Gurjus Singh

MSDS 458 – Deep Learning and AI

February 21st, 2021

Week 7: A.3 Third Research/Programming Assignment

Abstract

In this research, I explored the AG News Topics Dataset which consisted of 127,600 news headlines divided into 4 news categories World, Science/Tech, Business and Sports. The goal of this research was to use 1D CNN, RNN and LSTM architectures to do supervised learning multiclass classification on the news dataset. This involved heavy amount of preprocessing and encoding of the words in the articles. In the research, I did a total of 23 experiments.

Introduction

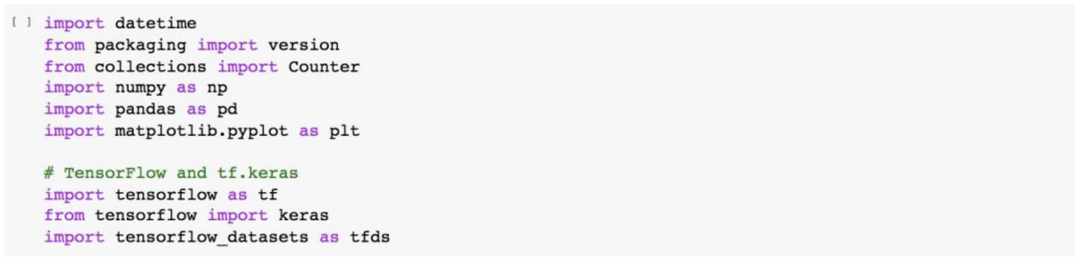
This research involved exploring a topic of Deep Learning which is Natural Language Processing, and how to find patterns in the words to classify. The dataset I was trying to classify on was the News dataset from Reuters which involved supervised learning multiclass classification on four categories which included World, Science/Tech, Business and Sports. I tried to play around with different architectures to observe different accuracy results. The architectures included 1D CNN, RNN and LSTM architectures. I also noticed preprocessing made a huge difference in the results.

Literature Review

I decided to read two papers on the topic of NLP and news text classification. In the first paper surrounding news classification, it went over techniques surrounding news classification [1]. Some things I took away from this paper was that classification involves steps such as collection of the news from a variety of different sources [1]. Then the preprocessing happens which involves tokenization [1]. In the process of tokenization, the words are broken up and treated as strings [1]. After tokenization, stop words removal happens which removes words that do not carry and information [1]. Stemming also happens in preprocessing which reduces a word to its root [1]. Feature selection also happens during preprocessing [1]. Once preprocessing happens, then it is time to classify the news [1]. The paper mentions several techniques for classification which are to use Naïve Bayes, Support Vector Machines, Artificial Neural Networks, Decision Trees, or K-Nearest Neighbors [1]. In the second article, it compared several modeling techniques to each other using the AG News Dataset [2]. It talked about using bag of words, and ngrams in the preprocessing stages. The results showed that Bag of Words with CNN performed with 88.76 percent accuracy [2].

Methods

The first step in the research involved importing the packages seen in 1-1.

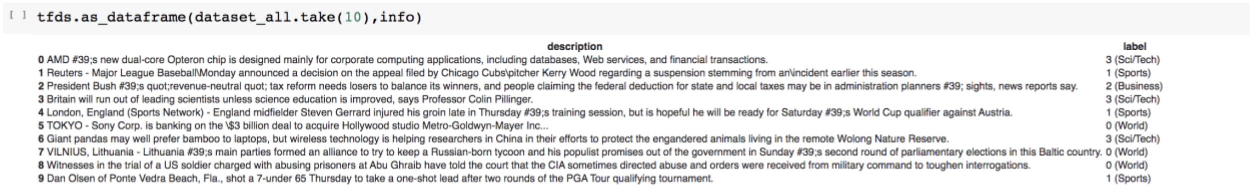


```
[ ] import datetime
    from packaging import version
    from collections import Counter
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt

    # TensorFlow and tf.keras
    import tensorflow as tf
    from tensorflow import keras
    import tensorflow_datasets as tfds
```

Import Packages 1-1

Several Packages were helpful in the research which was the Keras and Tensorflow packages. These were important to make the architecture for RNNs, LSTMs, and CNNs. I also checked the Tensorflow and Keras version to make sure they were both current. Next, I downloaded the AG News Dataset using the **tfds.load** function. What I observed after downloading is that there were 127,600 examples in the dataset. I then converted the dataset to a dataframe using **tfds.as_dataframe** to see the first 10 examples as shown in 1-2.



```
[ ] tfds.as_dataframe(dataset_all.take(10),info)
```

	description	label
0	AMD #39;s new dual-core Opteron chip is designed mainly for corporate computing applications, including databases, Web services, and financial transactions.	3 (Sci/Tech)
1	Reuters - Major League Baseball\Monday announced a decision on the appeal filed by Chicago Cubs\pitcher Kerry Wood regarding a suspension stemming from an\incident earlier this season.	1 (Sports)
2	President Bush #39;s quot;revenue-neutral quot; tax reform needs losers to balance its winners, and people claiming the federal deduction for state and local taxes may be in administration planners #39; sights, news reports say.	2 (Business)
3	Britain will run out of leading scientists unless science education is improved, says Professor Colin Pillinger.	3 (Sci/Tech)
4	London, England (Sports Network) - England midfielder Steven Gerrard injured his groin late in Thursday #39;s training session, but is hopeful he will be ready for Saturday #39;s World Cup qualifier against Austria.	1 (Sports)
5	TOKYO - Sony Corp. is banking on the \$3 billion deal to acquire Hollywood studio Metro-Goldwyn-Mayer Inc...	0 (World)
6	Giant pandas may well prefer bamboo to laptops, but wireless technology is helping researchers in China in their efforts to protect the engandered animals living in the remote Wolong Nature Reserve.	3 (Sci/Tech)
7	VILNIUS, Lithuania - Lithuania #39;s main parties formed an alliance to try to keep a Russian-born tycoon and his populist promises out of the government in Sunday #39;s second round of parliamentary elections in this Baltic country.	0 (World)
8	Witnesses in the trial of a US soldier charged with abusing prisoners at Abu Ghraib have told the court that the CIA sometimes directed abuse and orders were received from military command to toughen interrogations.	0 (World)
9	Dan Olsen of Ponte Vedra Beach, Fla., shot a 7-under 65 Thursday to take a one-shot lead after two rounds of the PGA Tour qualifying tournament.	1 (Sports)

First 10 Examples 1-2

I also got information about the labels, and noticed there were 4 categories mainly World, Sports, Business, and Science/Tech. After observing the data and labels, I then split up the datasets into Train, Validation and Test.

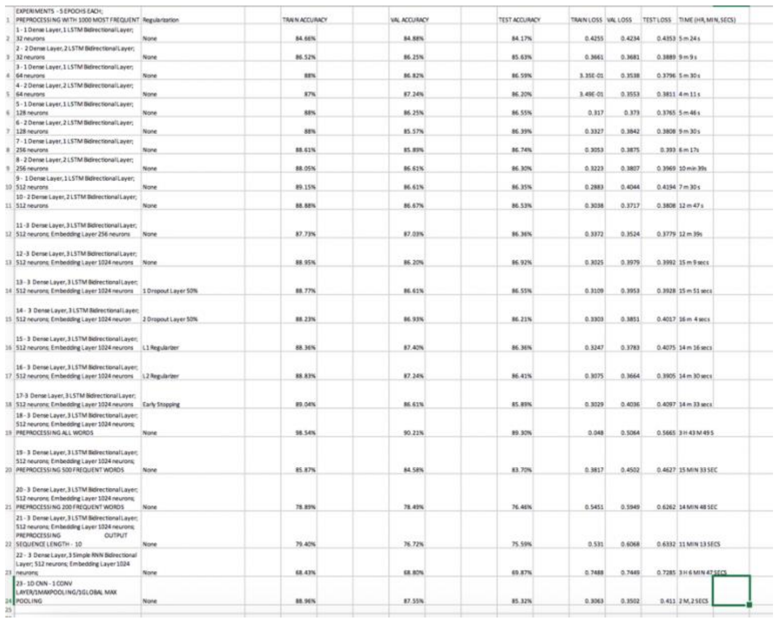
Next was the preprocessing part. The preprocessing T**extVectorization** was important for this as it gave me a way for the number of words we could use in our corpus. I first limited the dataset to the 1000 frequent words, which I would later change in the experiments. After limiting it to 1000 most words, the encoder was used to represent the words as numbers which then could be inputted into the neural network. After preprocessing, I then created my architectures. I tried out different number of layers such as dense, bidirectional , LSTM, and RNN layers. I also tried out using different neural networks such as switching between RNNs, CNNs, to LSTMs. I also played around with the **TextVectorization** function as explained above shrinking the vocabulary or increasing it from most frequent words.

In this research, there was a layer that was not used in previous research that I have done which was the embedding layer. The embedding layer is used to input 2D tensor of words mapped to integers [3]. The embedding layer functions in that it learns the relationships between the words and puts represents it in a geometric space [3].

Results

In this research, I did a total **of 23 experiments, 5 Epochs each**. I played around with the preprocessing, layering and regularization parameters. My goal was to get the Test Accuracy up 90 percent, but according to my results in 1-3, I could not obtain 90 percent and the closest I got was 89 percent. From the results in 1-3, it suggests that the best model was Experiment 12 which has a Training Accuracy of 88.92 percent and a Testing Accuracy of 86.92 percent. For Experiment 12, my architecture involved an embedding layer with 1024 neurons, three Dense/Bidirectional LSTM Layers with 512 neurons each. The reason I did not chose the model with Testing Accuracy of 89.30 percent because it was highly overfitting with a Training Accuracy of 98.54 percent.

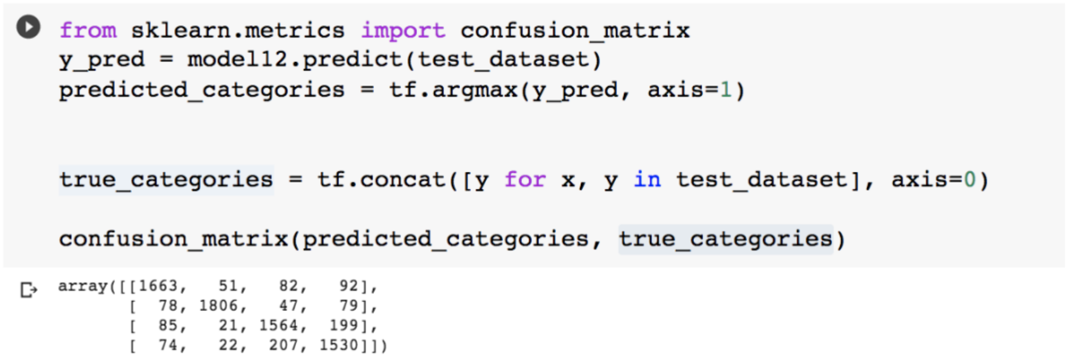
In my experiments I also tried shrinking/increasing the number of frequent words used when I reduced most frequent words it started underfitting, but when I increased the number of most frequent words, the accuracy increased and started overfitting, which led to the experiment with 98 percent accuracy. I also tried one 1D-CNN model and my accuracy was 88.96 percent for Train Accuracy and 85.32 percent for Test Accuracy.



EXPERIMENTS - 5 EPOCHS EACH; PREPROCESSING WITH 1000 MOST FREQUENT	Regularization	TRAIN ACCURACY	VAL ACCURACY	TEST ACCURACY	TRAIN LOSS	VAL LOSS	TEST LOSS	TIME (HR, MIN, SECS)
1 - 1 Dense Layer, 1 LSTM Bidirectional Layer; 32 neurons	None	84.66%	84.88%	84.17%	0.4255	0.4234	0.4353	5 m 24 s
2 - 2 Dense Layer, 2 LSTM Bidirectional Layer; 32 neurons	None	86.52%	86.25%	85.63%	0.3661	0.3681	0.3889	9 m 9 s
3 - 1 Dense Layer, 1 LSTM Bidirectional Layer; 64 neurons	None	88%	86.82%	86.59%	3.35E-01	0.3538	0.3796	5 m 30 s
4 - 2 Dense Layer, 2 LSTM Bidirectional Layer; 64 neurons	None	87%	87.24%	86.20%	3.49E-01	0.3553	0.3811	4 m 11 s
5 - 1 Dense Layer, 1 LSTM Bidirectional Layer; 128 neurons	None	88%	86.25%	86.55%	0.317	0.373	0.3765	5 m 46 s
6 - 2 Dense Layer, 2 LSTM Bidirectional Layer; 128 neurons	None	88%	85.57%	86.39%	0.3327	0.3842	0.3808	9 m 30 s
7 - 1 Dense Layer, 1 LSTM Bidirectional Layer; 256 neurons	None	88.61%	85.89%	86.74%	0.3053	0.3875	0.393	6 m 17s
8 - 2 Dense Layer, 2 LSTM Bidirectional Layer; 256 neurons	None	88.05%	86.61%	86.30%	0.3223	0.3807	0.3969	10 min 39s
9 - 1 Dense Layer, 1 LSTM Bidirectional Layer; 512 neurons	None	89.15%	86.61%	86.35%	0.2883	0.4044	0.4194	7 m 30 s
10 - 2 Dense Layer, 2 LSTM Bidirectional Layer; 512 neurons	None	88.88%	86.67%	86.53%	0.3038	0.3717	0.3808	12 m 47 s
11 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 256 neurons	None	87.73%	87.03%	86.36%	0.3372	0.3524	0.3779	12 m 39s
12 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons	None	88.95%	86.20%	86.92%	0.3025	0.3979	0.3992	15 m 9 secs
13 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons	1 Dropout Layer 50%	88.77%	86.61%	86.55%	0.3109	0.3953	0.3928	15 m 51 secs
14 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neuron	2 Dropout Layer 50%	88.23%	86.93%	86.21%	0.3303	0.3851	0.4017	16 m 4 secs
15 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons	L1 Regularizer	88.36%	87.40%	86.36%	0.3247	0.3783	0.4075	14 m 16 secs
16 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons	L2 Regularizer	88.83%	87.24%	86.41%	0.3075	0.3664	0.3905	14 m 30 secs
17-3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons	Early Stopping	89.04%	86.61%	85.89%	0.3029	0.4036	0.4097	14 m 33 secs
18 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons; PREPROCESSING ALL WORDS	None	98.54%	90.21%	89.30%	0.048	0.5064	0.5665	3 H 43 M 49 S
19 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons; PREPROCESSING 500 FREQUENT WORDS	None	85.87%	84.58%	83.70%	0.3817	0.4502	0.4627	15 MIN 33 SEC
20 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons; PREPROCESSING 200 FREQUENT WORDS	None	78.89%	78.49%	76.46%	0.5451	0.5949	0.6262	14 MIN 48 SEC
21 - 3 Dense Layer, 3 LSTM Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons; PREPROCESSING OUTPUT SEQUENCE LENGTH - 10	None	79.40%	76.72%	75.59%	0.531	0.6068	0.6332	11 MIN 13 SECS
22 - 3 Dense Layer, 3 Simple RNN Bidirectional Layer; 512 neurons; Embedding Layer 1024 neurons;	None	68.43%	68.80%	69.87%	0.7488	0.7449	0.7285	3 H 6 MIN 47 SECS
23 - 1D CNN - 1 CONV LAYER/1MAXPOOLING/1GLOBAL MAX POOLING	None	88.96%	87.55%	85.32%	0.3063	0.3502	0.411	2 M, 2 SECS

Experiments and Results 1-3

After examining each model and choosing Experiment 12 as the best model, I then wanted to examine the Confusion Matrix for Test Dataset as shown in 1-4. I also examined the precision and recall score which were both around 0.86 as seen in 1-5. This showed the model was doing very well in its results.



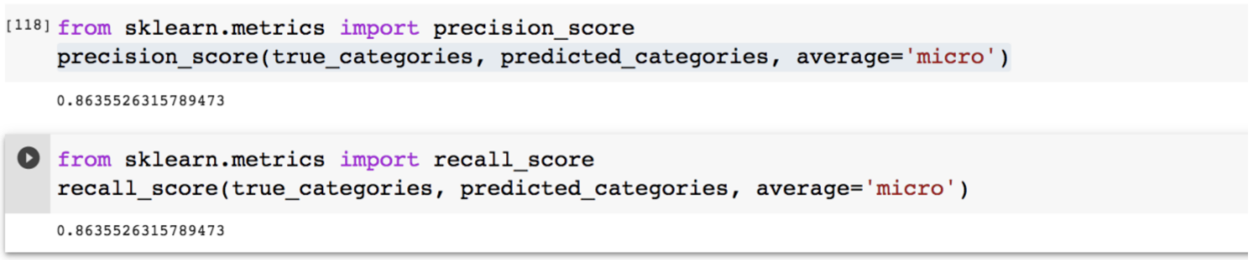
```
from sklearn.metrics import confusion_matrix
y_pred = model12.predict(test_dataset)
predicted_categories = tf.argmax(y_pred, axis=1)


true_categories = tf.concat([y for x, y in test_dataset], axis=0)

confusion_matrix(predicted_categories, true_categories)
```

```
array([[1663,   51,   82,   92],
       [  78, 1806,   47,   79],
       [  85,   21, 1564,  199],
       [  74,   22,  207, 1530]])
```

Confusion Matrix Experiment 12 Test Dataset 1-4



```
[118] from sklearn.metrics import precision_score
      precision_score(true_categories, predicted_categories, average='micro')
```

```
0.8635526315789473
```

```
from sklearn.metrics import recall_score
recall_score(true_categories, predicted_categories, average='micro')
```

```
0.8635526315789473
```

Precision and Recall Score Test Dataset 1-5

Conclusion

This research involved examining the AG News Dataset with 127,600 examples. The purpose of the research was to do a Supervised Learning Multiclass Classification on 4 category labels mainly World, Business, Sports and Science/Tech. After exploring my 23 experiments, I decided that the best model was Experiment 12 which involved an architecture of one embedding layer with 1024 neurons, three Dense/Bidirectional LSTM Layers with 512 neurons each.

Therefore, for my management recommendation, I recommend an architecture of one

embedding layer with 1024 neurons, three Dense/Bidirectional LSTM Layers with 512 neurons each.

For further research, I hope to examine more CNN layers as I felt on the first try the CNN did particularly well. I also hope to examine more data preprocessing techniques such as removing stop words and incorporating ntlk package. I did use all the words in one experiment, but my model ended up overfitting as explained above. I can try using regularizers on this overfitted model, or early stopping.

References

[1] Kaur, G., & Bajaj, K. (2016). *News classification and its techniques: a review*. Semantic Scholar. http://www.iosrjournals.org/iosr-jce/papers/Vol18-issue1/Version-3/D018132226.pdf

[2] SreeDevi, J., Rama Bai, M., & Chandrashekar Reddy, M. (2020, March 5). *Newspaper Article Classification using Machine Learning Techniques*. International Journal of Innovative Technology and Exploring Engineering. http://www.ijitee.org/wp-content/uploads/papers/v9i5/E2753039520.pdf

[3] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Company.

Appendix

Northwestern

MSDS458 Research Assignment 3

Analyze AG_NEWS_SUBSET Data

AG is a collection of more than 1 million news articles. News articles have been gathered from more than 2000 news sources by ComeToMyHead in more than 1 year of activity. ComeToMyHead is an academic news search engine which has been running since July, 2004. The dataset is provided by the academic comunity for research purposes in data mining (clustering, classification, etc), information retrieval (ranking, search, etc), xml, data compression, data streaming, and any other non-commercial activity.

For more information, please refer to the link http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

The AG's news topic classification dataset is constructed by choosing 4 largest classes (**World**, **Sports**, **Business**, and **Sci/Tech**) from the original corpus. Each class contains 30,000 training samples and 1,900 testing samples. The total number of training samples is 120,000 and testing 7,600.

Homepage: https://arxiv.org/abs/1509.01626

Source code: tfds.text.AGNewsSubset

Versions:

1.0.0 (default): No release notes. Download size: 11.24 MiB

Dataset size: 35.79 MiB

More Technical: Throughout the notebook. This types of boxes provide more technical details and extra references about what you are seeing. They contain helpful tips, but you can safely skip them the first time you run through the code.

Import packages

```
import datetime
from packaging import version
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds

%matplotlib inline
np.set_printoptions(precision=3, suppress=True)
```

Create a Helper Function to Plot Graphs:

```
def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
```

Verify TensorFlow Version and Keras Version

```
print("This notebook requires TensorFlow 2.0 or above")
print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >=2
```

```
    This notebook requires TensorFlow 2.0 or above
    TensorFlow version:  2.4.1
```

```
print("Keras version: ", keras.__version__)
```

```
    Keras version:  2.4.0
```

Suppress warning messages

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

Mount Google Drive to Colab Environment

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
    Mounted at /content/gdrive
```

TensorFlow Datasets Information

ag_news_subset

See https://www.tensorflow.org/datasets/catalog/ag_news_subset

Get all the words in the documents (as well as the number of words in each document) by using the encoder to get the indices associated with each token and then translating the indices to tokens. But first we need to get the "unpadded" new articles so that we can get their length.

```
#register  ag_news_subset so that tfds.load doesn't generate a che
!python -m tensorflow_datasets.scripts.download_and_prepare --regi

# https://www.tensorflow.org/datasets/splits
# The full `train` and `test` splits, interleaved together.
ri = tfds.core.ReadInstruction('train') + tfds.core.ReadInstructio
dataset_all, info = tfds.load('ag_news_subset', with_info=True,  s
```

```
    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 2 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 3 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 4 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 5 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 6 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 7 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 8 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 9 MiB [00:05,  5.49s/ MiB]

    Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
    Dl Size...: 10 MiB [00:05,  5.49s/ MiB]
```

```
Dl Completed...:   0% 0/1 [00:05<?, ? url/s]
Dl Size...: 11 MiB [00:05,  5.49s/ MiB]

Dl Completed...: 100% 1/1 [00:05<00:00,  5.59s/ url]
Dl Size...: 11 MiB [00:05,  5.49s/ MiB]

Dl Completed...: 100% 1/1 [00:05<00:00,  5.59s/ url]
Dl Size...: 11 MiB [00:05,  5.49s/ MiB]

Extraction completed...:   0% 0/1 [00:05<?, ? file/s]

Dl Completed...: 100% 1/1 [00:05<00:00,  5.59s/ url]
Dl Size...: 11 MiB [00:05,  5.49s/ MiB]

Extraction completed...: 100% 1/1 [00:05<00:00,  5.96s/ file]
Extraction completed...: 100% 1/1 [00:05<00:00,  5.96s/ file]

Dl Size...: 11 MiB [00:05,  1.84 MiB/s]

Dl Completed...: 100% 1/1 [00:05<00:00,  5.96s/ url]
I0220 19:51:41.106563 140320970397568 dataset_builder.py:970] Generating split train
Shuffling and writing examples to /root/tensorflow_datasets/ag_news_subset/1.0.0.incomple
 69% 82535/120000 [00:00<34:05, 18.32 examples/s]  I0220 19:52:16.678695 140320970397568
I0220 19:52:16.695761 140320970397568 dataset_builder.py:970] Generating split test
Shuffling and writing examples to /root/tensorflow_datasets/ag_news_subset/1.0.0.incomple
  0% 0/7600 [00:00<?, ? examples/s]I0220 19:52:18.963265 140320970397568 tfrecords_writer
I0220 19:52:18.964528 140320970397568 dataset_builder.py:412] Skipping computing stats fo
Dataset ag_news_subset downloaded and prepared to /root/tensorflow_datasets/ag_news_subse
name: "ag_news_subset"
description: "AG is a collection of more than 1 million news articles.\nNews articles hav
citation: "@misc{zhang2015characterlevel,\n    title={Character-level Convolutional Netwo
location {
  urls: "https://arxiv.org/abs/1509.01626"
```

Exploratory Analysis AG News Subset

Get information about the ag_news_subset dataset. We combined the training and test data for a total of 127,600 news articles.

info

```
tfds.core.DatasetInfo(
    name='ag_news_subset',
    version=1.0.0,
    description='AG is a collection of more than 1 million news articles.
News articles have been gathered from more than 2000  news sources by ComeToMyHead in mor
ComeToMyHead is an academic news search engine which has been running since July, 2004.
The dataset is provided by the academic comunity for research purposes in data mining (cl
information retrieval (ranking, search, etc), xml, data compression, data streaming,
and any other non-commercial activity.
For more information, please refer to the link http://www.di.unipi.it/~gulli/AG_corpus_of

The AG's news topic classification dataset is constructed by Xiang Zhang (xiang.zhang@nyu
It is used as a text classification benchmark in the following paper:
Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Clas

The AG's news topic classification dataset is constructed by choosing 4 largest classes f
Each class contains 30,000 training samples and 1,900 testing samples.
The total number of training samples is 120,000 and testing 7,600.',
    homepage='https://arxiv.org/abs/1509.01626',
    features=FeaturesDict({
        'description': Text(shape=(), dtype=tf.string),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=4),
        'title': Text(shape=(), dtype=tf.string),
    }),
    total_num_examples=127600,
    splits={
        'test': 7600,
        'train': 120000,
    },
    supervised_keys=('description', 'label'),
    citation="""@misc{zhang2015characterlevel,
        title={Character-level Convolutional Networks for Text Classification},
        author={Xiang Zhang and Junbo Zhao and Yann LeCun},
        year={2015},
        eprint={1509.01626},
        archivePrefix={arXiv},
        primaryClass={cs.LG}
    }""",
    redistribution_info=,
)
```

```
tfds.as_dataframe(dataset_all.take(10),info)
```

	description	label
0	AMD #39;s new dual-core Opteron chip is designed mainly for corporate computing applications, including databases, Web services, and financial transactions.	3 (Sci/Tech)
1	Reuters - Major League Baseball\Monday announced a decision on the appeal filed by Chicago Cubs\pitcher Kerry Wood regarding a suspension stemming from an\incident earlier this season.	1 (Sports)
2	President Bush #39;s quot;revenue-neutral quot; tax reform needs losers to balance its winners, and people claiming the federal deduction for state and local taxes may be in administration planners #39; sights, news reports say.	2 (Business)
3	Britain will run out of leading scientists unless science education is improved, says Professor Colin Pillinger.	3 (Sci/Tech)
	London, England (Sports Network) - England midfielder Steven Gerrard injured his groin late in Thursday	

Review Labels (Categories)

```
print(f'There are {info.features["label"].num_classes} classes in
print(f'The class names are {info.features["label"].names}')
```

```
There are 4 classes in the dataset.
The class names are ['World', 'Sports', 'Business', 'Sci/Tech']
```

```
# classes dictionary
categories =dict(enumerate(info.features["label"].names))
categories
```

```
{0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}
```

The 127,600 news articles are evenly distributed among the 4 categories

```
train_categories = [categories[label] for label in dataset_all.map
Counter(train_categories).most_common()
```

```
[('Sci/Tech', 31900), ('Sports', 31900), ('Business', 31900), ('World', 31900)]
```

We will use the tf.keras.layers.experimental.preprocessing.TextVectorization layer to transform each news article into a "list" of non-negative integers representing the tokens in the news article.

For the purpose of training our models each such "encoding" will have a fixed length corresponding to the news article(s) with the most tokens. Shorter articles will be right-padded with zeros in the encoding. Also to speed up the training process, we will set max_tokens = 1000 so that words not in the vabulary set of top 1000 most common tokes are encoded as 1. But first we set max_tokens = None (which is the default value) in order to get the vocabulary size of the corpus.

```
%%time
encoder = tf.keras.layers.experimental.preprocessing.TextVectoriza
encoder.adapt(dataset_all.map(lambda text, label: text))
vocab = np.array(encoder.get_vocabulary())
```

```
WARNING:tensorflow:AutoGraph could not transform <function <lambda> at 0x7f01901cdae8> an
Cause: could not parse the source code of <function <lambda> at 0x7f01901cdae8>: no match
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_con
WARNING:tensorflow:AutoGraph could not transform <function <lambda> at 0x7f01901cdae8> an
Cause: could not parse the source code of <function <lambda> at 0x7f01901cdae8>: no match
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_con
WARNING: AutoGraph could not transform <function <lambda> at 0x7f01901cdae8> and will run
Cause: could not parse the source code of <function <lambda> at 0x7f01901cdae8>: no match
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_con
CPU times: user 2min 4s, sys: 26.8 s, total: 2min 31s
Wall time: 1min 34s
```

```
print(f"There are {len(vocab)} vocabulary words in the corpus.")
```

```
There are 95976 vocabulary words in the corpus.
```

There are 95976 vocabulary words in the corpus.

The .adapt method sets the layer's vocabulary. Here are the first 20 tokens. After the padding and unknown tokens they're sorted by frequency:

```
vocab[:20]
```

```
array(['', '[UNK]', 'the', 'a', 'to', 'of', 'in', 'and', 'on', 'for',
 'that', '39s', 'with', 'its', 'as', 'at', 'is', 'said', 'by', 'it'],
dtype='<U150')
```

Let's use how the encoding works on a sample string all of whose words are in the vocabulary of the corpus

```
example = "the dog ran after a red ball as it rolled by the hat on
for word in example.split():
  print(f'"{word}" is {"*not* " if word not in vocab  else ""}in t
```

```
"the" is in the vocabulary.
"dog" is in the vocabulary.
"ran" is in the vocabulary.
"after" is in the vocabulary.
"a" is in the vocabulary.
"red" is in the vocabulary.
"ball" is in the vocabulary.
"as" is in the vocabulary.
"it" is in the vocabulary.
"rolled" is in the vocabulary.
"by" is in the vocabulary.
"the" is in the vocabulary.
"hat" is in the vocabulary.
"on" is in the vocabulary.
"the" is in the vocabulary.
"ground." is *not* in the vocabulary.
```

```
encoder(example)
```

```
<tf.Tensor: shape=(16,), dtype=int64, numpy=
array([   2, 5958, 1287,   29,    3,  232, 1414,   14,   19, 2548,   18,
          2, 2435,    8,    2,  999])>
```

Let us get the total number of words in the corpus and the sizes of the news articles

```
%%time
doc_sizes = []
corpus = []
for example, _ in dataset_all.as_numpy_iterator():
  enc_example = encoder(example)
  doc_sizes.append(len(enc_example))
  corpus+=list(enc_example.numpy())
```

```
CPU times: user 13min 49s, sys: 1min 32s, total: 15min 21s
Wall time: 12min 55s
```

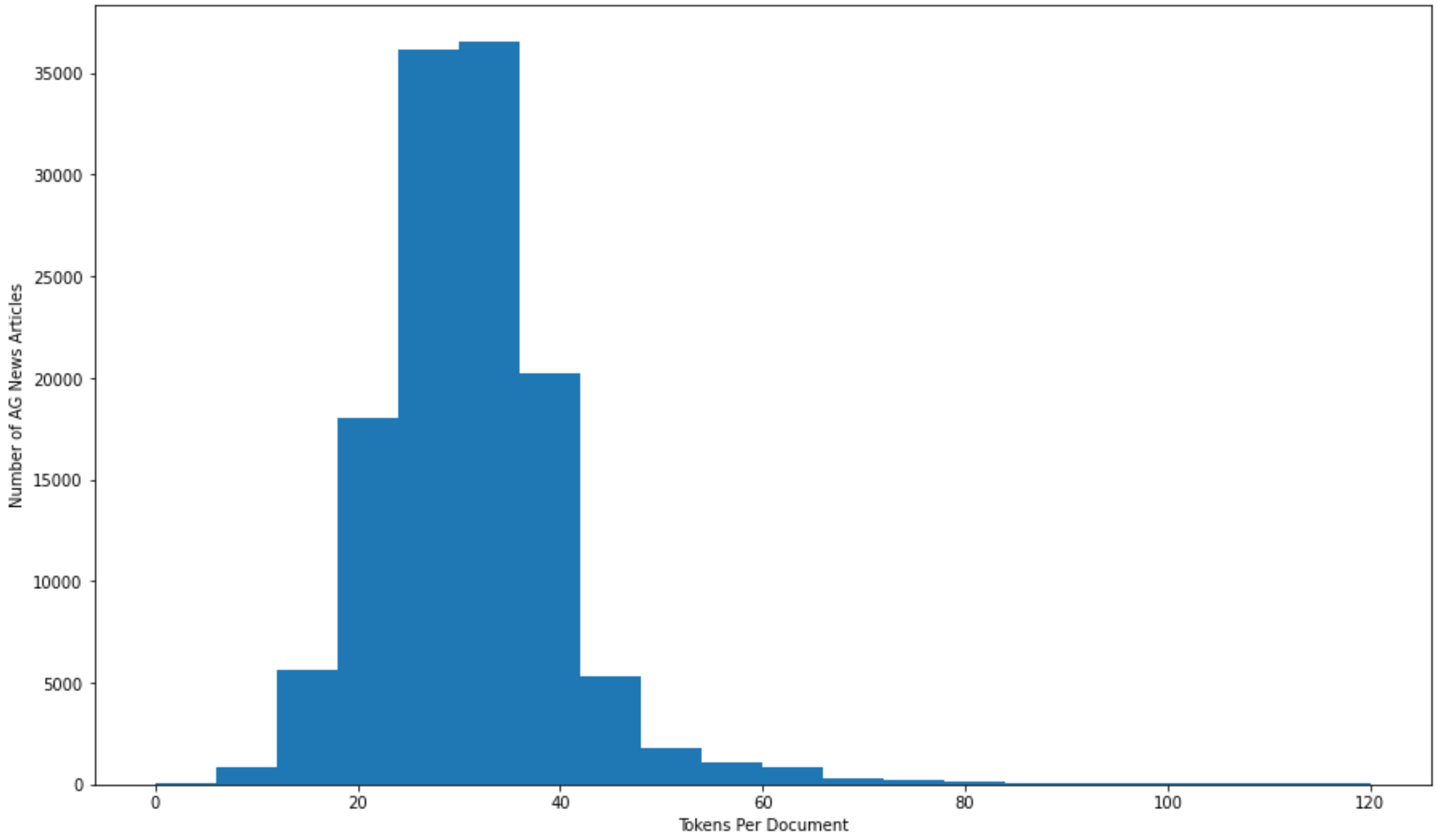
```
print(f"There are {len(corpus)} words in the corpus of {len(doc_si
print(f"Each news article has between {min(doc_sizes)} and {max(do
```

```
There are 3909695 words in the corpus of 127600 news articles.
```

```
Each news article has between 3 and 173 tokens in it.
```

```
plt.figure(figsize=(15,9))
plt.hist(doc_sizes, bins=20,range = (0,120))
plt.xlabel("Tokens Per Document")
plt.ylabel("Number of AG News Articles")
```

```
Text(0, 0.5, 'Number of AG News Articles')
```



Encode the news articles using the top 1000 most common words in the corpus

```
%%time
encoder_1000 = tf.keras.layers.experimental.preprocessing.TextVect
encoder_1000.adapt(dataset_all.map(lambda text, label: text))
vocab_1000 = np.array(encoder_1000.get_vocabulary())
```

```
WARNING:tensorflow:AutoGraph could not transform <function <lambda> at 0x7f01278e1bf8> an
Cause: could not parse the source code of <function <lambda> at 0x7f01278e1bf8>: no match
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_con
WARNING:tensorflow:AutoGraph could not transform <function <lambda> at 0x7f01278e1bf8> an
Cause: could not parse the source code of <function <lambda> at 0x7f01278e1bf8>: no match
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_con
WARNING: AutoGraph could not transform <function <lambda> at 0x7f01278e1bf8> and will run
Cause: could not parse the source code of <function <lambda> at 0x7f01278e1bf8>: no match
```

```
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_con
CPU times: user 2min 4s, sys: 26 s, total: 2min 30s
Wall time: 1min 33s
```

The .adapt method sets the layer's vocabulary. Here are the first 20 tokens. After the padding and unknown tokens they're sorted by frequency:

```
vocab_1000[:20]
```

```
array(['', '[UNK]', 'the', 'a', 'to', 'of', 'in', 'and', 'on', 'for',
       'that', '39s', 'with', 'its', 'as', 'at', 'is', 'said', 'by', 'it'],
      dtype='<U14')
```

In particular, 0 is use for padding, 1 for the unknown words, 2 for the common word, i.e. 'the', etc. Let us look at the same example we encoded previously using the encoder for all the vocabulary words. Note that there are now five 1's denoting words that are not in the top 1000 in frequency.

We encode the same example as before using the new encoder. Note that there are now 5 out of vocabulary words.

```
example = "the dog ran after a red ball as it rolled by the hat on
encoder_1000(example)
```

```
<tf.Tensor: shape=(16,), dtype=int64, numpy=
array([  2,   1,   1,  29,   3, 232,   1,  14,  19,   1,  18,   2,   1,
         8,   2, 999])>
```

```
for word in example.split():
  print(f'"{word}" is {"*not* " if word not in vocab_1000  else ""
```

```
"the" is in the vocabulary.
"dog" is *not* in the vocabulary.
"ran" is *not* in the vocabulary.
"after" is in the vocabulary.
"a" is in the vocabulary.
"red" is in the vocabulary.
"ball" is *not* in the vocabulary.
"as" is in the vocabulary.
"it" is in the vocabulary.
"rolled" is *not* in the vocabulary.
"by" is in the vocabulary.
"the" is in the vocabulary.
"hat" is *not* in the vocabulary.
"on" is in the vocabulary.
"the" is in the vocabulary.
"ground." is *not* in the vocabulary.
```

We want to determine the number of non-vocabulary words in each news articles (denoted by 1s in the encoding)

```
%%time
doc1000_sizes = []
corpus1000 = []
count1000=0
useless = 0
# stop = 0
percents = []
for example, _ in dataset_all.as_numpy_iterator():
  # stop+=1
  # if stop > 5: break
  enc_example = encoder_1000(example)
  num_ones = tf.math.count_nonzero(enc_example==1).numpy()
  percent_ones = round(num_ones*100/len(enc_example))
  # print(f"{percent_ones}%")
  percents.append(percent_ones)

  s = set(list(enc_example.numpy()))
  if s == {1}: useless+=1

  doc1000_sizes.append(len(enc_example))
  corpus1000+=list(enc_example.numpy())

  count1000 += tf.math.count_nonzero(enc_example>1)
```

```
CPU times: user 16min 15s, sys: 1min 41s, total: 17min 57s
Wall time: 15min 19s
```

```
print(f"There are {len(corpus1000)} words in the corpus of {len(do
print(f"Each news article has between {min(doc1000_sizes)} and {ma
```

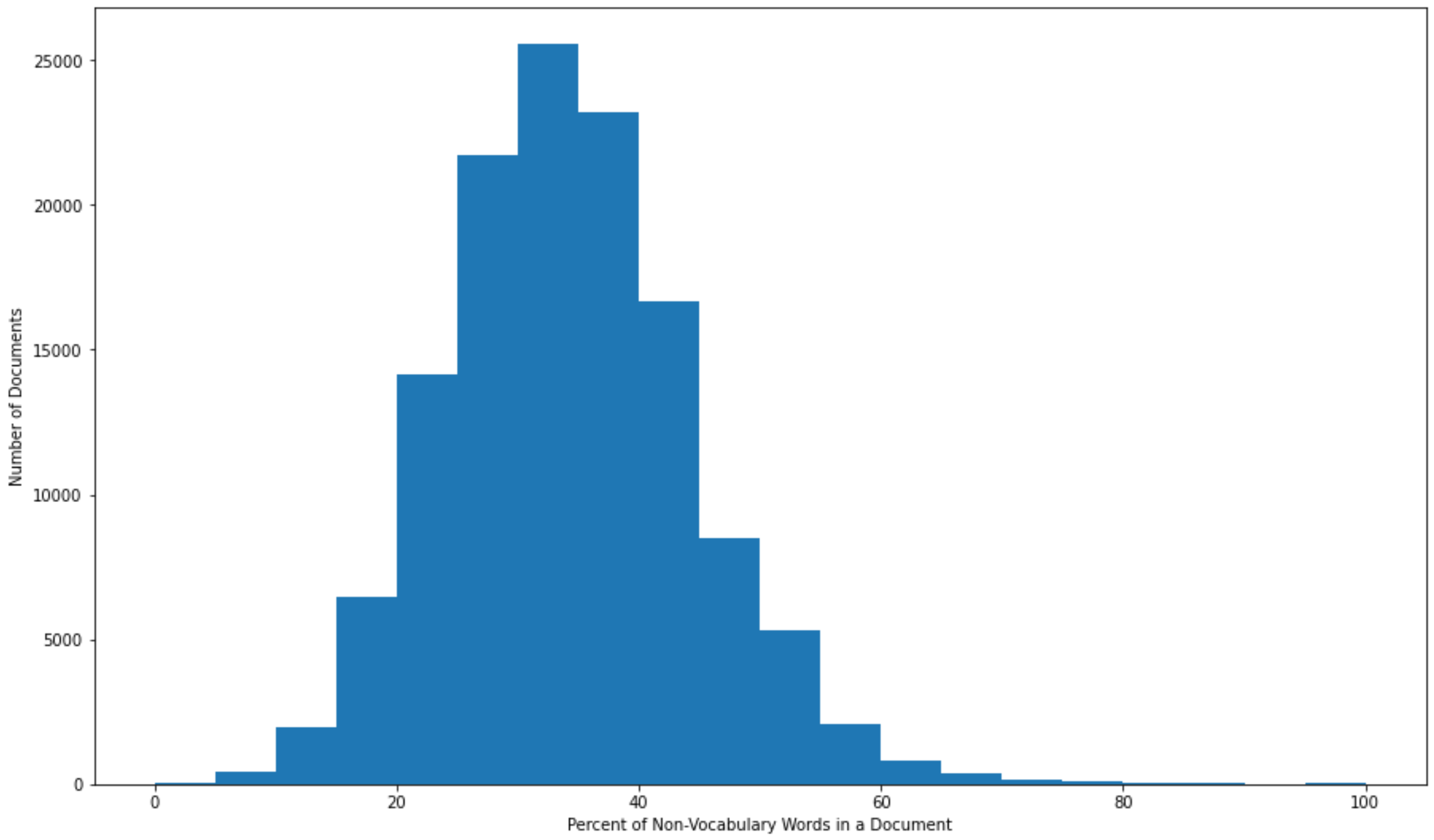
```
There are 3909695 words in the corpus of 127600 news articles.
Each news article has between 3 and 173 tokens in it.
```

Note below that most of the news articles consists of at least 60% (top 1000) vocabulary words (with only 22 out for 127,600 news articles containing no top 1000 vacabulary words)

```
Counter(percents).most_common(10)
np.unique(percents, return_counts=True)
```

```
(array([  0,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,
         15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27,
         28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40,
         41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,
         54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,  66,
         67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,
         80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,
         93,  95,  97, 100]),
 array([  12,    5,   20,   29,   49,   71,  120,  137,  172,  274,  486,
         368,  682,  777,  828, 1346, 1376, 2123, 1610, 2863, 2687, 3018,
        3957, 3315, 4605, 3934, 4148, 5737, 4954, 5072, 6123, 6165, 3255,
        5095, 4596, 3438, 5880, 4205, 3109, 4063, 3555, 3030, 2904, 2294,
        1967, 1716, 2174,  368, 2486,  234, 1246,  747,  573,  547,  520,
         441,  348,  217,  250,  135,  234,   49,  131,  101,   11,  135,
          40,   62,   33,   53,    8,   34,   10,   55,   10,    8,   15,
           3,   27,    9,    5,   19,    1,    8,   12,    6,   16,   11,
           5,    4,    2,    3,    1,    1,   22]))
```

```
plt.figure(figsize=(15,9))
plt.hist(percents, 20)
plt.ylabel('Number of Documents')
plt.xlabel('Percent of Non-Vocabulary Words in a Document');
```



```
print(f"There are {len(corpus1000)} words in the corpus with {usel
print(f"There are {count1000} top {len(vocab_1000)} vocabulary wor
```

```
There are 3909695 words in the corpus with 22 documents not containing any of the top 100
There are 2602670 top 1000 vocabulary words in the corpus.
```

Preprocessing Data Create Input Pipeline

```
# register  ag_news_subset so that tfds.load doesn't generate a ch
!python -m tensorflow_datasets.scripts.download_and_prepare --regi

# Example Approaches to Split Data Set
# dataset, info = tfds.load('ag_news_subset', with_info=True,  spl
dataset, info = tfds.load('ag_news_subset', with_info=True,  split
# dataset, info = tfds.load('ag_news_subset', with_info=True,  spl
                          as_supervised=True)
train_dataset, validation_dataset, test_dataset = dataset
# train_dataset, test_dataset = dataset['train'],dataset['test']
```

```
2021-02-20 20:24:13.174102: I tensorflow/stream_executor/platform/default/dso_loader.cc:4
I0220 20:24:16.083602 140367600310144 download_and_prepare.py:200] Running download_and_p
ag_news_subset
I0220 20:24:16.084728 140367600310144 dataset_info.py:361] Load dataset info from /root/t
I0220 20:24:16.086317 140367600310144 download_and_prepare.py:138] download_and_prepare f
I0220 20:24:16.086621 140367600310144 dataset_builder.py:299] Reusing dataset ag_news_sub
name: "ag_news_subset"
description: "AG is a collection of more than 1 million news articles.\nNews articles hav
citation: "@misc{zhang2015characterlevel,\n    title={Character-level Convolutional Netwo
location {
  urls: "https://arxiv.org/abs/1509.01626"
}
splits {
  name: "test"
  shard_lengths: 7600
  num_bytes: 2226751
}
splits {
  name: "train"
  shard_lengths: 120000
  num_bytes: 35301386
}
supervised_keys {
  input: "description"
  output: "label"
}
version: "1.0.0"
download_size: 11784327
```

```
####  Dataset Splits (Training, Test, Validation)
#### .8934 Training, .0470 Test, .0596 Validation


len(train_dataset),len(validation_dataset),len(test_dataset)
# len(train_dataset),len(test_dataset)
```

```
(114000, 6000, 7600)
```

Review Distribution of Categorical Labels for the 114000 training data (news articles)

```
from collections import Counter
train_categories = [categories[label] for label in train_dataset.m
Counter(train_categories).most_common()
```

```
[('Business', 28531), ('Sports', 28495), ('World', 28491), ('Sci/Tech', 28483)]
```

Review Example with Interger Label Encoded Classification(text, label pairs):

```
for example, label in train_dataset.take(1):
  print('text: ', example.numpy())
  print('class: ', categories[label.numpy()])
```

```
text:  b'AMD #39;s new dual-core Opteron chip is designed mainly for corporate computing
class:  Sci/Tech
```

Preprocessing Shuffle Data for Training and Create Batches of `(text, label)` pairs:

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

```
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZ
validation_dataset = validation_dataset.shuffle(BUFFER_SIZE).batch
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.exp
```

```
for example, label in train_dataset.take(2):
  print('texts: ', example.numpy()[:3])
  print()
  print('labels: ', label.numpy()[:3])
```

```
texts:  [b'Nearly a decade in the making, the MBTA is entering the homestretch of a new \
 b'THE GRAND NABOB of the world #39;s software giant, Microsoft, Bill Gates, has told tec
 b'The volume of worms and viruses is increasing, but the rate of successful attacks has

labels:  [3 3 3]
texts:  [b'Kraft, the largest US food company, on Monday will reveal details of a high-st
 b'PalmSource is to focus on wireless devices with its new version of Palm OS, but can it
 b"AP - The Baltimore Ravens did their part to get into the playoffs. It wasn't enough. A

labels:  [2 3 1]
```

```
for example, label in train_dataset.take(2):
  print('texts: ', example.numpy()[:3])
  print()
  print( labels:  , [categories[n] for n in label.numpy()[:3]])
  print()
```

```
texts:  [b'By TARA BURGHART     CHICAGO (AP) -- When Rueben Martinez became a barber in t
 b'LA CROSSE, Wis. - In 2000, political pundits summed up the race in three words: Florid
 b'Ford Motor Co. said Monday it will begin offering Sirius Satellite Radio as a dealer-i

labels:  ['Sci/Tech', 'World', 'Business']

texts:  [b'The previous NL game in an AL ballpark was in 1946, when the Boston Braves pla
 b'South Korea #39;s economy may miss its 5 percent growth target for 2004 after expandin
 b' SINGAPORE (Reuters) - Oil prices rose for the second day in  a row on Thursday with a

labels:  ['Sports', 'Business', 'Business']
```

Create the Text Encoder

The raw text loaded by `tfds` needs to be processed before it can be used in a model. The simplest way to process text for training is using the `experimental.preprocessing.TextVectorization` layer. This layer has many capabilities, but this tutorial sticks to the default behavior.

Create the layer, and pass the dataset's text to the layer's `.adapt` method:

The `.adapt` method sets the layer's vocabulary. Here are the first 20 tokens. After the padding and unknown tokens they're sorted by frequency:

```
VOCAB_SIZE=1000
encoder = tf.keras.layers.experimental.preprocessing.TextVectoriza
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label: text))
```

```
vocab = np.array(encoder.get_vocabulary())
len(vocab)
```

```
1000
```

```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

```
array(['', '[UNK]', 'the', 'a', 'to', 'of', 'in', 'and', 'on', 'for',
       'that', '39s', 'with', 'its', 'as', 'at', 'is', 'said', 'by', 'it'],
      dtype='<U14')
```

Here are the 20 least frequent words.

```
vocab[-20:]
```

```
vocab[-20:]
```

```
array(['black', 'turn', 'build', 'countrys', 'advanced', 'whose',
       'crisis', 'create', '23', 'sources', 'body', 'militant', 'hope',
       'event', 'started', 'ready', 'jones', 'lawsuit', 'focus',
       'singapore'], dtype='<U14')
```

Once the vocabulary is set, the layer can encode text into indices. The tensors of indices are 0-padded to the longest sequence in the batch (unless you set a fixed `output_sequence_length`):

```
encoded_example = encoder(example)[:3].numpy()
encoded_example
```

```
array([[  2,   1,   1,  87,   6,  22, 783,   1,  27,   6,   1,  68,   2,
        288,   1,   1, 385, 203,  15,   1, 872, 220,   3,   1, 545,   8,
          2,   1,  15,   1, 487,  67,  60,   1, 146,   4,   2,   1, 210,
          1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [197, 626,  11, 353, 107,   1,  13, 705,  88, 272, 966,   9, 112,
         29,   1,  15,  13,   1,   1,   6,   3,  63,   6,   2, 155, 221,
          2, 968,  97,  17,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [999,  28,  70,  94, 240,   9,   2, 103, 121,   6,   3,   1,   8,
         45,  12,  22,   1,   1, 814,   4, 247,   2,  72, 114,  95,  76,
          1,  47, 443,   9,   1,   1,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]])
```

With the default settings, the process is not completely reversible. There are three main reasons for that:

1. The default value for `preprocessing.TextVectorization`'s `standardize` argument is `"lower_and_strip_punctuation"`.
2. The limited vocabulary size and lack of character-based fallback results in some unknown tokens.

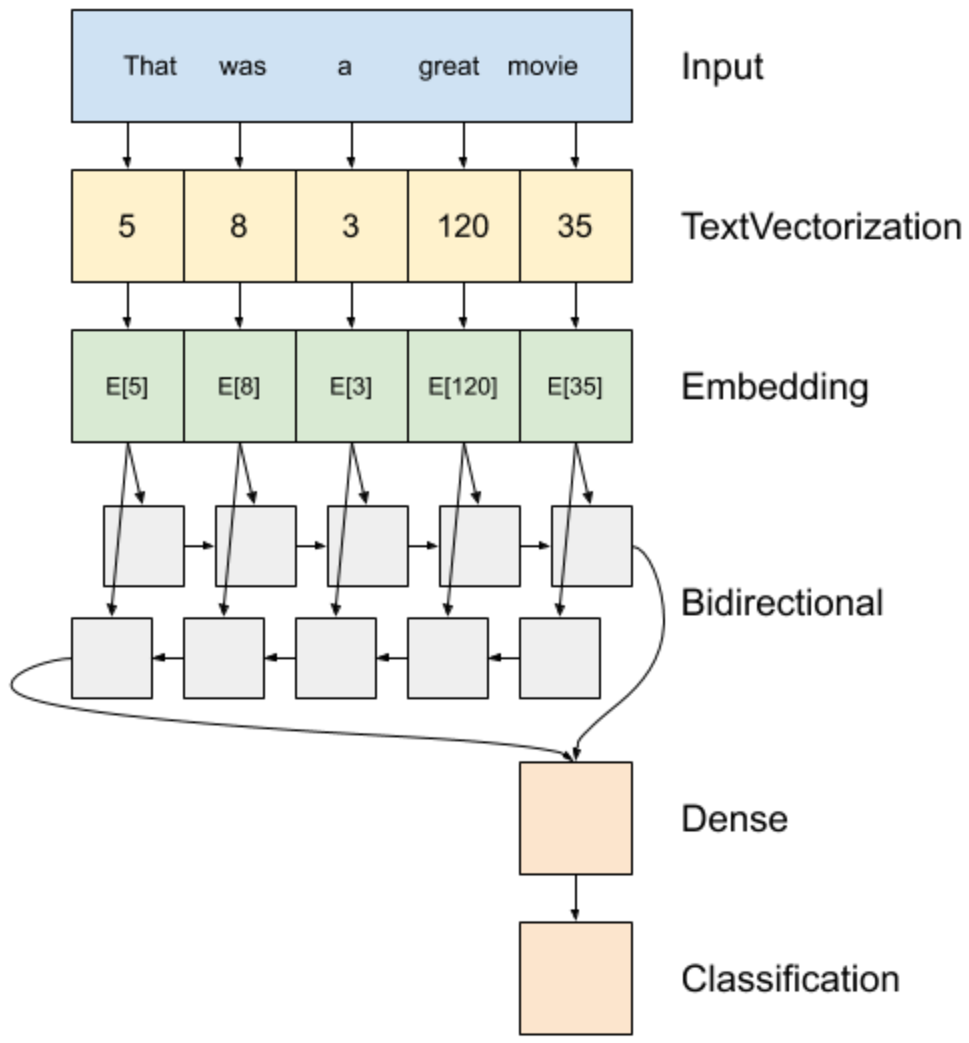
```
for n in range(3):
  print("Original: ", example[n].numpy())
  print("Round-trip: ", " ".join(vocab[encoded_example[n]]))
  print()
```

```
Original:  b'The previous NL game in an AL ballpark was in 1946, when the Boston Braves p
Round-trip:  the [UNK] [UNK] game in an al [UNK] was in [UNK] when the boston [UNK] [UNK]

Original:  b'South Korea #39;s economy may miss its 5 percent growth target for 2004 afte
Round-trip:  south korea 39s economy may [UNK] its 5 percent growth target for 2004 after

Original:  b' SINGAPORE (Reuters) - Oil prices rose for the second day in  a row on Thurs
Round-trip:  singapore reuters oil prices rose for the second day in a [UNK] on thursday
```

Create the model



Above is a diagram of the model.

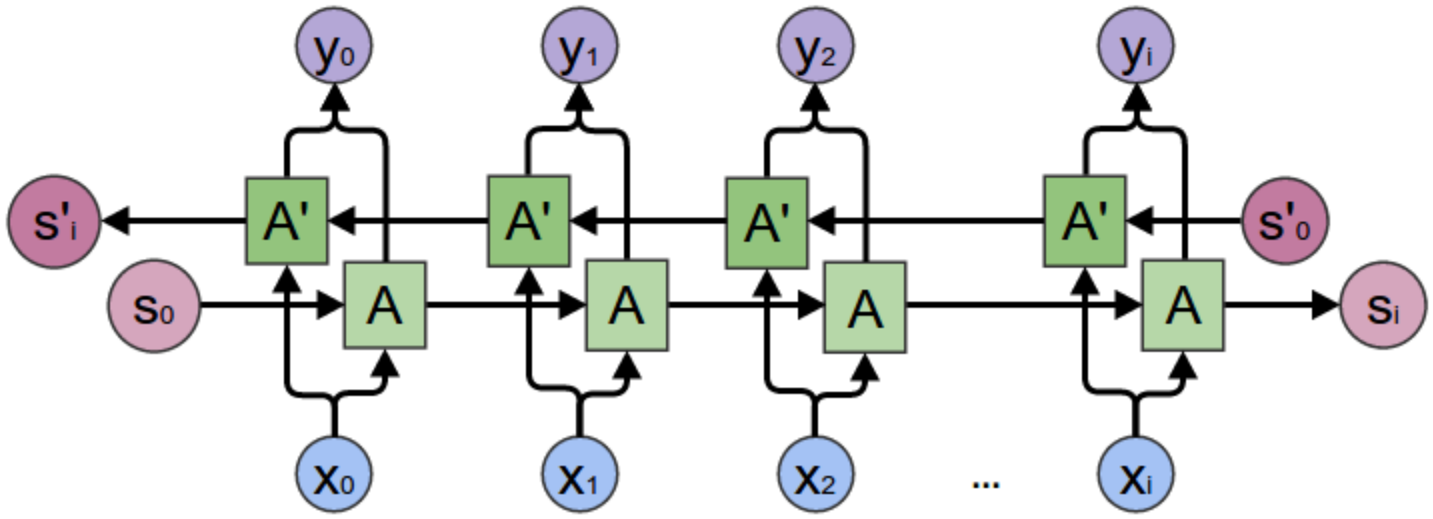
1. This model can be build as a `tf.keras.Sequential`.
2. The first layer is the `encoder`, which converts the text to a sequence of token indices.
3. After the encoder is an embedding layer. An embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.

   This index-lookup is much more efficient than the equivalent operation of passing a one-hot encoded vector through a `tf.keras.layers.Dense` layer.
4. A recurrent neural network (RNN) processes sequence input by iterating through the elements. RNNs pass the outputs from one timestep to their input on the next timestep.

   The `tf.keras.layers.Bidirectional` wrapper can also be used with an RNN layer. This propagates the input forward and backwards through the RNN layer and then concatenates the final output.

    - The main advantage to a bidirectional RNN is that the signal from the beginning of the input doesn't need to be processed all the way through every timestep to affect the output.
    - The main disadvantage of a bidirectional RNN is that you can't efficiently stream predictions as words are being added to the end.
5. After the RNN has converted the sequence to a single vector the two `layers.Dense` do some final processing, and convert from this vector representation to a single logit as the classification output.

tf.keras.layers.Bidirectional
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional



Experiment 1

```
num_classes = 4
model = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(input_dim
                              ,output_dim=32
                                # Use masking to handle the variab
                              ,mask_zero=True)
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(32, activatio
                              ,tf.keras.layers.Dense(num_classes,a
])
```

Please note that we choose to Keras sequential model here since all the layers in the model only have single input and produce single output.

tf.keras.Model
https://www.tensorflow.org/api_docs/python/tf/keras/Model

Compile Model

tf.keras.losses.SparseCategoricalCrossentropy
https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4)
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

Please note that Keras sequential model is used here since all the layers in the model only have single input and produce single output. In case you want to use stateful RNN layer, you might want to build your model with Keras functional API or model subclassing so that you can retrieve and reuse the RNN layer states. Please check Keras RNN guide for more details.

The embedding layer uses masking to handle the varying sequence-lengths. All the layers after the `Embedding` support masking:

```
print([layer.supports_masking for layer in model.layers])
```

```
[False, True, True, True, True]
```

Train the model

Module: tf.keras.callbacks

tf.keras.callbacks.EarlyStopping
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping
tf.keras.callbacks.ModelCheckpoint
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint

```
%%time
history = model.fit(train_dataset
                    ,epochs = 5
                    ,validation_data=validation_dataset
                    )
```

```
Epoch 1/5
1782/1782 [==============================] - 47s 18ms/step - loss: 1.1274 - accuracy: 0.4
Epoch 2/5
1782/1782 [==============================] - 30s 17ms/step - loss: 0.5165 - accuracy: 0.8
Epoch 3/5
1782/1782 [==============================] - 29s 16ms/step - loss: 0.4588 - accuracy: 0.8
Epoch 4/5
 678/1782 [==========>...................] - ETA: 17s - loss: 0.4405 - accuracy: 0.8457
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-50-41d509b6d568> in <module>()
----> 1 get_ipython().run_cell_magic('time', '', 'history = model.fit(train_dataset\n
,epochs = 5\n                      ,validation_data=validation_dataset\n
)')

9 frames

<decorator-gen-60> in time(self, line, cell, local_ns)

<timed exec> in <module>()

/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Plotting Performance Metrics - Single Layer Bidirectional RNN

We use Matplotlib to create 2 plots--displaying the training and validation loss (resp. accuracy) for each (training) epoch side by side.

```
history_dict = history.history
history_dict.keys()
```

```
history_df=pd.DataFrame(history_dict)
history_df.tail(10)
```

```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

```
plt.figure(figsize=(16, 4))
for i, metrics in enumerate(zip([losses, accs], [val_losses, val_a
    plt.subplot(1, 2, i + 1)
    plt.plot(range(epochs), metrics[0], label='Training {}'.format
    plt.plot(range(epochs), metrics[1], label='Validation {}'.form
    plt.legend()
plt.show()
```

Model Architecture Summary Single Layer Bidirectional RNN

```
model.summary()
```

```
keras.utils.plot_model(model, "BiDirectionalLSTM.png", show_shapes
```

Stack two or more LSTM layers

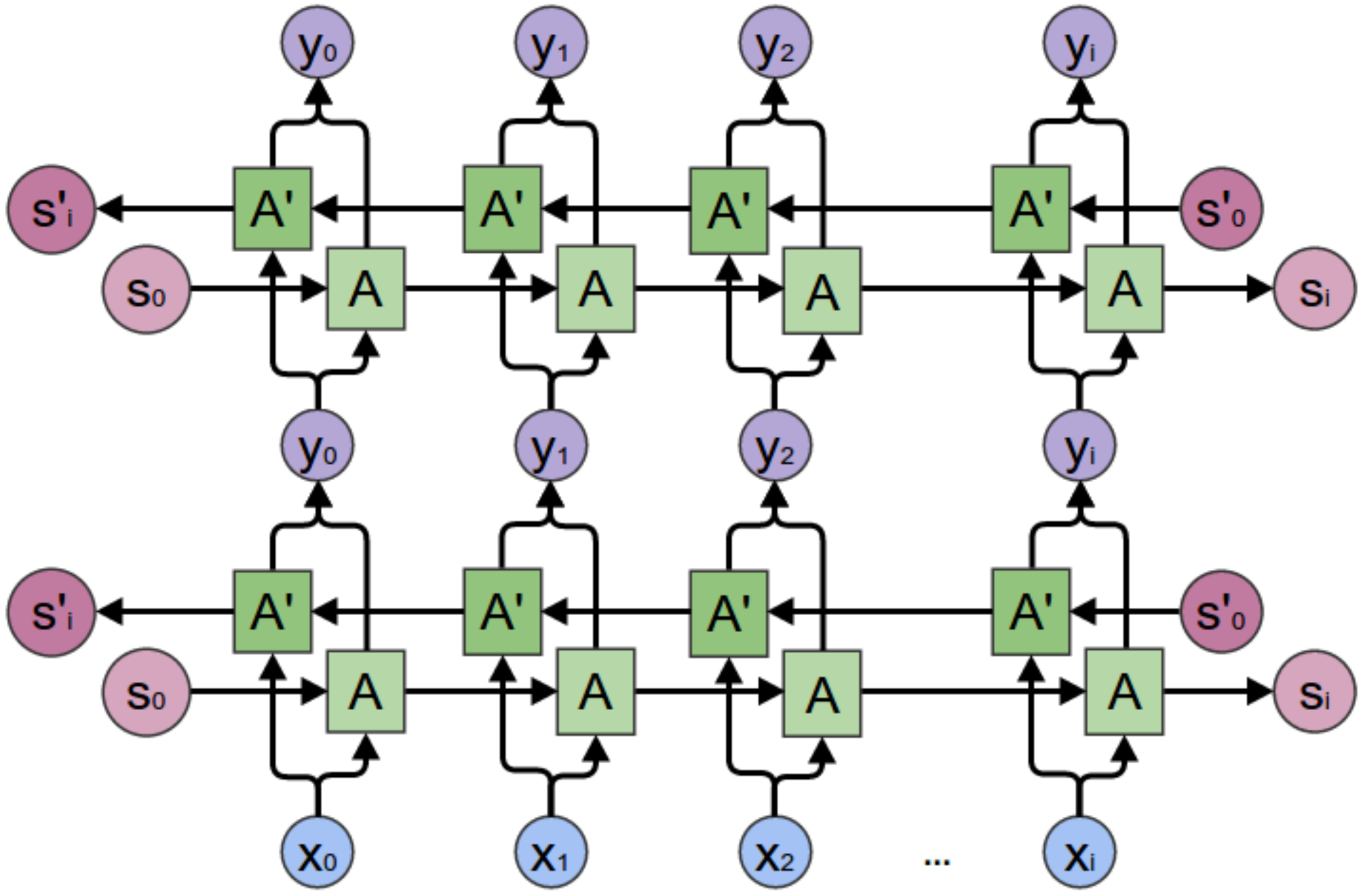
Keras recurrent layers have two available modes that are controlled by the `return_sequences` constructor argument:

- If `False` it returns only the last output for each input sequence (a 2D tensor of shape (batch_size, output_features)). This is the default, used in the previous model.
- If `True` the full sequences of successive outputs for each timestep is returned (a 3D tensor of shape `(batch_size, timesteps, output_features)`).

Here is what the flow of information looks like with `return_sequences=True`:

![layered_bidirectional]?raw=1"

The interesting thing about using an `RNN` with `return_sequences=True` is that the output still has 3-axes, like the input, so it can be passed to another RNN layer, like this:



Experiment 2

```
model2 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(32, activatio
                              ,tf.keras.layers.Dense(32, activatio
                              ,tf.keras.layers.Dense(num_classes,a
])
```

tf.keras.losses.SparseCategoricalCrossentropy
https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

```
model2.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history2 = model2.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model2.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Plotting Performance Metrics - Multi-Layer Bidirectional RNN

We use Matplotlib to create 2 plots--displaying the training and validation loss (resp. accuracy) for each (training) epoch side by side.

```
history_dict2 = history2.history
history_dict2.keys()
```

```
history2_df=pd.DataFrame(history_dict2)
history2_df.tail()
```

```
losses = history2.history['loss']
accs = history2.history['accuracy']
val_losses = history2.history['val_loss']
val_accs = history2.history['val_accuracy']
epochs = len(losses)
```

```
plt.figure(figsize=(16, 4))
for i, metrics in enumerate(zip([losses, accs], [val_losses, val_a
    plt.subplot(1, 2, i + 1)
    plt.plot(range(epochs), metrics[0], label='Training {}'.format
    plt.plot(range(epochs), metrics[1], label='Validation {}'.form
    plt.legend()
plt.show()
```

Model Architecture Summary Single Layer Bidirectional RNN

```
model2.summary()
```

```
keras.utils.plot_model(model2, "2Layer_BiDirectionalLSTM.png", sho
```

Check out other existing recurrent layers such as GRU layers.

If you're interestied in building custom RNNs, see the Keras RNN Guide.

Experiment 3

```
model3 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(64, activatio
                              ,tf.keras.layers.Dense(num_classes,a
])
```

```
model3.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history3 = model3.fit(train_dataset
                     ,epochs=5
                     ,validation_data=validation_dataset
                     ,validation_steps=30
                     )
```

```
test_loss, test_acc = model3.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 4

```
model4 = tf.keras.Sequential([
                              encoder
```

```
                                ,tf.keras.layers.Embedding(len(encod
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Dense(64, activatio
                                ,tf.keras.layers.Dense(64, activatio
                                ,tf.keras.layers.Dense(num_classes,a
])

model4.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history4 = model4.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model4.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 5

```
model5 = tf.keras.Sequential([
                                encoder
                                ,tf.keras.layers.Embedding(len(encod
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Dense(128, activati
                                ,tf.keras.layers.Dense(num_classes,a
])

model5.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
```

```
history5 = model5.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model5.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 6

```
model6 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(128, activati
                              ,tf.keras.layers.Dense(128, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model6.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history6 = model6.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model6.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 7

```
model7 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(256, activati
                              ,tf.keras.layers.Dense(num_classes,a
])
```

```
model7.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history7 = model7.fit(train_dataset
                     ,epochs=5
                     ,validation_data=validation_dataset
                     ,validation_steps=30
                     )
```

```
test_loss, test_acc = model7.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 8

```
model8 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(256, activati
                              ,tf.keras.layers.Dense(256, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model8.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history8 = model8.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )
```

```
test_loss, test_acc = model8.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 9

```
model9 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model9.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history9 = model9.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )
                    )

test_loss, test_acc = model9.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 10

```
model10 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model10.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history10 = model10.fit(train_dataset
                      ,epochs=5
                      ,validation_data=validation_dataset
                      ,validation_steps=30
                      )

test_loss, test_acc = model10.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 11

```
model11 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model11.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history11 = model11.fit(train_dataset
                     ,epochs=5
                     ,validation_data=validation_dataset
                     ,validation_steps=30
                     )

test_loss, test_acc = model11.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 12

```
model12 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])
```

```
model12.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history12 = model12.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )
```

```
test_loss, test_acc = model12.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
from sklearn.metrics import confusion_matrix
y_pred = model12.predict(test_dataset)
predicted_categories = tf.argmax(y_pred, axis=1)


true_categories = tf.concat([y for x, y in test_dataset], axis=0)

confusion_matrix(predicted_categories, true_categories)
```

```
from sklearn.metrics import precision_score
precision_score(true_categories, predicted_categories, average='mi
```

```
from sklearn.metrics import recall_score
recall_score(true_categories, predicted_categories, average='micro
```

Experiment 13

```
model13 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dropout(0.5)
                                ,tf.keras.layers.Dense(num_classes,a
])

model13.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history13 = model13.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model13.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 14

```
model14 = tf.keras.Sequential([
                                encoder
                                ,tf.keras.layers.Embedding(len(encod
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Dropout(0.5)
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dropout(0.5)
                                ,tf.keras.layers.Dense(num_classes,a
])
```

```
model14.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history14 = model14.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model14.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 15

```
from tensorflow.keras import regularizers
model15 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model15.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history15 = model15.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
```

```
                      ,validation_steps=30
                      )

test_loss, test_acc = model15.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 16

```
model16 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model16.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history16 = model16.fit(train_dataset
                      ,epochs=5
                      ,validation_data=validation_dataset
                      ,validation_steps=30
                      )

test_loss, test_acc = model16.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 17

```
model17 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model17.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history17 = model17.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    ,callbacks=[tf.keras.callbacks.EarlyStopping(m
                    )

test_loss, test_acc = model17.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 18

```
encoder_none = tf.keras.layers.experimental.preprocessing.TextVect
    max_tokens=None)
encoder_none.adapt(train_dataset.map(lambda text, label: text))
model18 = tf.keras.Sequential([
                              encoder_none
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])
```

```
model18.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])
```

```
%%time
history18 = model18.fit(train_dataset
                       ,epochs=5
                       ,validation_data=validation_dataset
                       ,validation_steps=30
                       )

test_loss, test_acc = model18.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 19

```
encoder_500 = tf.keras.layers.experimental.preprocessing.TextVecto
    max_tokens=500)
encoder_500.adapt(train_dataset.map(lambda text, label: text))
model19 = tf.keras.Sequential([
                                encoder_500
                                ,tf.keras.layers.Embedding(len(encod
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(num_classes,a
])


model19.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history19 = model19.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )

test_loss, test_acc = model19.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 20

```
encoder_none = tf.keras.layers.experimental.preprocessing.TextVect
    max_tokens=200)
encoder_none.adapt(train_dataset.map(lambda text, label: text))
model20 = tf.keras.Sequential([
                                encoder_none
                                ,tf.keras.layers.Embedding(len(encod
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Bidirectional(tf.ke
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(512, activati
                                ,tf.keras.layers.Dense(num_classes,a
])
```

```
model20.compile(optimizer='adam'
               ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
               ,metrics=['accuracy'])

%%time
history20 = model20.fit(train_dataset
                     ,epochs=5
                     ,validation_data=validation_dataset
                     ,validation_steps=30
                     )

test_loss, test_acc = model20.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 21

```
encoder_none = tf.keras.layers.experimental.preprocessing.TextVect
    max_tokens=1000, output_sequence_length=10)
encoder_none.adapt(train_dataset.map(lambda text, label: text))
model21 = tf.keras.Sequential([
                              encoder_none
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model21.compile(optimizer='adam'
               ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
               ,metrics=['accuracy'])

%%time
history21 = model21.fit(train_dataset
                     ,epochs=5
```

```
                      ,validation_data=validation_dataset
                      ,validation_steps=30
                      )

test_loss, test_acc = model21.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

Experiment 22

```
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers import Embedding
from keras.models import Sequential
from keras.layers import GlobalMaxPooling1D
from keras.layers import Dense, Activation, Flatten

model22 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(input_dim
                              ,tf.keras.layers.Conv1D(32,7)
                              ,tf.keras.layers.MaxPooling1D(5)
                              ,tf.keras.layers.GlobalMaxPooling1D(
                              ,tf.keras.layers.Dense(num_classes,a

model22.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history22 = model22.fit(train_dataset
                      ,epochs=5
                      ,validation_data=validation_dataset
                      ,validation_steps=30
                      )
```

```
Epoch 1/5
1782/1782 [==============================] - 16s 8ms/step - loss: 0.7222 - accuracy: 0.72
Epoch 2/5
1782/1782 [==============================] - 11s 6ms/step - loss: 0.3924 - accuracy: 0.85
```

```
Epoch 3/5
1782/1782 [==============================] - 11s 6ms/step - loss: 0.3591 - accuracy: 0.87
Epoch 4/5
1782/1782 [==============================] - 11s 6ms/step - loss: 0.3313 - accuracy: 0.87
Epoch 5/5
1782/1782 [==============================] - 11s 6ms/step - loss: 0.3063 - accuracy: 0.88
CPU times: user 1min 38s, sys: 24.6 s, total: 2min 2s
Wall time: 58.6 s
```

```
test_loss, test_acc = model22.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
119/119 [==============================] - 1s 5ms/step - loss: 0.4110 - accuracy: 0.8532
Test Loss: 0.41099876165390015
Test Accuracy: 0.8531578779220581
```

Experiment 23

```
model23 = tf.keras.Sequential([
                              encoder
                              ,tf.keras.layers.Embedding(len(encod
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Bidirectional(tf.ke
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(512, activati
                              ,tf.keras.layers.Dense(num_classes,a
])

model23.compile(optimizer='adam'
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy(
              ,metrics=['accuracy'])

%%time
history23 = model23.fit(train_dataset
                    ,epochs=5
                    ,validation_data=validation_dataset
                    ,validation_steps=30
                    )
```

```
test_loss, test_acc = model23.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```