Keras for Text Classification

Learning Objectives

1. Learn how to create a text classification datasets using BigQuery
2. Learn how to tokenize and integerize a corpus of text for training in Keras
3. Learn how to do one-hot-encodings in Keras
4. Learn how to use embedding layers to represent words in Keras
5. Learn about the bag-of-word representation for sentences
6. Learn how to use DNN/CNN/RNN model to classify text in keras

Introduction

In this notebook, we will implement text models to recognize the probable source (Github, Tech-Crunch, or The New-York Times) of the titles we have in the title dataset we constructed in the first task of the lab.

In the next step, we will load and pre-process the texts and labels so that they are suitable to be fed to a Keras model. For the texts of the titles we will learn how to split them into a list of tokens, and then how to map each token to an integer using the Keras Tokenizer class. What will be fed to our Keras models will be batches of padded list of integers representing the text. For the labels, we will learn how to one-hot-encode each of the 3 classes into a 3 dimensional basis vector.

Then we will explore a few possible models to do the title classification. All models will be fed padded list of integers, and all models will start with a Keras Embedding layer that transforms the integer representing the words into dense vectors.

The first model will be a simple bag-of-word DNN model that averages up the word vectors and feeds the tensor that results to further dense layers. Doing so means that we forget the word order (and hence that we consider sentences as a “bag-of-words”). In the second and in the third model we will keep the information about the word order using a simple RNN and a simple CNN allowing us to achieve the same performance as with the DNN model but in much fewer epochs.

In [6]:

```
import os

from google.cloud import bigquery
import pandas as pd
```

In [7]:

```
%load_ext google.cloud.bigquery
```

```
The google.cloud.bigquery extension is already loaded. To reload it, use:
  %reload_ext google.cloud.bigquery
```

Replace the variable values in the cell below:

In [1]:
```
PROJECT = "qwiklabs-gcp-04-184484a58d79"  # Replace with your PROJECT
BUCKET = PROJECT  # defaults to PROJECT
REGION = "us-central1"  # Replace with your REGION
SEED = 0
```

Create a Dataset from BigQuery

Hacker news headlines are available as a BigQuery public dataset. The dataset contains all headlines from the sites inception in October 2006 until October 2015.

Here is a sample of the dataset:

In [2]:
```
%%bigquery --project $PROJECT

SELECT
    url, title, score
FROM
    `bigquery-public-data.hacker_news.stories`
WHERE
    LENGTH(title) > 10
    AND score > 10
    AND LENGTH(url) > 0
LIMIT 10
```

```
Query complete after 0.00s: 100%|██████████| 1/1 [00:00<00:00, 1230.00query/s]
Downloading: 100%|██████████| 10/10 [00:01<00:00,  8.89rows/s]
```

Out[2]:

	url	title	score
0	http://www.dumpert.nl/mediabase/6560049/3eb18e...	Calling the NSA: "I accidentally deleted an e-...	258
1	http://blog.liip.ch/archive/2013/10/28/hhvm-an...	Amazing performance with HHVM and PHP with a S...	11
2	http://www.gamedev.net/page/resources/_/techni...	A Journey Through the CPU Pipeline	11
3	http://jfarcand.wordpress.com/2011/02/25/atmos...	Atmosphere Framework 0.7 released: GWT, Wicket...	11
4	http://tech.gilt.com/post/90578399884/immutabl...	Immutable Infrastructure with Docker and EC2 [...	11
5	http://thechangelog.com/post/501053444/episode...	Changelog 0.2.0 - node.js w/Felix Geisendorfer	11
6	http://openangelforum.com/2010/09/09/second-bo...	Second Open Angel Forum in Boston Oct 13th--fr...	11
7	http://bredele.github.io/async	A collection of JavaScript asynchronous patterns	11
8	http://www.smashingmagazine.com/2007/08/25/20-...	20 Free and Fresh Icon Sets	11
9	http://www.cio.com/article/147801/Study_Finds_...	Study: Only 1 in 5 Workers is "Engaged" in The...	11

Let's do some regular expression parsing in BigQuery to get the source of the newspaper article from the URL. For example, if the url is http://mobile.nytimes.com/...., I want to be left with *nytimes*

In [3]:
```
%%bigquery --project $PROJECT

SELECT
    ARRAY_REVERSE(SPLIT(REGEXP_EXTRACT(url, '.*://(.[^/]+)/'), '.'))[OFFSET(1)]
    COUNT(title) AS num_articles
FROM
    `bigquery-public-data.hacker_news.stories`
WHERE
    REGEXP_CONTAINS(REGEXP_EXTRACT(url, '.*://(.[^/]+)/'), '.com$')
    AND LENGTH(title) > 10
GROUP BY
    source
ORDER BY num_articles DESC
  LIMIT 100
```

```
Query complete after 0.00s: 100%|██████████| 1/1 [00:00<00:00, 1340.03query/s]
Downloading: 100%|██████████| 100/100 [00:01<00:00, 72.70rows/s]
```

Out[3]:

	source	num_articles
0	blogspot	41386
1	github	36525
2	techcrunch	30891
3	youtube	30848
4	nytimes	28787
...	...	...
95	f5	1254
96	gamasutra	1249
97	cnbc	1229
98	indiatimes	1223
99	computerworlduk	1166

100 rows × 2 columns

Now that we have good parsing of the URL to get the source, let's put together a dataset of source and titles. This will be our labeled dataset for machine learning.

In [4]:
```
regex = '.*://(.[^/]+)/'


sub_query = """
SELECT
    title,
    ARRAY_REVERSE(SPLIT(REGEXP_EXTRACT(url, '{0}'), '.'))[OFFSET(1)] AS source

FROM
    `bigquery-public-data.hacker_news.stories`
WHERE
```

```
    REGEXP_CONTAINS(REGEXP_EXTRACT(url, '{0}'), '.com$')
    AND LENGTH(title) > 10
""".format(regex)


query = """
SELECT
    LOWER(REGEXP_REPLACE(title, '[^a-zA-Z0-9 $.-]', ' ')) AS title,
    source
FROM
  ({sub_query})
WHERE (source = 'github' OR source = 'nytimes' OR source = 'techcrunch')
""".format(sub_query=sub_query)

print(query)
```

```
SELECT
    LOWER(REGEXP_REPLACE(title, '[^a-zA-Z0-9 $.-]', ' ')) AS title,
    source
FROM
  (
SELECT
    title,
    ARRAY_REVERSE(SPLIT(REGEXP_EXTRACT(url, '.*://(.[^/]+)/'), '.'))[OFFSET(1)]
AS source

FROM
    `bigquery-public-data.hacker_news.stories`
WHERE
    REGEXP_CONTAINS(REGEXP_EXTRACT(url, '.*://(.[^/]+)/'), '.com$')
    AND LENGTH(title) > 10
)
WHERE (source = 'github' OR source = 'nytimes' OR source = 'techcrunch')
```

For ML training, we usually need to split our dataset into training and evaluation datasets (and perhaps an independent test dataset if we are going to do model or feature selection based on the evaluation dataset). AutoML however figures out on its own how to create these splits, so we won't need to do that here.

In [8]:

```
bq = bigquery.Client(project=PROJECT)
title_dataset = bq.query(query).to_dataframe()
title_dataset.head()
```

Out[8]:

	title	source
0	this guy just found out how to bypass adblocker	github
1	show hn dodo command line task management f...	github
2	without coding test test automation for javas...	github
3	clojure s first code commit authored 8 years ...	github
4	hikaricp a solid high-performance jdbc connect...	github

AutoML for text classification requires that

- the dataset be in csv form with

- the first column being the texts to classify or a GCS path to the text
- the last colum to be the text labels

The dataset we pulled from BiqQuery satisfies these requirements.

In [9]:
```
print("The full dataset contains {n} titles".format(n=len(title_dataset)))
```

```
The full dataset contains 96203 titles
```

Let's make sure we have roughly the same number of labels for each of our three labels:

In [10]:
```
title_dataset.source.value_counts()
```

Out[10]:
```
github        36525
techcrunch    30891
nytimes       28787
Name: source, dtype: int64
```

Finally we will save our data, which is currently in-memory, to disk.

We will create a csv file containing the full dataset and another containing only 1000 articles for development.

Note: It may take a long time to train AutoML on the full dataset, so we recommend to use the sample dataset for the purpose of learning the tool.

In [11]:
```
DATADIR = './data/'

if not os.path.exists(DATADIR):
    os.makedirs(DATADIR)
```

In [12]:
```
FULL_DATASET_NAME = 'titles_full.csv'
FULL_DATASET_PATH = os.path.join(DATADIR, FULL_DATASET_NAME)

# Let's shuffle the data before writing it to disk.
title_dataset = title_dataset.sample(n=len(title_dataset))

title_dataset.to_csv(
    FULL_DATASET_PATH, header=False, index=False, encoding='utf-8')
```

Now let's sample 1000 articles from the full dataset and make sure we have enough examples for each label in our sample dataset (see here for further details on how to prepare data for AutoML).

In [13]:
```
sample_title_dataset = title_dataset.sample(n=1000)
sample_title_dataset.source.value_counts()
```

Out[13]:
```
github        361
techcrunch    349
nytimes       290
Name: source, dtype: int64
```

Let's write the sample datatset to disk.

In [14]:
```
SAMPLE_DATASET_NAME = 'titles_sample.csv'
SAMPLE_DATASET_PATH = os.path.join(DATADIR, SAMPLE_DATASET_NAME)

sample_title_dataset.to_csv(
    SAMPLE_DATASET_PATH, header=False, index=False, encoding='utf-8')
```

In [15]:
```
sample_title_dataset.head()
```

Out[15]:

	title	source
49309	venture capital was tight for tech start-ups i...	nytimes
73365	fujitsu to launch handsets in europe. u.s. next	techcrunch
6853	hp y u no like consumers	github
82488	is mint going after freshbooks its new featur...	techcrunch
85148	gmail traffic between google servers now encry...	techcrunch

In [16]:
```
import os
import shutil

import pandas as pd
import tensorflow as tf
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping
from tensorflow.keras.layers import (
    Embedding,
    Flatten,
    GRU,
    Conv1D,
    Lambda,
    Dense,
)
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical


print(tf.__version__)
```

```
2.6.0
```

In [17]:
```
%matplotlib inline
```

Let's start by specifying where the information about the trained models will be saved as well as where our dataset is located:

In [18]:
```
LOGDIR = "./text_models"
DATA_DIR = "./data"
```

Loading the dataset

Our dataset consists of titles of articles along with the label indicating from which source these articles have been taken from (GitHub, Tech-Crunch, or the New-York Times).

In [19]:
```
DATASET_NAME = "titles_full.csv"
TITLE_SAMPLE_PATH = os.path.join(DATA_DIR, DATASET_NAME)
COLUMNS = ['title', 'source']

titles_df = pd.read_csv(TITLE_SAMPLE_PATH, header=None, names=COLUMNS)
titles_df.head()
```

Out[19]:

	title	source
0	google acquires nik software the popular snaps...	techcrunch
1	a vcr-style alternative to console.log	github
2	cheaters find an adversary in technology	nytimes
3	show hn jquery folder-like collapsible lists	github
4	a backup agent to export your data from the at...	github

Integerize the texts

The first thing we need to do is to find how many words we have in our dataset (`VOCAB_SIZE`), how many titles we have (`DATASET_SIZE`), and what the maximum length of the titles we have (`MAX_LEN`) is. Keras offers the `Tokenizer` class in its `keras.preprocessing.text` module to help us with that:

In [20]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(titles_df.title)
```

In [21]:
```
integerized_titles = tokenizer.texts_to_sequences(titles_df.title)
integerized_titles[:3]
```

Out[21]:
```
[[14, 154, 17274, 158, 1, 1053, 17275, 500, 1149, 22, 4, 74],
 [2, 10466, 197, 396, 3, 624, 1045],
 [17276, 284, 19, 11900, 5, 246]]
```

In [22]:
```
VOCAB_SIZE = len(tokenizer.index_word)
VOCAB_SIZE
```

Out[22]:
```
47271
```

In [23]:
```
DATASET_SIZE = tokenizer.document_count
DATASET_SIZE
```

Out[23]:
```
96203
```

In [24]:
```
MAX_LEN = max(len(sequence) for sequence in integerized_titles)
MAX_LEN
```

Out[24]:
```
26
```

Let's now implement a function `create_sequence` that will

- take as input our titles as well as the maximum sentence length and
- returns a list of the integers corresponding to our tokens padded to the sentence maximum length

Keras has the helper functions `pad_sequence` for that on the top of the tokenizer methods.

In [25]:

```
# TODO 1
def create_sequences(texts, max_len=MAX_LEN):
    sequences = tokenizer.texts_to_sequences(texts)
    padded_sequences = pad_sequences(sequences, max_len, padding='post')
    return padded_sequences
```

In [26]:

```
sequences = create_sequences(titles_df.title[:3])
sequences
```

Out[26]:

```
array([[   14,   154, 17274,   158,     1,  1053, 17275,   500,  1149,
           22,     4,    74,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0],
       [    2, 10466,   197,   396,     3,   624,  1045,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0],
       [17276,   284,    19, 11900,     5,   246,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]],
      dtype=int32)
```

In [27]:

```
titles_df.source[:4]
```

Out[27]:

```
0    techcrunch
1        github
2       nytimes
3        github
Name: source, dtype: object
```

We now need to write a function that

- takes a title source and
- returns the corresponding one-hot encoded vector

Keras `to_categorical` is handy for that.

In [28]:

```
CLASSES = {
    'github': 0,
    'nytimes': 1,
    'techcrunch': 2
}
N_CLASSES = len(CLASSES)
```

In [29]:

```
# TODO 2
def encode_labels(sources):
    classes = [CLASSES[source] for source in sources]
```

```
    one_hots = to_categorical(classes)
    return one_hots
```

In [30]:
```
encode_labels(titles_df.source[:4])
```

Out[30]:
```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]], dtype=float32)
```

Preparing the train/test splits

Let's split our data into train and test splits:

In [31]:
```
N_TRAIN = int(DATASET_SIZE * 0.80)

titles_train, sources_train = (
    titles_df.title[:N_TRAIN], titles_df.source[:N_TRAIN])

titles_valid, sources_valid = (
    titles_df.title[N_TRAIN:], titles_df.source[N_TRAIN:])
```

To be on the safe side, we verify that the train and test splits have roughly the same number of examples per classes.

Since it is the case, accuracy will be a good metric to use to measure the performance of our models.

In [32]:
```
sources_train.value_counts()
```

Out[32]:
```
github        29285
techcrunch    24656
nytimes       23021
Name: source, dtype: int64
```

In [33]:
```
sources_valid.value_counts()
```

Out[33]:
```
github        7240
techcrunch    6235
nytimes       5766
Name: source, dtype: int64
```

Using `create_sequence` and `encode_labels`, we can now prepare the training and validation data to feed our models.

The features will be padded list of integers and the labels will be one-hot-encoded 3D vectors.

In [34]:
```
X_train, Y_train = create_sequences(titles_train), encode_labels(sources_train)
X_valid, Y_valid = create_sequences(titles_valid), encode_labels(sources_valid)
```

In [35]:
```
X_train[:3]
```

Out[35]:
```
array([[   14,   154, 17274,   158,     1,  1053, 17275,   500,  1149,
           22,     4,    74,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0],
       [    2, 10466,   197,   396,     3,   624,  1045,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0],
       [17276,   284,    19, 11900,     5,   246,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]],
      dtype=int32)
```

In [36]:
```
Y_train[:3]
```

Out[36]:
```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.]], dtype=float32)
```

Building a DNN model

The build_dnn_model function below returns a compiled Keras model that implements a simple embedding layer transforming the word integers into dense vectors, followed by a Dense softmax layer that returns the probabilities for each class.

Note that we need to put a custom Keras Lambda layer in between the Embedding layer and the Dense softmax layer to do an average of the word vectors returned by the embedding layer. This is the average that's fed to the dense softmax layer. By doing so, we create a model that is simple but that loses information about the word order, creating a model that sees sentences as "bag-of-words".

In [37]:
```
def build_dnn_model(embed_dim):

    model = Sequential([
        Embedding(VOCAB_SIZE + 1, embed_dim, input_shape=[MAX_LEN]), # TODO 3
        Lambda(lambda x: tf.reduce_mean(x, axis=1)), # TODO 4
        Dense(N_CLASSES, activation='softmax') # TODO 5
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

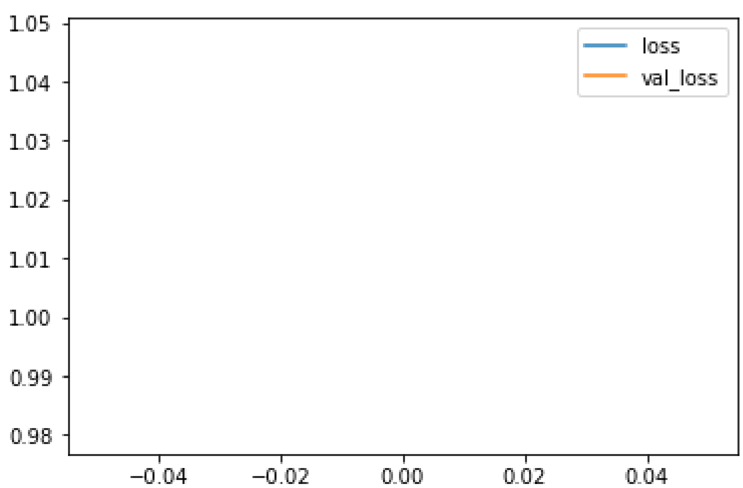
Below we train the model on 100 epochs but adding an `EarlyStopping` callback that will stop the training as soon as the validation loss has not improved after a number of steps specified by `PATIENCE` . Note that we also give the `model.fit` method a Tensorboard callback so that we can later compare all the models using TensorBoard.

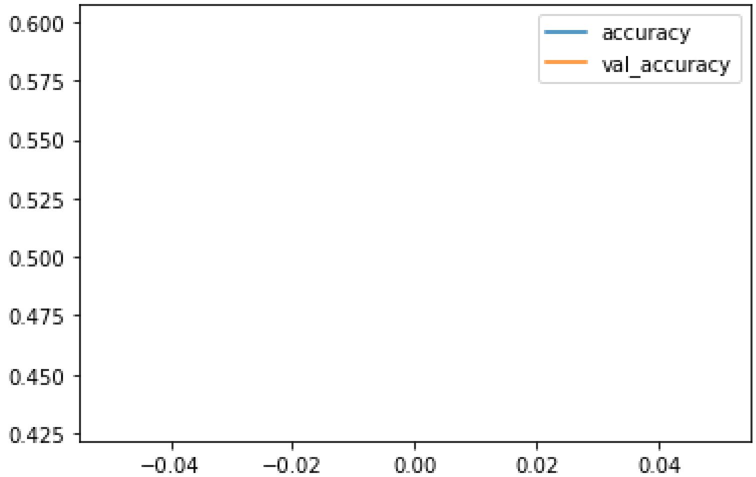
In [38]:
```
%%time

tf.random.set_seed(33)

MODEL_DIR = os.path.join(LOGDIR, 'dnn')
```

```
shutil.rmtree(MODEL_DIR, ignore_errors=True)

BATCH_SIZE = 300
EPOCHS = 100
EMBED_DIM = 10
PATIENCE = 0

dnn_model = build_dnn_model(embed_dim=EMBED_DIM)

dnn_history = dnn_model.fit(
    X_train, Y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=(X_valid, Y_valid),
    callbacks=[EarlyStopping(patience=PATIENCE), TensorBoard(MODEL_DIR)],
)

pd.DataFrame(dnn_history.history)[['loss', 'val_loss']].plot()
pd.DataFrame(dnn_history.history)[['accuracy', 'val_accuracy']].plot()

dnn_model.summary()
```

```
2021-11-03 13:49:14.204197: I tensorflow/core/common_runtime/process_util.cc:14
6] Creating new thread pool with default inter op setting: 2. Tune using inter_o
p_parallelism_threads for best performance.
2021-11-03 13:49:14.266315: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-11-03 13:49:14.266349: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-11-03 13:49:14.267275: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
2021-11-03 13:49:14.341391: I tensorflow/compiler/mlir/mlir_graph_optimization_p
ass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/100
 17/257 [>.............................] - ETA: 2s - loss: 1.0949 - accuracy: 0.
3724
2021-11-03 13:49:14.969183: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-11-03 13:49:14.970199: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-11-03 13:49:14.982868: I tensorflow/core/profiler/lib/profiler_session.cc:6
6] Profiler session collecting data.
2021-11-03 13:49:14.988346: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
2021-11-03 13:49:15.000294: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: ./text_models/dnn/train/plugins/profile/2021_11_03_13
_49_14

2021-11-03 13:49:15.003014: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for trace.json.gz to ./text_models/dnn/train/plu
gins/profile/2021_11_03_13_49_14/tensorflow-2-6-20211103-092159.trace.json.gz
2021-11-03 13:49:15.017674: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: ./text_models/dnn/train/plugins/profile/2021_11_03_13
_49_14

2021-11-03 13:49:15.018881: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for memory_profile.json.gz to ./text_models/dnn/
train/plugins/profile/2021_11_03_13_49_14/tensorflow-2-6-20211103-092159.memory_
profile.json.gz
2021-11-03 13:49:15.020215: I tensorflow/core/profiler/rpc/client/capture_profil
```

```
e.cc:251] Creating directory: ./text_models/dnn/train/plugins/profile/2021_11_03
_13_49_14
Dumped tool data for xplane.pb to ./text_models/dnn/train/plugins/profile/2021_1
1_03_13_49_14/tensorflow-2-6-20211103-092159.xplane.pb
Dumped tool data for overview_page.pb to ./text_models/dnn/train/plugins/profil
e/2021_11_03_13_49_14/tensorflow-2-6-20211103-092159.overview_page.pb
Dumped tool data for input_pipeline.pb to ./text_models/dnn/train/plugins/profil
e/2021_11_03_13_49_14/tensorflow-2-6-20211103-092159.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./text_models/dnn/train/plugins/prof
ile/2021_11_03_13_49_14/tensorflow-2-6-20211103-092159.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./text_models/dnn/train/plugins/profile/
2021_11_03_13_49_14/tensorflow-2-6-20211103-092159.kernel_stats.pb

257/257 [==============================] - 3s 10ms/step - loss: 1.0474 - accurac
y: 0.4302 - val_loss: 0.9802 - val_accuracy: 0.5990
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 26, 10)            472720
_________________________________________________________________
lambda (Lambda)              (None, 10)                0
_________________________________________________________________
dense (Dense)                (None, 3)                 33
=================================================================
Total params: 472,753
Trainable params: 472,753
Non-trainable params: 0
_________________________________________________________________
CPU times: user 5.75 s, sys: 5.33 s, total: 11.1 s
Wall time: 3.52 s
```





Building a RNN model

The `build_dnn_model` function below returns a compiled Keras model that implements a simple RNN model with a single `GRU` layer, which now takes into account the word order in the sentence.

The first and last layers are the same as for the simple DNN model.

Note that we set `mask_zero=True` in the `Embedding` layer so that the padded words (represented by a zero) are ignored by this and the subsequent layers.

In [39]:

```
def build_rnn_model(embed_dim, units):

    model = Sequential([
        Embedding(VOCAB_SIZE + 1, embed_dim, input_shape=[MAX_LEN], mask_zero=Tr
        GRU(units), # TODO 5
        Dense(N_CLASSES, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Let's train the model with early stoping as above.

Observe that we obtain the same type of accuracy as with the DNN model, but in less epochs (~3 v.s. ~20 epochs):

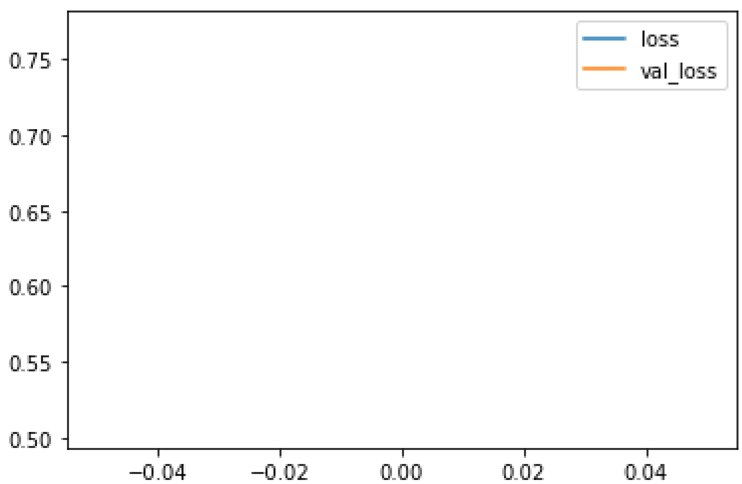
In [40]:

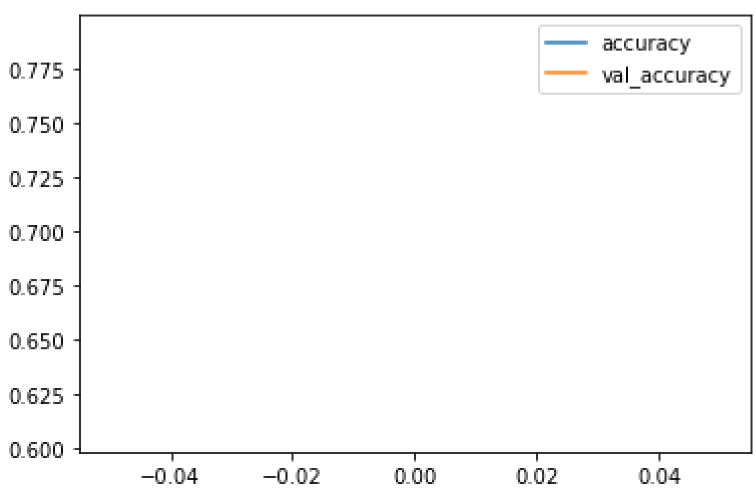
```
%%time

tf.random.set_seed(33)

MODEL_DIR = os.path.join(LOGDIR, 'rnn')
shutil.rmtree(MODEL_DIR, ignore_errors=True)

EPOCHS = 100
```

```
BATCH_SIZE = 300
EMBED_DIM = 10
UNITS = 16
PATIENCE = 0

rnn_model = build_rnn_model(embed_dim=EMBED_DIM, units=UNITS)

history = rnn_model.fit(
    X_train, Y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=(X_valid, Y_valid),
    callbacks=[EarlyStopping(patience=PATIENCE), TensorBoard(MODEL_DIR)],
)

pd.DataFrame(history.history)[['loss', 'val_loss']].plot()
pd.DataFrame(history.history)[['accuracy', 'val_accuracy']].plot()

rnn_model.summary()
```

```
2021-11-03 13:49:51.169497: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-11-03 13:49:51.169554: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-11-03 13:49:51.169600: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
Epoch 1/100
  2/257 [..............................] - ETA: 1:01 - loss: 1.0981 - accuracy:
0.3500
2021-11-03 13:49:54.975723: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-11-03 13:49:54.975762: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
  3/257 [..............................] - ETA: 1:07 - loss: 1.0974 - accuracy:
0.3511
2021-11-03 13:49:55.197045: I tensorflow/core/profiler/lib/profiler_session.cc:6
6] Profiler session collecting data.
2021-11-03 13:49:55.217602: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
2021-11-03 13:49:55.248193: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: ./text_models/rnn/train/plugins/profile/2021_11_03_13
_49_55

2021-11-03 13:49:55.267932: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for trace.json.gz to ./text_models/rnn/train/plu
gins/profile/2021_11_03_13_49_55/tensorflow-2-6-20211103-092159.trace.json.gz
2021-11-03 13:49:55.298431: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: ./text_models/rnn/train/plugins/profile/2021_11_03_13
_49_55

2021-11-03 13:49:55.303550: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for memory_profile.json.gz to ./text_models/rnn/
train/plugins/profile/2021_11_03_13_49_55/tensorflow-2-6-20211103-092159.memory_
profile.json.gz
2021-11-03 13:49:55.304454: I tensorflow/core/profiler/rpc/client/capture_profil
e.cc:251] Creating directory: ./text_models/rnn/train/plugins/profile/2021_11_03
_13_49_55
Dumped tool data for xplane.pb to ./text_models/rnn/train/plugins/profile/2021_1
1_03_13_49_55/tensorflow-2-6-20211103-092159.xplane.pb
Dumped tool data for overview_page.pb to ./text_models/rnn/train/plugins/profil
```

```
e/2021_11_03_13_49_55/tensorflow-2-6-20211103-092159.overview_page.pb
Dumped tool data for input_pipeline.pb to ./text_models/rnn/train/plugins/profil
e/2021_11_03_13_49_55/tensorflow-2-6-20211103-092159.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./text_models/rnn/train/plugins/prof
ile/2021_11_03_13_49_55/tensorflow-2-6-20211103-092159.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./text_models/rnn/train/plugins/profile/
2021_11_03_13_49_55/tensorflow-2-6-20211103-092159.kernel_stats.pb

257/257 [==============================] - 17s 51ms/step - loss: 0.7688 - accura
cy: 0.6078 - val_loss: 0.5064 - val_accuracy: 0.7907
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 26, 10)            472720
_________________________________________________________________
gru (GRU)                    (None, 16)                1344
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 51
=================================================================
Total params: 474,115
Trainable params: 474,115
Non-trainable params: 0
_________________________________________________________________
CPU times: user 29.2 s, sys: 24.3 s, total: 53.5 s
Wall time: 23.8 s
```





Build a CNN model

The `build_dnn_model` function below returns a compiled Keras model that implements a simple CNN model with a single `Conv1D` layer, which now takes into account the word order in the sentence.

The first and last layers are the same as for the simple DNN model, but we need to add a `Flatten` layer betwen the convolution and the softmax layer.

Note that we set `mask_zero=True` in the `Embedding` layer so that the padded words (represented by a zero) are ignored by this and the subsequent layers.

In [41]:

```
def build_cnn_model(embed_dim, filters, ksize, strides):

    model = Sequential([
        Embedding(
            VOCAB_SIZE + 1,
            embed_dim,
            input_shape=[MAX_LEN],
            mask_zero=True), # TODO 3
        Conv1D( # TODO 5
            filters=filters,
            kernel_size=ksize,
            strides=strides,
            activation='relu',
        ),
        Flatten(), # TODO 5
        Dense(N_CLASSES, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Let's train the model.

Again we observe that we get the same kind of accuracy as with the DNN model but in many fewer steps.

In [42]:

```
%%time

tf.random.set_seed(33)

MODEL_DIR = os.path.join(LOGDIR, 'cnn')
shutil.rmtree(MODEL_DIR, ignore_errors=True)

EPOCHS = 100
BATCH_SIZE = 300
EMBED_DIM = 5
FILTERS = 200
STRIDES = 2
KSIZE = 3
PATIENCE = 0
```

```
cnn_model = build_cnn_model(
    embed_dim=EMBED_DIM,
    filters=FILTERS,
    strides=STRIDES,
    ksize=KSIZE,
)

cnn_history = cnn_model.fit(
    X_train, Y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=(X_valid, Y_valid),
    callbacks=[EarlyStopping(patience=PATIENCE), TensorBoard(MODEL_DIR)],
)

pd.DataFrame(cnn_history.history)[['loss', 'val_loss']].plot()
pd.DataFrame(cnn_history.history)[['accuracy', 'val_accuracy']].plot()

cnn_model.summary()
```

```
2021-11-03 13:50:37.786026: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-11-03 13:50:37.786067: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-11-03 13:50:37.786101: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
Epoch 1/100
  3/257 [..............................] - ETA: 29s - loss: 1.0973 - accuracy:
0.3544
2021-11-03 13:50:38.348274: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-11-03 13:50:38.348624: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-11-03 13:50:38.507176: I tensorflow/core/profiler/lib/profiler_session.cc:6
6] Profiler session collecting data.
2021-11-03 13:50:38.508442: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
2021-11-03 13:50:38.509956: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: ./text_models/cnn/train/plugins/profile/2021_11_03_13
_50_38

2021-11-03 13:50:38.510812: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for trace.json.gz to ./text_models/cnn/train/plu
gins/profile/2021_11_03_13_50_38/tensorflow-2-6-20211103-092159.trace.json.gz
2021-11-03 13:50:38.513284: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: ./text_models/cnn/train/plugins/profile/2021_11_03_13
_50_38

2021-11-03 13:50:38.513722: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for memory_profile.json.gz to ./text_models/cnn/
train/plugins/profile/2021_11_03_13_50_38/tensorflow-2-6-20211103-092159.memory_
profile.json.gz
2021-11-03 13:50:38.514126: I tensorflow/core/profiler/rpc/client/capture_profil
e.cc:251] Creating directory: ./text_models/cnn/train/plugins/profile/2021_11_03
_13_50_38
Dumped tool data for xplane.pb to ./text_models/cnn/train/plugins/profile/2021_1
1_03_13_50_38/tensorflow-2-6-20211103-092159.xplane.pb
Dumped tool data for overview_page.pb to ./text_models/cnn/train/plugins/profil
e/2021_11_03_13_50_38/tensorflow-2-6-20211103-092159.overview_page.pb
Dumped tool data for input_pipeline.pb to ./text_models/cnn/train/plugins/profil
```

```
e/2021_11_03_13_50_38/tensorflow-2-6-20211103-092159.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./text_models/cnn/train/plugins/prof
ile/2021_11_03_13_50_38/tensorflow-2-6-20211103-092159.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./text_models/cnn/train/plugins/profile/
2021_11_03_13_50_38/tensorflow-2-6-20211103-092159.kernel_stats.pb

257/257 [==============================] - 7s 25ms/step - loss: 0.6707 - accurac
y: 0.6838 - val_loss: 0.4234 - val_accuracy: 0.8263
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 26, 5)             236360
_________________________________________________________________
conv1d (Conv1D)              (None, 12, 200)           3200
_________________________________________________________________
flatten (Flatten)            (None, 2400)              0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 7203
=================================================================
Total params: 246,763
Trainable params: 246,763
Non-trainable params: 0
_________________________________________________________________
CPU times: user 16 s, sys: 10.1 s, total: 26.1 s
Wall time: 7.15 s
```