a story, what a mess!	2	0
4	I hope this group of film-makers never re-unites.	1	0



Complete dataset exploration and display review scores with class categories distribution. The dataset is ready to be processed. Drop unused columns and convert review scores to binary class labels. Let us begin with the training procedure

### 14.6.3 Training the TextClassifier

Use multi-label classifier to train a binary text classifier this time.

1. Import spaCy classes as follows:

```
In[39] ▶ import spacy
import random
from spacy.training import Example
from spacy.pipeline.textcat_multilabel import
DEFAULT_MULTI_TEXTCAT_MODEL
```

2. Create a pipeline object `nlp`, define classifier configuration, and add *TextCategorizer* component to `nlp` with the following configuration:

```
In[40] ▶ # Load the spaCy NLP model
nlp = spacy.load("en_core_web_md")

# Set the threshold and model
config = {
    "threshold": 0.5,
    "model": DEFAULT_MULTI_TEXTCAT_MODEL
}

# Create the TextCategorizer object (tCategorizer)
tCategorizer = nlp.add_pipe("textcat_multilabel", config=config)
```

3. When *TextCategorizer* object is available, create movie comment sample object as a list and load all user comments and categories into it.

```
In[41] ▶ # Create the IMDB movie comment sample object
movie_comment_exp = []

# Load all the IMDB user comments and categories
for idx, rw in mcommentDF.iterrows():
    comments = rw["Review"]
    rating = rw["Sentiment"]
    category = {"POS": True, "NEG": False} if rating == 1 else
{"NEG": True, "POS": False}
    movie_comment_exp.append(Example.from_dict(nlp.make_doc
(comments), {"cats": category}))
```

4. Let's check `movie_comment_exp`:

```
In[42] movie_comment_exp[0]
Out[42] {'doc_annotation': {'cats': {'NEG': True, 'POS': False}, 'entities': ['O', 'O', 'O', 'O', 'O', 'O'], 'links': {}}, 'token_annotation': {'ORTH': ['*', '*', 'Possible', 'Spoilers', '*', '*'], 'SPACY': [False, False, True, False, False, False], 'TAG': [' ', ' ', ' ', ' ', ' ', ' '], 'LEMMA': [' ', ' ', ' ', ' ', ' ', ' '], 'POS': [' ', ' ', ' ', ' ', ' ', ' '], 'MORPH': [' ', ' ', ' ', ' ', ' ', ' '], 'HEAD': [0, 1, 2, 3, 4, 5], 'DEP': [' ', ' ', ' ', ' ', ' ', ' '], 'SENT_START': [1, 0, 0, 0, 0, 0]}}
```

5. Use POS and NEG labels for positive and negative sentiment respectively. Introduce these labels to the new component and initialize it with examples.

```
In[43] # Add the two sentiment categories into tCategorizer
tCategorizer.add_label("POS")
tCategorizer.add_label("NEG")
tCategorizer.initialize(lambda: movie_comment_exp, nlp=nlp)
```

```
In[44] tCategorizer
Out[44] <spacy.pipeline.textcat_multilabel.MultiLabel_TextCategorizer at 0x2626f-9c4ee0>
```

6. Define training loop by examining the training set for two epochs but can examine further if necessary. The following code snippet will train the new text categorizer component:

```
In[45] # Set the training epochs to 2 to save time
epochs = 2

# Main program loop
with nlp.select_pipes(enable="textcat_multilabel"):
    optimizer = nlp.resume_training()
    for i in range(epochs):
        random.shuffle(movie_comment_exp)
        for exp in movie_comment_exp:
            nlp.update([exp], sgd=optimizer)
```

7. Test how text classifier component works for two example sentences:

```
In[46] ▶ test5 = nlp("This is the best movie that I have ever watched")
```

```
In[47] ▶ test5.cats
```

```
Out[47] {'POS': 0.9857660531997681, 'NEG': 0.018266398459672928}
```

```
In[48] ▶ test6 = nlp("This movie is so bad")
```

```
In[49] ▶ test6.cats
```

```
Out[49] {'POS': 0.1364014744758606, 'NEG': 0.8908849358558655}
```



Note both NEG and POS labels appeared in prediction results because it used multi-label classifier. The results are satisfactory, but it can improve if the numbers for training epochs are increased. The first sentence has a high positive probability output, and the second sentence has predicted as negative with a high probability

SpaCy's text classifier component training is completed.

The next section will explore Kera, a popular deep learning library and how to write Keras code for text classification with another machine learning library—TensorFlow's Keras API.

## 14.7 Artificial Neural Network in a Nutshell

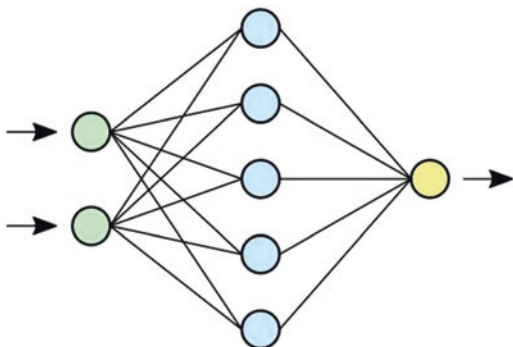
This workshop section will learn how to incorporate spaCy technology with ANN (Artificial Neural Networks) technology using TensorFlow and its Keras package (Géron 2019; Kedia and Rasu 2020; TensorFlow 2022).

A typical ANN has:

1. Input layer consists of input neurons, or nodes.
2. Hidden layer consists of hidden neurons, or nodes.
3. Output layer consists of output neurons, or nodes

ANN will learn knowledge by its network weights update through network training with sufficient sample inputs and target outputs pairs. The network can predict or match unseen inputs to corresponding output result after it had sufficient training to a predefined accuracy. A typical ANN architecture is shown in Fig. 14.4.

**Fig. 14.4** System architecture of ANN



## 14.8 An Overview of TensorFlow and Keras

TensorFlow (Géron 2019; TensorFlow 2022) is a popular Python tool widely used for machine learning. It has huge community support and great documentation available at TensorFlow official site (TensorFlow 2022), while Keras is a Python based deep learning tool that can be integrated with Python platforms such as TensorFlow, Theano, and CNTK.

TensorFlow 1 was disagreeable to symbolic graph computations and other low-level computations, but TensorFlow 2 initiated great changes in machine learning methods allowing developers to use Keras with TensorFlow's low-level methods. Keras is popular in R&D because it supports rapid prototyping and user-friendly API to neural network architectures (Kedia and Rasu 2020; Srinivasa-Desikan 2018).

Neural networks are commonly used for computer vision and NLP tasks including object detection, image classification, scene understanding, text classification, POS tagging, text summarization, and natural language generation.

TensorFlow 2 will be used to study the details of a neural network architecture for text classification with `tf.keras` implementation throughout this section.

## 14.9 Sequential Modeling with LSTM Technology

Long-Short Term Memory network (LSTM network) is one of the significant recurrent networks used in various machine learning applications such as NLP applications nowadays (Ekman 2021; Korstanje 2021).

RNNs are special neural networks that can process sequential data in steps.

All inputs and outputs are independent but not for text data in neural networks. Every word's presence depends on neighboring words, e.g. a word is predicted by considering all preceding predicted words and stored the past sequence token of words within a LSTM cell in a machine translation task. A LSTM is showed in Fig. 14.5.



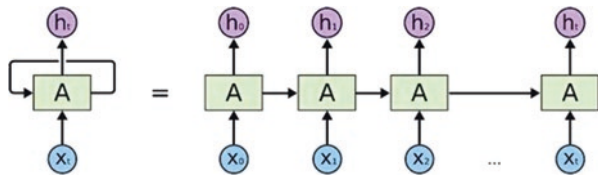


Fig. 14.5 RNN with LSTM technology

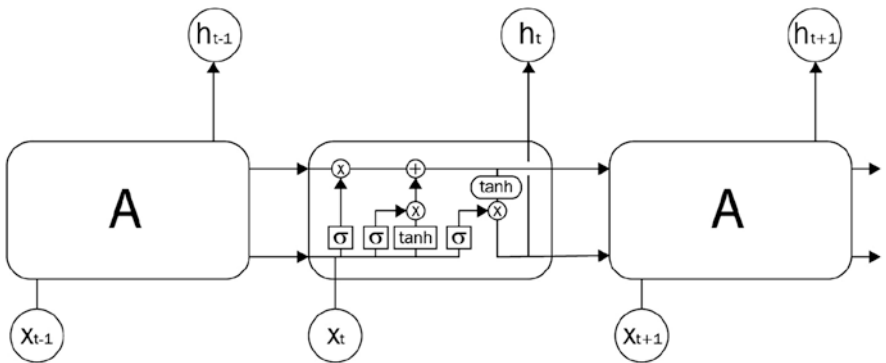


Fig. 14.6 Architecture of LSTM cell

An LSTM cell is moderately complex than an RNN cell, but computation logic is identical. A diagram of a LSTM cell is shown in Fig. 14.6. Note that input and output steps are identical to RNN counterparts:

Keras has extensive support for RNN variations GRU, LSTM, and simple API for training RNNs. RNN variations are crucial for NLP tasks as language data’s nature is sequential, i.e. text is a sequence of words, speech is a sequence of sounds, and so on.

Since the type of statistical model has identified in the design, it can transform a sequence of words into a word IDs sequence and build vocabulary with Keras pre-processing module simultaneously.

### 14.10 Keras Tokenizer in NLP

Text is a sequence of words or characters data. A sentence can be fed by a tokens sequence. Hence, tokens are to be vectorized first by the following steps:

1. Tokenize each utterance and turn these utterances into a sequence of tokens.
2. Build a vocabulary from set of tokens presented in step 1. These are tokens to be recognized by neural network design.
3. Create a vocabulary and assign ID to each token.

## 4. Map token vectors with corresponding token IDs.

Let us look at a short example of a corpus for three sentences:

```
In[50] testD = [ "I am going to buy a gift for Christmas tomorrow morning.",
                "Yesterday my mom cooked a wonderful meal.",
                "Jack promised he would remember to turn off the lights." ]
```

```
In[51] testD
Out[51] ['I am going to buy a gift for Christmas tomorrow morning.',
         'Yesterday my mom cooked a wonderful meal.',
         'Jack promised he would remember to turn off the lights.']
```

Let's tokenize words into utterances:

```
In[52] import spacy

# Load the NLP model
nlp = spacy.load("en_core_web_md")

# Create the utterances object
utterances = [[token.text for token in nlp(utterance)] for utterance in testD]
for utterance in utterances:
    utterance
```



All tokens of Doc object generated by calling `nlp(sentence)` are iterated in the preceding code. Note that punctuation marks have not filtered as this filtering depends on the task e.g. punctuation marks such as '!', correlate to the result in sentiment analysis, they are preserved in this example.

Build vocabulary and token sequences into token-ID sequences using *Tokenizer* as shown:

```
In[53] # Import Tokenizer
from tensorflow.keras.preprocessing.text import Tokenizer

# Create tokenizer object (ktoken)
ktoken = Tokenizer(lower=True)
```

In[54] ▶	<pre>ktoken.fit_on_texts(testD) ktoken</pre>
Out[54]	<pre>&lt;keras_preprocessing.text.Tokenizer at 0x2322f3ae970&gt;</pre>

In[55] ▶	<pre>ktoken.word_index</pre>
Out[55]	<pre>{'to': 1, 'i': 2, 'am': 3, 'going': 4, 'buy': 5, 'some': 6, 'gift': 7, 'for': 8, 'christmas': 9, 'tomorrow': 10, 'morning': 11, 'yesterday': 12, 'my': 13, 'mom': 14, 'cooked': 15, 'a': 16, 'wonderful': 17, 'meal': 18, 'john': 19, 'promised': 20, 'he': 21, 'would': 22, 'remember': 23, 'turn': 24, 'off': 25, 'the': 26, 'lights': 27}</pre>



The following are performed in the above codes:

1. Import Tokenizer from Keras text preprocessing module
2. Create a tokenizer object (*ktoken*) with parameter `lower=True`, which means tokenizer should lower all words for vocabulary formation
3. Call *ktoken.fit\_on\_texts* on data to form vocabulary. *fit\_on\_text* work on a tokens sequence; input should always be a list of tokens
4. Examine vocabulary by printing *ktoken.word\_index*. *Word\_index* is a dictionary where keys are vocabulary tokens and values are token-IDs

Call *ktoken.texts\_to\_sequences()* method to retrieve a token ID.

Notice that the input to this method should always be a list, even if single token is fed.

Feed one-word input as a list (notice list brackets) in the following code segment:

In[56] ▶	<pre>ktoken.texts_to_sequences(["Christmas"])</pre>
Out[56]	<pre>[[9]]</pre>

In[57] ▶	<pre>ktoken.texts_to_sequences(["cooked", "meal"])</pre>
Out[57]	<pre>[[15], [18]]</pre>



1. Note token-IDs start from 1 and not 0. 0 is a reserved value, which means a padding value with specific meaning
2. Keras cannot process utterances of different lengths, hence need to pad all utterances
3. Pad each sentence of dataset to a maximum length by adding padding utterances either at the start or end of utterances
4. Keras inserts 0 for the padding which means it is a padding value without a token

Let's understand how padding works with a simple example.

```
In[58] ▶ # Import the pad_sequences package
        from tensorflow.keras.preprocessing.sequence import pad_sequences

        # Create the utterance sequences
        seq_utterance = [[7], [8,1], [9,11,12,14]]

        # Define Maximum Length (MLEN)
        MLEN=4

        # Pad the utterance sequences.
        pad_sequences(seq_utterance, MLEN, padding="post")
```

```
Out[58] array([[ 7,  0,  0,  0],
               [ 8,  1,  0,  0],
               [ 9, 11, 12, 14]])
```

```
In[59] ▶ pad_sequences(seq_utterance, MLEN, padding="pre")
```

```
Out[59] array([[ 0,  0,  0,  7],
               [ 0,  0,  8,  1],
               [ 9, 11, 12, 14]])
```

Call `pad_sequences` on this sequences list and every sequence is padded with zeros so that its length reaches `MAX_LEN=4` which is the length of the longest sequence. Then pad sequences from the right or left with *post* and *pre* options. Sentences with post option were padded in the preceding code, hence the sentences were padded from the right.

When these sequences are organized, the complete text preprocessing steps are as follows:

```
In[60] # Import the Tokenizer and pad sequences package
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Create the token object
ktoken = Tokenizer(lower=True)
ktoken.fit_on_texts(testD)

# Create the sequence utterance object
sutterance = ktoken.texts_to_sequences(testD)
MLEN=7

# Pad the utterance sequences
pseq_utterance = pad_sequences(sutterance, MLEN, padding="post")
pseq_utterance
```

```
Out[60] array([[ 5,  6,  7,  8,  9, 10, 11],
               [12, 13, 14, 15, 16, 17, 18],
               [22, 23,  1, 24, 25, 26, 27]])
```

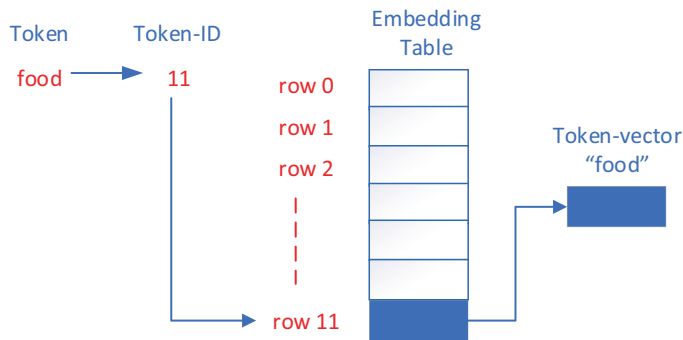


Transform utterances into a token-IDs sequence for tokens vectorization so that utterances will be ready to feed into neural network.

### 14.10.1 Embedding Words

Tokens can be transformed into token vectors. Embedding tokens into vectors occurred via a lookup embedding table. Each row holds a token vector indexed by token-IDs, hence the flow of obtaining a token vector is as follows:

1. token->token-ID: A token-ID is assigned with each token with Keras' Tokenizer in previous section. Tokenizer holds all vocabularies and maps each vocabulary token to an ID.
2. token-ID->token vector: A token-ID is an integer that can be used as an index to embed table's rows. Each token-ID corresponds to one row and when a token vector is required, first obtain its token-ID and lookup in the embedding table rows with this token-ID.



**Fig. 14.7** A sample of embedding words into token vectors

- A sample of embedding words into token vectors is shown in Fig. 14.7. Remember when a list of utterances began in the previous section:
1. Each utterance is divided into tokens and built a vocabulary with Keras' Tokenizer.
  2. The Tokenizer object held a token index with a token->token-ID mapping.
  3. When a token-ID is obtained, lookup to embedding table rows with this token-ID to acquire a token vector.
  4. This token vector is fed to neural network.

There are several steps to transform sentences into vectors as training a neural network is complex.

A LSTM neural network architecture can be designed to perform model training after these preliminary steps.

### 14.11 Movie Sentiment Analysis with LTSM Using Keras and spaCy

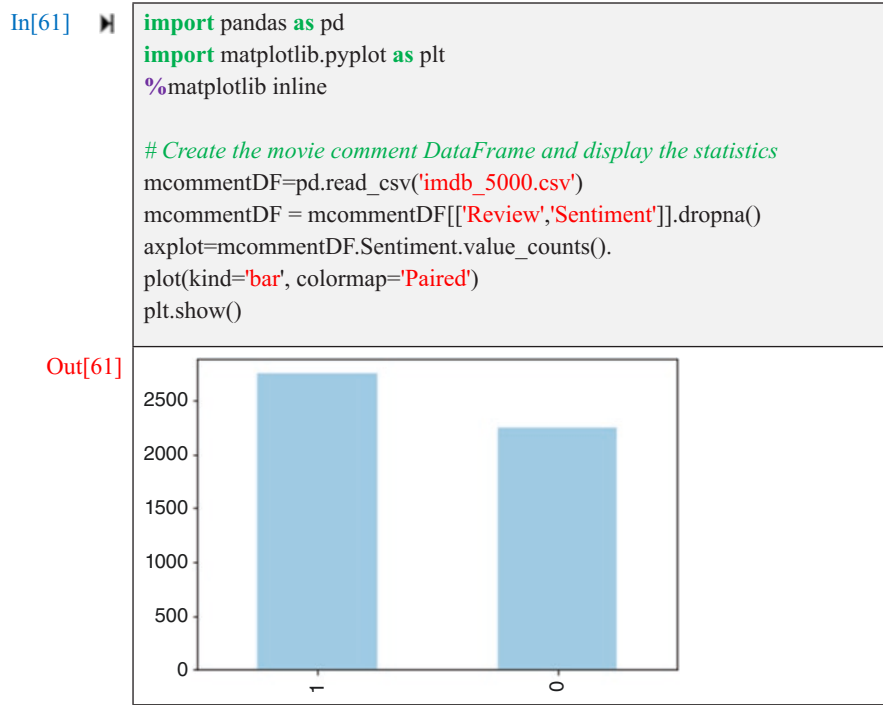
- This section will demonstrate the design of LSTM-based RNN text classifier for sentiment analysis with steps below:
1. Data retrieve and preprocessing.
  2. Tokenize review utterances with padding.
  3. Create utterances pad sequence and put into Input Layer.
  4. Vectorize each token and verified by token-ID in Embedding Layer.
  5. Input token vectors into LSTM.
  6. Train LSTM network.

Let us start by recalling the dataset again.

14.11.1 Step 1: Dataset

IMDB movie reviews identical dataset from sentiment analysis with spaCy section will be used. They had already processed with *pandas* and condensed into two columns with binary labels.

Reload reviews table and perform data preprocessing as done in previous section to ensure the data is up to date:



Here is how *mcommentDF* dataset should look:

In[62]

mcommentDF.head()

Out[62]

	Review	Sentiment
0	<b>**Possible Spoilers**</b>	0
1	Read the book, forget the movie!	0
2	<b>**Possible Spoilers Ahead**</b>	0
3	What a script, what a story, what a mess!	0
4	I hope this group of film-makers never re-unites.	0

Next, extract review text and review label from each dataset row and add them into Python lists:

```
In[63] ▶ # Import spaCy
import spacy

# Load the spaCy NLP model
nlp = spacy.load("en_core_web_md")
```

```
In[64] ▶ # Create movie comment sample and categories objects
movie_comment_exp = []
categories = []

# Perform Tokenization
for idx, rw in mcommentDF.iterrows():
    comments = rw["Review"]
    rating = rw["Sentiment"]
    categories.append(rating)
    mtoks = [token.text for token in nlp(comments)]
    movie_comment_exp.append(mtoks)
```

```
In[65] ▶ movie_comment_exp[0]
Out[65] ['*', '*', 'Possible', 'Spoilers', '*', '*']
```



Note that a list of words to `movie_comment_exp` has added, hence each element of this list is a list of tokens. Next, invoke Keras' Tokenizer on this tokens list to build vocabulary

### 14.11.2 Step 2: Data and Vocabulary Preparation

Since the dataset had already processed, tokenize dataset sentences and build a vocabulary.

1. Import necessary Python packages.

```
In[66] ▶ # Import Tokenizer, pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
```



2. Feed *ktoken* into token list and convert them into IDs by calling *texts\_to\_sequences*:

```
In[67] ▶ # Create ktoken and perform tokenization
ktoken = Tokenizer(lower=True)
ktoken.fit_on_texts(movie_comment_exp)

# Create utterance sequences object
seq_utterance = ktoken.texts_to_sequences(movie_comment_exp)
```

3. Pad short utterance sequences to a maximum length of 50. This will truncate long reviews to 50 words:

```
In[68] ▶ # Set the max length to 50
MLEN = 50

# Create pad utterance sequence object
ps_utterance = pad_sequences(seq_utterance, MLEN, padding="post")
```

4. Convert this list of reviews and labels to numpy arrays:

```
In[69] ▶ # Convert the ps_utterance into numpy arrays
ps_utterance = np.array(ps_utterance)

# Create the category list (catlist)
catlist = np.array(categories)
```

```
In[70] ▶ catlist = catlist.reshape(catlist.shape[0] , 1)
```

```
In[71] ▶ catlist.shape
```

```
Out[71] (5000, 1)
```

All basic preparation works are completed at present to create a LSTM network and input data.

Load TensorFlow Keras related modules:

```
In[72] ▶ # Import the LSTM model and the optimizers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense, Embedding
from tensorflow.keras import optimizers
```

### 14.11.3 Step 3: Implement the Input Layer

In[73] ▶ `utterance_input = Input(shape=(None,))`



Do not confuse *None* as input shape. Here, *None* means that dimension can be any scalar number. So, use this expression when Keras infers the input shape

### 14.11.4 Step 4: Implement the Embedding Layer

Create an Embedding Layer as follows:

In[74] ▶ 

```
# Create the Embedding_Layer
embedding = Embedding(input_dim = len(ktoken.word_index)+1,
                      output_dim = 100)(utterance_input)
```



1. When defining embedding layer, input dimension should always be tokens number in the vocabulary (+1 because the indices start from 1 and not 0. Index 0 is reserved for padding value)
2. Here, 100 is selected as the output shape, hence token vectors for vocabulary tokens will be 100-dimensional. Popular numbers for token vector dimensions are 50, 100, and 200 depending on task complexity

### 14.11.5 Step 5: Implement the LSTM Layer

Create LSTM\_Layer:

In[75] ▶ 

```
# Create the LSTM_Layer
LSTM_layer = LSTM(units=256)(embedding)
```




Here, *units* = 256 is the dimension of hidden state. LSTM output shape and hidden state shape are identical due to LSTM architecture.

14.11.6 Step 6: Implement the Output Layer

When a 256-dimensional vector from LSTM layer has obtained, it will be condensed into a 1-dimensional vector (possible values of this vector are 0 and 1, which are class labels):

In[76] ▶

# Create the Output Layer  
outlayer = Dense(1, activation='sigmoid')(LSTM\_layer)



A sigmoid function is an S-shaped function used as an activation function to map its input to a [0-1] range in output layer. It is commonly used in many neural networks

14.11.7 Step 7: System Compilation

After the model has defined, it required to compile with an optimizer, a loss function, and an evaluation metric:

In[77] ▶

# Create the IMDB User Review LSTM Model (imdb\_md1)  
imdb\_md1 = Model(inputs=[utterance\_input],outputs=[outlayer])

Let's look at an *imdb\_md1* model setup:

In[78] ▶

imdb\_md1.summary()

Out[78]

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, None)]	0
embedding (Embedding)	(None, None, 100)	1874400
lstm (LSTM)	(None, 256)	365568
dense (Dense)	(None, 1)	257

Total params: 2, 240, 225  
Trainable params: 2, 240, 225  
Non-trainable params: 0

Next, invoke model compilation:

```
In[79] ► imdb_md1.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
```



1. Use ADAM (adaptive moment estimation) as optimizer for LSTM training for `imdb_md1` LSTM model
2. Use binary cross-entropy as loss function
3. A list of supported performance metrics can be found in Keras official site (Keras 2022)

### 14.11.8 Step 8: Model Fitting and Experiment Evaluation

Feed `imdb_md1` model to data with 5 epochs to reduce time:

```
In[80] ► # Model fitting by using 5 epochs
imdb_md1.fit(x=ps_utterance,
             y=catlist,
             batch_size=64,
             epochs=5,
             validation_split=0.3)
```

```
Out[80] Epoch 1/5
55/55 [=====] - 30s 528ms/step - loss: 0.6062 - accuracy: 0.6591 - val_
loss: 0.5078 - val_accuracy: 0.7427
Epoch 2/5
55/55 [=====] - 28s 501ms/step - loss: 0.2577 - accuracy: 0.9054 - val_
loss: 0.4609 - val_accuracy: 0.7893
Epoch 3/5
55/55 [=====] - 26s 465ms/step - loss: 0.0940 - accuracy: 0.9737 - val_
loss: 0.8353 - val_accuracy: 0.7580
Epoch 4/5
55/55 [=====] - 24s 432ms/step - loss: 0.0357 - accuracy: 0.9911 - val_
loss: 0.9209 - val_accuracy: 0.7633
Epoch 5/5
55/55 [=====] - 24s 438ms/step - loss: 0.0314 - accuracy: 0.9917 - val_
loss: 0.5480 - val_accuracy: 0.7793

<keras.callbacks.History at 0x26289a53f10>
```



1.  $x$  is a list of *ps\_utterance* for network training and  $y$  is the list of categories (catlist). The epochs parameter is set to 5 to process 5 passes over the data
2. Data has processed 5 times in parameter *batch\_size* = 64 means a batch of 64 training utterances are fed into the memory at a time due to memory limitations
3. The *validation\_split* = 0.3 means 70% of dataset are used for training and 30% are used for system validation
4. An experiment validation accuracy rate 0.7793 is acceptable for a basic LSTM network training for 5 epochs only



### Workshop 5.2 Further Exploration of LSTM model on Movie Sentiment Analysis

1. Follow Workshop 14.1 logic and use rating (0–10) field of IMDB movie reviews dataset to reconstruct a LSTM for sentiment analysis into 3 categories: Positive, Neutral and Negative
2. Verify training performance
3. Experiment with the code further by placing dropout layers at different locations such as after embedding layer or, after LSTM layer
4. Try different values for embedding dimensions such as 50, 150, and 200 to observe change in accuracy
5. Experiment with different values instead of 256 at LSTM layer's hidden dimension. Try different parameters for each to perform simulations and see whether the best configuration can be found

## References

- Agarwal, B. (2020) Deep Learning-Based Approaches for Sentiment Analysis (Algorithms for Intelligent Systems). Springer.
- Albrecht, J., Ramachandran, S. and Winkler, C. (2020) Blueprints for Text Analytics Using Python: Machine Learning-Based Solutions for Common Real World (NLP) Applications. O'Reilly Media.
- Altinok, D. (2021) Mastering spaCy: An end-to-end practical guide to implementing NLP applications using the Python ecosystem. Packt Publishing.
- Arumugam, R., & Shanmugamani, R. (2018). Hands-on natural language processing with python. Packt Publishing.
- Bird, S., Klein, E., and Loper, E. (2009). Natural language processing with python. O'Reilly.
- Ekman, M. (2021) Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow. Addison-Wesley Professional.
- George, A. (2022) Python Text Mining: Perform Text Processing, Word Embedding, Text Classification and Machine Translation. BPB Publications.
- Géron, A. (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.
- Kaggle (2022) IMDB Large Movie Review Dataset from Kaggle. <https://www.kaggle.com/code/nisargchodavadiya/movie-review-analytics-sentiment-ratings/data>. Accessed 23 June 2022.

- Kedia, A. and Rasu, M. (2020) Hands-On Python Natural Language Processing: Explore tools and techniques to analyze and process text with a view to building real-world NLP applications. Packt Publishing.
- Keras (2022) Keras official site performance metrics. <https://keras.io/api/metrics>. Accessed 23 June 2022.
- Korstanje, J. (2021) Advanced Forecasting with Python: With State-of-the-Art-Models Including LSTMs, Facebook's Prophet, and Amazon's DeepAR. Apress.
- Perkins, J. (2014). Python 3 text processing with NLTK 3 cookbook. Packt Publishing Ltd.
- Pozzi, F., Fersini, E., Messina, E. and Liu, B. (2016) Sentiment Analysis in Social Networks. Morgan Kaufmann.
- SpaCy (2022) spaCy official site. <https://spacy.io/>. Accessed 16 June 2022.
- Sarkar, D. (2019) Text Analytics with Python: A Practitioner's Guide to Natural Language Processing. Apress.
- Siahaan, V. and Sianipar, R. H. (2022) Text Processing and Sentiment Analysis using Machine Learning and Deep Learning with Python GUI. Balige Publishing.
- Srinivasa-Desikan, B. (2018). Natural language processing and computational linguistics: A practical guide to text analysis with python, gensim, SpaCy, and keras. Packt Publishing Limited.
- TensorFlow (2022) TensorFlow official site. <https://tensorflow.org/>. Accessed 22 June 2022.
- Vasiliev, Y. (2020) Natural Language Processing with Python and spaCy: A Practical Introduction. No Starch Press.