Summary

- Apply transfer learning with BERT on BBC news data, **classifying news** into Sports, Tech, Business, Entertainment, and Politics
- The model will be used with **tensorflow serving on AWS**
- In production, the model shall not only classify but also handle **all of the pre-processing** of incoming text data
- A **custom training loop** with **gradient tape** is used, to have more control over training
- During training, while the GPU trains the current batch, the CPU shall already pre-process the next input
- The model with the best validation accuracy is saved and used later on
- Train and validation steps use **graph execution** to speed up training
- Tensorboard is used visualize training and validation accuracy, as well as show the distribution of some variable's Adam learning rates across the epochs

Introduction

In this notebook, we use transfer learning on a BERT-network to classify BBC news articles into sports, tech, business, entertainment, and politics.

The finished model will be used to serve requests on AWS using tensorflow serving and docker. The API in front of the tf-container shall **not** transform the posted data in any way, **thus the pre-processing of text shall be part of the model**. This should reduce user's latency.

But let's start with some simple data analysis first.

Some code snippet were copied and transformed from this official example.

Get the data. Raw text files are used here.

In [ ]:
```
!unzip bbc-fulltext.zip
```

```
Archive:  bbc-fulltext.zip
replace bbc/business/001.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
```

In [ ]:
```
!pip install -q -U tensorflow-text
```

```
     |████████████████████████████████| 4.9 MB 9.9 MB/s
```

In [ ]:
```
import os
import fnmatch

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
import tensorflow_text as text
import numpy as np
import matplotlib.pyplot as plt
```

```
import datetime

tf.get_logger().setLevel('ERROR')
```

In [ ]:
```
tf.test.is_gpu_available()
```

Out[ ]:
```
True
```

Data Analysis

BERT models have pre-trained embeddings at their disposal. Thus for every received input there is an embedding. If a word is not known it is split down until only word fragments or single characters that can be assigned to an embedding are left.

However, in general BERT does not accept more than 512 input embeddings per input, and to speed up training the input should be set lower than 512.

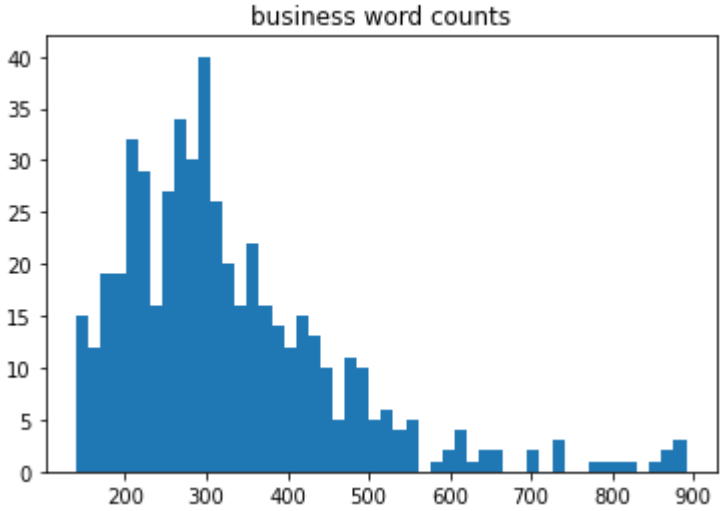
Hence, it makes sense to check and plot the word count of each topic.

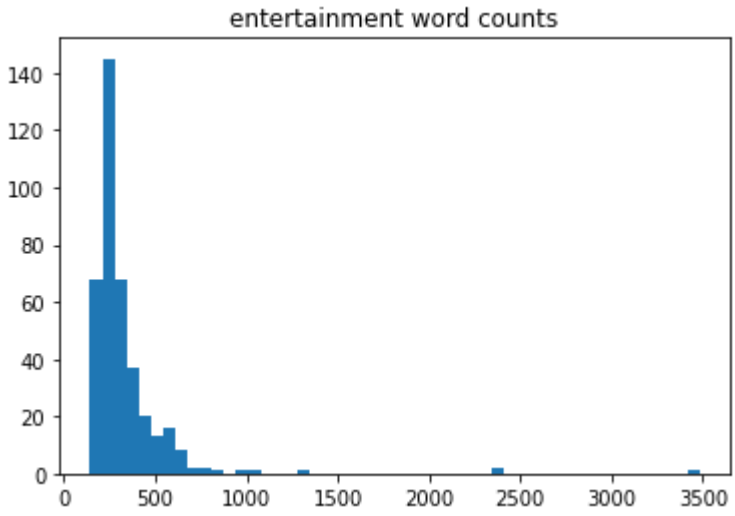
In [ ]:
```
labels = []
news = []
lengths = []
num_per_topic = {}
lngth_per_topic = {}
for label_type in ['business', 'entertainment', 'tech', 'sport', 'politics']:
    lngth_per_topic[label_type] = []
    dir_name = './bbc/' + label_type
    num_per_topic[label_type] = len(fnmatch.filter(os.listdir(dir_name), '*.txt'))
    print(label_type, num_per_topic[label_type])
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding='utf-8', errors='ignore')
            news.append(f.read())
            f.close()
            lengths.append(len(news[-1].split()))
        else:
            continue
        if label_type == 'business':
            labels.append(0)
            lngth_per_topic[label_type].append(len(news[-1].split()))
        elif label_type == 'entertainment':
            labels.append(1)
            lngth_per_topic[label_type].append(len(news[-1].split()))
        elif label_type == 'tech':
            labels.append(2)
            lngth_per_topic[label_type].append(len(news[-1].split()))
        elif label_type == 'sport':
            labels.append(3)
            lngth_per_topic[label_type].append(len(news[-1].split()))
        elif label_type == 'politics':
            labels.append(4)
            lngth_per_topic[label_type].append(len(news[-1].split()))
print('Total number of news:', len(news))
print('Most words in a single document', max(lengths))
print('\n')
for key, val in lngth_per_topic.items():
    print('Most words per topic ' + key, max(val))
```

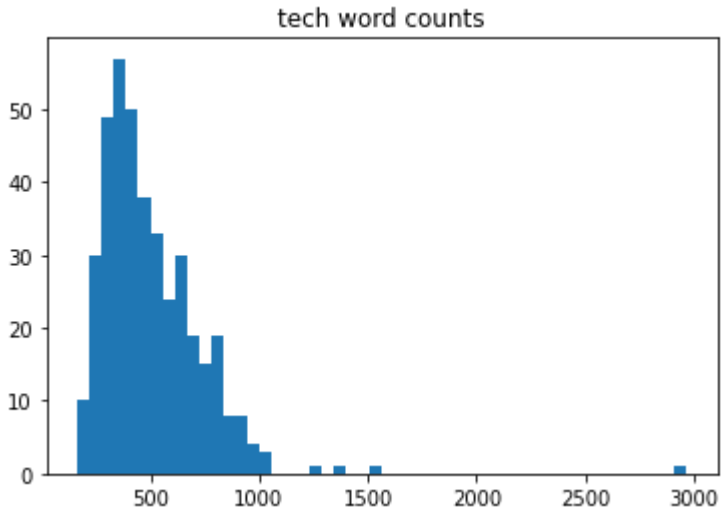
```
    print('Average # words per topic ' + key, round(np.mean(val)))
    print('\n')
```

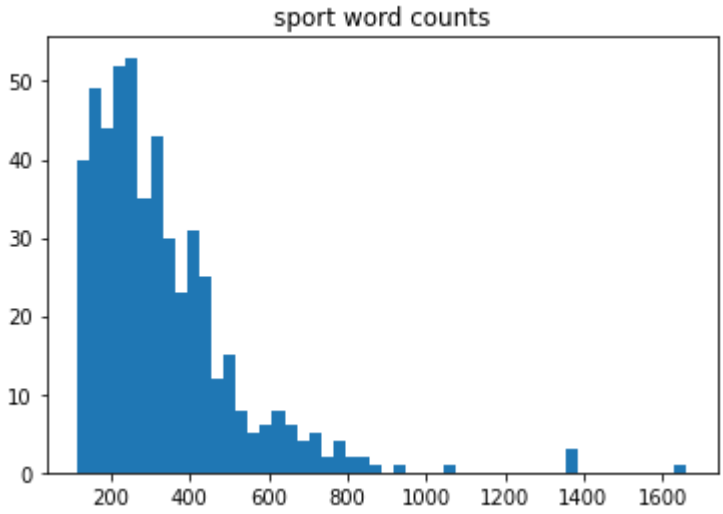
```
business 510
entertainment 386
tech 401
sport 511
politics 417
Total number of news: 2225
Most words in a single document 4432


Most words per topic business 891
Average # words per topic business 329


Most words per topic entertainment 3482
Average # words per topic entertainment 331


Most words per topic tech 2969
Average # words per topic tech 503


Most words per topic sport 1662
Average # words per topic sport 329


Most words per topic politics 4432
Average # words per topic politics 454
```

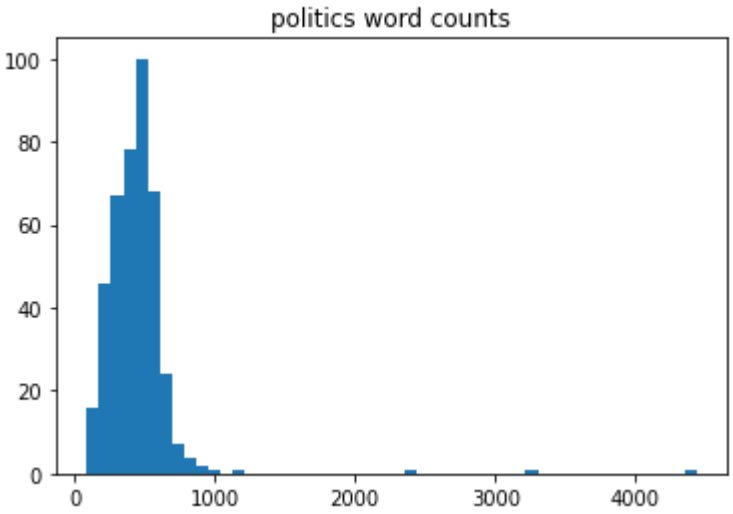
In [ ]:
```
for key, val in lngth_per_topic.items():
    plt.title(key + ' word counts')
    plt.hist(val, bins=50)
    plt.show()
```




entertainment word counts
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500


tech word counts
50
40
30
20
10
0
500
1000
1500
2000
2500
3000


sport word counts
50
40
30
20
10
0
200
400
600
800
1000
1200
1400
1600



Most texts are in the 200-400 words range, which should work well with BERT.

The trimming of too long texts and choosing of the final input size will be done later on.

Tensorflow Datasets

Luckily, keras has functions that make it very simple to load the txt-files and their corresponding class labels into tensorflow datasets

The data will be loaded in batches.

Caching loads the data into the cache.

Prefetching will be used to start pre-processing while the prior batch is in training. AUTOTUNE will determine the size of the prefetch.

In [ ]:
```
AUTOTUNE = tf.data.AUTOTUNE
batch_size = 16
seed = 42

raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    './bbc',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)

raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    './bbc',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)

train_ds = raw_train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = raw_val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
Found 2225 files belonging to 5 classes.
Using 1780 files for training.
Found 2225 files belonging to 5 classes.
Using 445 files for validation.
```

Let's check whether the labels are similarly distributed in both datasets.

In [ ]:
```
train_labels = []
for _, label_batch in raw_train_ds.unbatch().take(1780):
    train_labels.append(label_batch.numpy())
```

In [ ]:
```
val_labels = []
for _, label_batch in raw_val_ds.unbatch().take(445):
    val_labels.append(label_batch.numpy())
```

In [ ]:
```
print('train label distribution:', np.unique(train_labels, return_counts=True))
```

```
train label distribution: (array([0, 1, 2, 3, 4], dtype=int32), array([419, 307, 329,
413, 312]))
```

In [ ]:
```
print('val label distribution:', np.unique(val_labels, return_counts=True))
```

```
val label distribution: (array([0, 1, 2, 3, 4], dtype=int32), array([91, 79, 88, 98,
89]))
```

Looks good.

The BERT Model

We will use the small_bert/bert_en_uncased_L-4_H-512_A-8. It consists of 4 transformer blocks, a hidden/embedding size of 512, and 8 different heads per transformer block.

We also make use of a BERT preprocessing model. It's a Keras model without trainable parameters and an easy way to integrate the pre-processing as part of the model. In turn, it is also part of the servable model and the pre-processing of text in a live environment is not required to be separated from the model.

In [ ]:
```
#@title Choose a BERT model to fine-tune

bert_model_name = 'small_bert/bert_en_uncased_L-4_H-512_A-8'  #@param ["bert_en_uncase

map_name_to_handle = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_L-12_H-768_A-12/3',
    'bert_multi_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/3',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/1',
```

```
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-6_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-6_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-6_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-6_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-8_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-8_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-8_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-8_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-10_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-10_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-10_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-10_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-12_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-12_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-12_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-768_A-12/1',
    'albert_en_base':
        'https://tfhub.dev/tensorflow/albert_en_base/2',
    'electra_small':
        'https://tfhub.dev/google/electra_small/2',
    'electra_base':
        'https://tfhub.dev/google/electra_base/2',
    'experts_pubmed':
        'https://tfhub.dev/google/experts/bert/pubmed/2',
    'experts_wiki_books':
        'https://tfhub.dev/google/experts/bert/wiki_books/2',
    'talking-heads_base':
        'https://tfhub.dev/tensorflow/talkheads_ggelu_bert_en_base/1',
}

map_model_to_preprocess = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
```

```
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-6_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-6_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-6_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-6_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-8_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-8_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-8_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-8_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-10_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-10_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-10_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-10_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-12_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-12_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-12_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'small_bert/bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'bert_multi_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_multi_cased_preprocess/3',
    'albert_en_base':
        'https://tfhub.dev/tensorflow/albert_en_preprocess/3',
    'electra_small':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'electra_base':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'experts_pubmed':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'experts_wiki_books':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
    'talking-heads_base':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3',
}
```

```
tfhub_handle_encoder = map_name_to_handle[bert_model_name]
tfhub_handle_preprocess = map_model_to_preprocess[bert_model_name]

print(f'BERT model selected           : {tfhub_handle_encoder}')
print(f'Preprocess model auto-selected: {tfhub_handle_preprocess}')
```

```
BERT model selected           : https://tfhub.dev/tensorflow/small_bert/bert_en_uncas
ed_L-4_H-512_A-8/1
Preprocess model auto-selected: https://tfhub.dev/tensorflow/bert_en_uncased_preproce
ss/3
```

Let's see how the pre-processing transforms a given input.

In [ ]:
```
bert_preprocess = hub.load(tfhub_handle_preprocess)
processed = bert_preprocess.tokenize(tf.constant([news[0]]))
print(len(processed.to_tensor().numpy()[0]))
```

```
873
```

processed is a tf.RaggedTensor that holds the BERT-embedding ids of the first news article, without start and end tokens.

In [ ]:
```
processed
```

Out[ ]:
```
<tf.RaggedTensor [[[8956], [7206], [2735], [2000], [3171], [3314], [3458], [1996], [7
143], [3036], [3663], [1999], [5712], [3658], [2019], [4610], [1999], [11937, 24168],
[1012], [1037], [13925], [5402], [1997], [12163], [1010], [3532], [2591], [2578], [19
98], [5635], [2038], [2042], [2081], [4788], [2011], [1037], [3768], [1997], [5211],
[1012], [2061], [2045], [2003], [2172], [3246], [2008], [2019], [2700], [2231], [209
7], [3338], [1996], [2757, 7878], [1012], [1000], [2034], [3627], [1997], [2375], [10
10], [2059], [1996], [4610], [1010], [1000], [2758], [10958, 2094, 7447], [2018, 207
2], [1010], [4112], [6605], [2472], [1997], [12080], [1011], [2241], [3514], [1998],
[3806], [24853], [2304, 18866], [11540], [2578], [1010], [2029], [3133], [5712], [199
9], [2494], [1012], [2720], [2018, 2072], [1005], [1055], [3193], [2055], [2054], [19
96], [2047], [2231], [1005], [1055], [18402], [2323], [2022], [2003], [4207], [2011],
[2116], [8956, 2015], [1012], [1996], [4610], [2038], [2468], [1996], [2117], [1011],
[2087], [7444], [3277], [2005], [2116], [2576], [4243], [3805], [1997], [4465], [100
5], [1055], [2602], [1010], [2429], [2000], [7067], [2118], [2576], [7155], [4776],
[3656], [1010], [2040], [2003], [2551], [2006], [1037], [2622], [2008], [3504], [201
2], [10615], [1998], [3036], [1999], [2695], [1011], [2162], [5712], [1012], [3105],
[4325], [6938], [2152], [2119], [2006], [2602], [17124, 2015], [1998], [2006], [199
6], [8956], [2111], [1005], [1055], [4299], [2862], [1012], [6343], [4282], [3599],
[2129], [2116], [8956, 2015], [2024], [2041], [1997], [2147], [1010], [2021], [2009],
[2003], [3154], [2008], [1996], [3663], [2003], [18704], [1012], [1000], [10035], [19
97], [5712], [1005], [1055], [12163], [3446], [8137], [1010], [2021], [2057], [1019
7], [2009], [2000], [2022], [2090], [2382], [1011], [2871], [1003], [1010], [1000],
[1996], [2899], [1011], [2241], [2981], [2228], [1011], [4951], [1996], [9566, 8613],
[5145], [2758], [1999], [2049], [5712], [5950], [1012], [2021], [2070], [5082], [203
8], [2042], [2081], [1010], [4321], [4283], [2000], [1996], [2406], [1005], [1055],
[3514], [12594], [2029], [2031], [14872], [1002], [2570, 24700], [2144], [2238], [249
4], [1012], [5712], [1005], [1055], [6502], [2003], [2006], [1996], [2273, 2094], [10
10], [2007], [3862], [8377], [2383], [2042], [2081], [1999], [2752], [2107], [2004],
[6451], [4425], [1010], [12442], [1010], [7026], [6125], [1998], [1996], [2128], [101
1], [3098], [1997], [8323], [1012], [2021], [3809], [3471], [3961], [1998], [1996],
[3652], [11443], [2090], [2031, 2015], [1998], [2031], [1011], [2025, 2015], [2003],
[4963, 2075], [7206], [1012], [2028], [8956], [2450], [2409], [5796], [3656], [2055],
[2014], [9135], [2004], [2016], [3427], [2694], [4748, 16874, 2015], [2005], [2797],
[8323], [2574], [2044], [2383], [3478], [2000], [2650], [2091], [3937], [20233], [201
3], [13952], [1005], [1055], [6887, 27292, 20499], [1012], [24451], [2720], [2018, 20
72], [1024], [1000], [1996], [4610], [2012], [2556], [6017], [1037], [2502], [11443],
[1025], [1996], [4138], [2131], [26108], [1010], [1996], [3532], [2131], [27196], [10
12], [1000], [2019], [12407], [1997], [2023], [2064], [2022], [2464], [1999], [1996],
[2088], [1997], [5446], [2073], [1010], [1999], [5688], [2007], [1996], [3679], [2452
5], [1997], [6623], [2111], [1010], [2539], [2797], [5085], [5452], [1010], [2069],
[2028], [1997], [2029], [2003], [2448], [1999], [10388], [2007], [5499], [8169], [648
1], [1012], [8069], [2024], [2152], [2005], [1996], [2925], [1997], [5446], [1010],
[2061], [3097], [5085], [2031], [2042], [9343], [2046], [1996], [4753], [1012], [212
0], [2924], [1997], [13085], [2038], [4149], [1037], [3484], [8406], [1999], [4923],
[2924], [1997], [5712], [1010], [1996], [26276], [5211], [2924], [9167], [1004], [544
6], [2924], [2038], [4149], [4749], [1003], [1997], [2120], [2924], [1997], [5712],
[1012], [3097], [9786], [2036], [3246], [2000], [5356], [1999], [2006], [1996], [873
5], [3947], [1012], [2022, 10143, 2884], [1005], [1055], [4073], [2000], [14591], [28
16], [1998], [9239], [2373], [2031], [6296], [6704], [2004], [2092], [2004], [12992,
2075], [2049], [3953], [2240], [2096], [2534, 12322, 19585, 2239], [2038], [5632], [1
037], [7177], [1997], [2510], [8311], [1012], [2021], [1996], [6624], [1997], [3097],
[9786], [1999], [1996], [2740], [1998], [8169], [11105], [1998], [3458], [7719], [166
55, 21369, 2135], [2007], [2116], [8956, 2015], [2040], [2024], [17730], [2000], [199
6], [2110], [2635], [5368], [2005], [4972], [2008], [2024], [6827], [2000], [2437],
[2554], [2147], [1010], [24451], [5796], [3656], [1012], [1000], [2009], [2003], [246
4], [2004], [1037], [4855], [2125], [1997], [5712], [1005], [1055], [7045], [1998],
[5026], [1999], [20584, 2015], [2012], [1996], [10961], [1997], [8956], [5661], [199
8], [8956], [3667], [1010], [1000], [2016], [2758], [1012], [8821], [1010], [1996],
[17459], [2231], [2038], [2042], [3140], [2000], [2067, 6494, 3600], [1999], [3522],
[2706], [2058], [2049], [6378], [2000], [3499], [2531], [1003], [3097], [6095], [199
7], [8956], [7045], [1010], [2016], [7607], [1012], [1999], [1996], [2225], [1010],
```

```
[2009], [2003], [3733], [2000], [5293], [2008], [1996], [4728], [12077], [8670, 2545
7, 3367], [6939], [2109], [2000], [2298], [2044], [1996], [3484], [1997], [5712], [10
05], [1055], [4480], [2738], [2092], [1999], [3408], [1997], [3105], [4325], [1010],
[2591], [3036], [1998], [9871], [1012], [5448], [14592], [6592], [2008], [1000], [211
1], [2145], [2215], [1996], [2110], [2000], [2202], [1037], [2877], [2535], [1999],
[4346], [2122], [2477], [1000], [1010], [5796], [3656], [2758], [1012], [2664], [199
9], [2070], [2752], [1997], [1996], [4610], [1010], [5211], [2013], [6917], [2003],
[2145], [22775], [10979], [1010], [16818], [2720], [2018, 2072], [1010], [2019], [895
6], [2040], [2187], [1996], [2406], [2093], [5109], [3283], [1012], [1000], [1045],
[2228], [1996], [2797], [4753], [2097], [19852], [11757], [3435], [1010], [1000], [27
20], [2018, 2072], [2758], [1012], [1000], [5712], [1005], [1055], [6565], [3019], [4
219], [2064], [2490], [2151], [10194], [1997], [3171], [3930], [1012], [1000], [211
6], [3097], [3316], [2360], [2027], [2024], [10326], [2000], [2131], [1999], [2006],
[1996], [2552], [1010], [2664], [2261], [2024], [2941], [5738], [1996], [2406], [199
9], [2151], [15902], [2126], [1012], [2021], [2045], [2024], [11790], [1012], [2720],
[2018, 2072], [1005], [1055], [2304, 18866], [2003], [2074], [2028], [1997], [2116],
[2235], [9224], [8225], [2005], [1037], [2172], [7046], [2925], [1012], [2304, 1886
6], [1005], [1055], [13952], [1011], [2241], [8727], [11684], [2177], [2038], [9877],
[1997], [2111], [2551], [2005], [2009], [2408], [1996], [2406], [1999], [11332, 696
8], [1998], [13952], [1010], [1998], [2049], [6145], [1998], [20248], [1011], [6529],
[2147], [2007], [1996], [8956], [3514], [3757], [2000], [8691], [2041], [2974], [465
1], [3314], [1010], [2720], [2018, 2072], [2685], [2041], [1012], [1000], [2122], [43
64], [2024], [2667], [2000], [2147], [1012], [1996], [8956], [2449], [2111], [2097],
[2079], [2449], [2012], [2035], [2335], [1012], [1000], [2166], [3632], [2006], [199
9], [5712], [1010], [1996], [2111], [2202], [5368], [1010], [2027], [2215], [2000],
[2444], [3671], [3268], [1012], [1000]]]>
```

Bert is actually fed 3 tensors, the embedding or word ids including start and end token, the input mask telling the model which tokens are padded, the type ids which are only useful for text pair classification from my understanding.

In [ ]:
```
text_preprocessed = bert_preprocess.bert_pack_inputs([processed], tf.constant(50))

print('Shape Word Ids : ', text_preprocessed['input_word_ids'].shape)
print('Word Ids       : ', text_preprocessed['input_word_ids'][0, :])
print('Shape Mask     : ', text_preprocessed['input_mask'].shape)
print('Input Mask     : ', text_preprocessed['input_mask'][0, :])
print('Shape Type Ids : ', text_preprocessed['input_type_ids'].shape)
print('Type Ids       : ', text_preprocessed['input_type_ids'][0, :])
```

```
Shape Word Ids :  (1, 50)
Word Ids       :  tf.Tensor(
[  101  8956  7206  2735  2000  3171  3314  3458  1996  7143  3036  3663
  1999  5712  3658  2019  4610  1999 11937 24168  1012  1037 13925  5402
  1997 12163  1010  3532  2591  2578  1998  5635  2038  2042  2081  4788
  2011  1037  3768  1997  5211  1012  2061  2045  2003  2172  3246  2008
  2019   102], shape=(50,), dtype=int32)
Shape Mask     :  (1, 50)
Input Mask     :  tf.Tensor(
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1], shape=(50,), dtype=int32)
Shape Type Ids :  (1, 50)
Type Ids       :  tf.Tensor(
[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 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0], shape=(50,), dtype=int32)
```

Here we declare the preprocessing model. We create an input layer and make use of the BERT preprocessing model including its tokenizer. The outputs of the model are processed inputs ready to be sent to a transformer block.

Additional text preprocessing can be included in this step.

Note that each step is from a KerasLayer which enables us to integrate all these steps directly into the model.

seq_length determines the number of tokens per instance that are fed into BERT. BERT's maximum are 512 but to speed up training a lower number should be used.

The original pre-processing function comes from here.

In [ ]:
```
def make_bert_preprocess_model(seq_length=128):
  """Returns Model mapping string features to BERT inputs.

  Args:
    sentence_features: a list with the names of string-valued features.
    seq_length: an integer that defines the sequence length of BERT inputs.

  Returns:
    A Keras Model that can be called on a list or dict of string Tensors
    (with the order or names, resp., given by sentence_features) and
    returns a dict of tensors for input to BERT.
  """

  input_segments = [
      tf.keras.layers.Input(shape=(), dtype=tf.string, name='input')
  ]

  bert_preprocess = hub.load(tfhub_handle_preprocess)
  tokenizer = hub.KerasLayer(bert_preprocess.tokenize, name='tokenizer')
  segments = [tokenizer(s) for s in input_segments]

  packer = hub.KerasLayer(bert_preprocess.bert_pack_inputs,
                          arguments=dict(seq_length=seq_length),
                          name='packer')
  model_inputs = packer(segments)
  return tf.keras.Model(input_segments, model_inputs)
```

In [ ]:
```
test_preprocess_model = make_bert_preprocess_model()
```

Let's apply the preprocessing model on each batch of the training and validaiton set.

In [ ]:
```
train_ds = train_ds.map(lambda x, y: (test_preprocess_model(x), y))
val_ds = val_ds.map(lambda x, y: (test_preprocess_model(x), y))
```

Finally we use the BERT model from above add a dropout layer with 10% chance of being dropped and finally a dense layer whose size depends on the number of classes set by us.

We reference the "pooled_output" which is the embedding of the [CLS] token summarizing the information of the final transformer block. It is of dimension 512.

The "sequence_output" would return the newly trained embeddings of the final transformer block of each input token and since we want to classify, we use the pooled one.

Dropout has a 10% of setting one of the 512 [CLS] inputs to zero, before they are fed to the dense layer.

In [ ]:
```
def build_classifier_model(num_classes):

  class Classifier(tf.keras.Model):
    def __init__(self, num_classes):
      super(Classifier, self).__init__(name="prediction")
      self.encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True)
      self.dropout = tf.keras.layers.Dropout(0.1)
      self.dense = tf.keras.layers.Dense(num_classes)

    def call(self, preprocessed_text):
      encoder_outputs = self.encoder(preprocessed_text)
      pooled_output = encoder_outputs["pooled_output"]
      x = self.dropout(pooled_output)
      x = self.dense(x)
      return x

  model = Classifier(num_classes)
  return model
```

In [ ]:
```
!mkdir -p ./logs
```

Here writer files are created for each metric that we want to record during training and visualize in Tensorboard later on.

Every variable has its own m and v value when using Adam as optimizer. m and v reference gradient estimates of the first moment and second moment, respectively.

In [ ]:
```
train_log_path = os.path.join('./logs', 'train', datetime.datetime.now().strftime("%Y%
val_log_path = os.path.join('./logs', 'val', datetime.datetime.now().strftime("%Y%m%d-
m_log_path = os.path.join('./logs', 'adam_m', datetime.datetime.now().strftime("%Y%m%d
v_log_path = os.path.join('./logs', 'adam_v', datetime.datetime.now().strftime("%Y%m%d

train_file_writer = tf.summary.create_file_writer(train_log_path)
val_file_writer = tf.summary.create_file_writer(val_log_path)
m_file_writer = tf.summary.create_file_writer(m_log_path)
v_file_writer = tf.summary.create_file_writer(v_log_path)
```

Now, we can train the model. Since this is a multiclassification problem we choosse SparseCategoricalCrossentropy as our loss.

Adam's an initial learning rate is 0.00002.

The data set is fairly small and even the small BERT is quite large, 3 epochs should be enough.

We also define a location to save the weights that score the highest validation score.

In [ ]:
```
epochs = 3
init_lr = 2e-5
train_data_size = 1780
val_data_size = 445
num_classes = 5
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model = build_classifier_model(num_classes)
optimizer = tf.keras.optimizers.Adam(learning_rate=init_lr)
train_acc_metric = tf.keras.metrics.SparseCategoricalAccuracy()
```

```
val_acc_metric = tf.keras.metrics.SparseCategoricalAccuracy()
temp_val_acc = 0
checkpoint_path = os.path.join('./checkpoints', "best_epoch_weights.ckpt")
```

In [ ]:
```
!mkdir ./checkpoints
```

The next two functions are defined as graphs using the tf.function decorator. Defining these function as graphs speeds up training in this case, but it has more benefits for more sophisticated use cases.

In [ ]:
```
@tf.function
def train_step(x, y):
    with tf.GradientTape() as tape:
        logits = model(x, training=True)
        loss_value = loss_fn(y, logits)
    grads = tape.gradient(loss_value, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
    train_acc_metric.update_state(y, logits)
    return loss_value
```

In [ ]:
```
@tf.function
def test_step(x, y):
    val_logits = model(x, training=False)
    val_loss_value = loss_fn(y, val_logits)
    val_acc_metric.update_state(y, val_logits)
    return val_loss_value
```

Custom Training Loop

The customer training loop:

- loops through each epoch
- loops through each training batch
- executes the train_step-graph with the training batch as input, records all gradients thourgh gradient tape and returns the loss of each batch
- Updates the train accuracy metric given the latest
- Logs pre-defined metrics for Tensorboard
- Applies the newest weights on the validation batches
- Saves the model with best validation accuracy

In [ ]:
```
for epoch in range(epochs):
    print("\nStart of epoch %d" % (epoch + 1,))

    # Iterate over the training batches of the dataset.
    for step, (x_batch_train, y_batch_train) in enumerate(train_ds):
        loss_value = train_step(x_batch_train, y_batch_train)

    train_acc = train_acc_metric.result()
    print("Training acc over epoch: %.4f" % (float(train_acc),))
    with train_file_writer.as_default():
        tf.summary.scalar('epoch_loss', loss_value, step=epoch + 1)
        tf.summary.scalar('epoch_accuracy', train_acc, step=epoch + 1)
    # Reset training metrics at the end of each epoch
```

```
    train_acc_metric.reset_states()

    with m_file_writer.as_default():
        for var in optimizer.variables():
            if var.name.startswith('Adam/prediction/dense') and  var.name.endswith('ke
                tf.summary.histogram('hist_dense_adam_first_moment', var.numpy(), step
            elif var.name.startswith('Adam/prediction/dense') and  var.name.endswith('
                tf.summary.histogram('hist_dense_adam_second_moment', var.numpy(), ste

    # Run a validation loop at the end of each epoch.
    for x_batch_val, y_batch_val in val_ds:
        val_loss_value = test_step(x_batch_val, y_batch_val)

    val_acc = val_acc_metric.result()
    with val_file_writer.as_default():
        tf.summary.scalar('epoch_loss', val_loss_value, step=epoch + 1)
        tf.summary.scalar('epoch_accuracy', val_acc, step=epoch + 1)

    val_acc_metric.reset_states()
    print("Validation acc: %.4f" % (float(val_acc),))
    if temp_val_acc < val_acc:
        print('New validation acc %.4f better than prior best %.4f' % (float(val_acc),
        temp_val_acc = val_acc
        model.save_weights(checkpoint_path)
        print('saved!')
    else:
        print('Validation acc did not improve - best %.4f' % (float(temp_val_acc)))
```

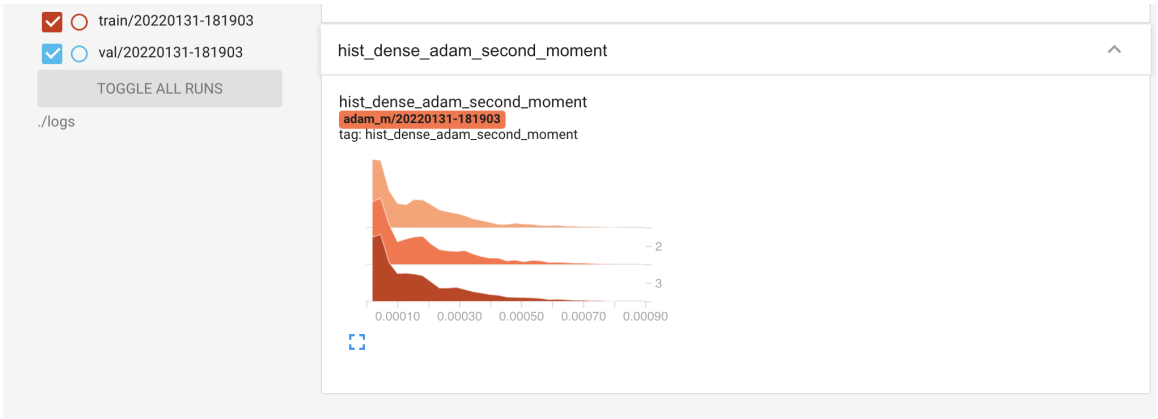
```
Start of epoch 1
Training acc over epoch: 0.8427
Validation acc: 0.9663
New validation acc 0.9663 better than prior best 0.0000
saved!

Start of epoch 2
Training acc over epoch: 0.9736
Validation acc: 0.9775
New validation acc 0.9775 better than prior best 0.9663
saved!

Start of epoch 3
Training acc over epoch: 0.9865
Validation acc: 0.9820
New validation acc 0.9820 better than prior best 0.9775
saved!
```

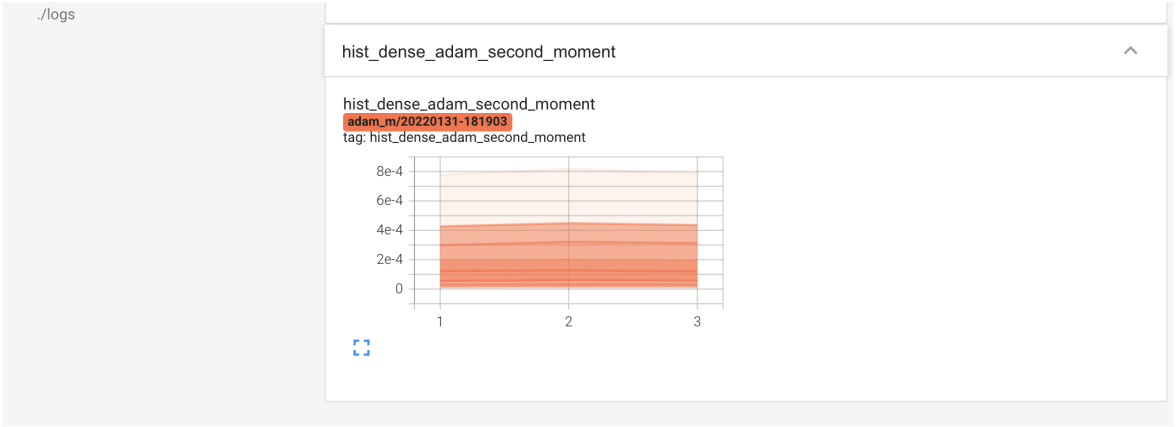
Tensorboard

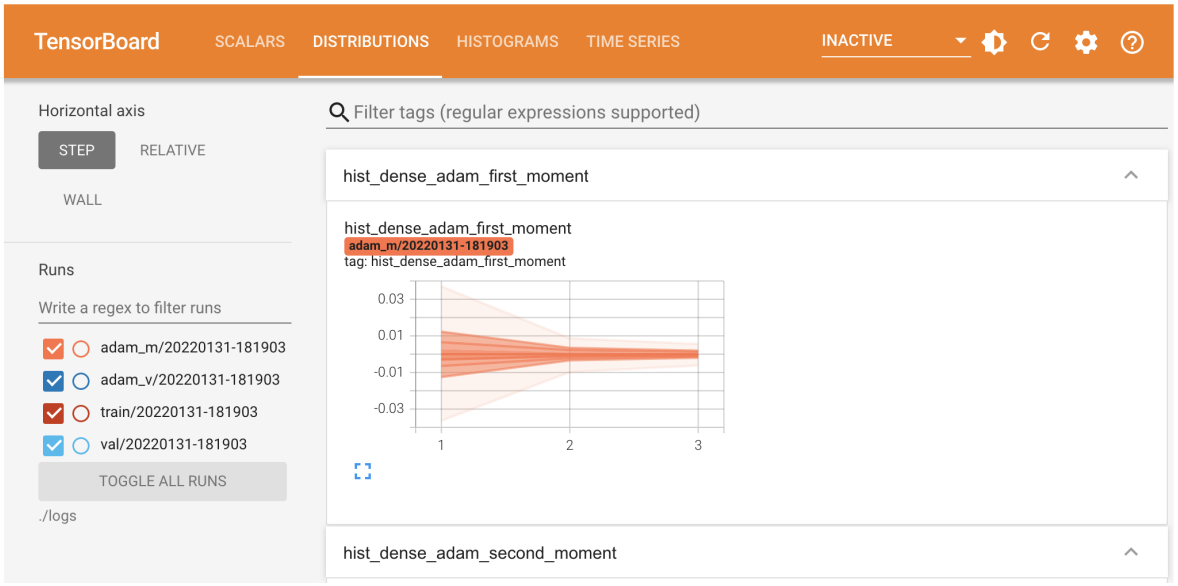
In [ ]:
```
%load_ext tensorboard
```

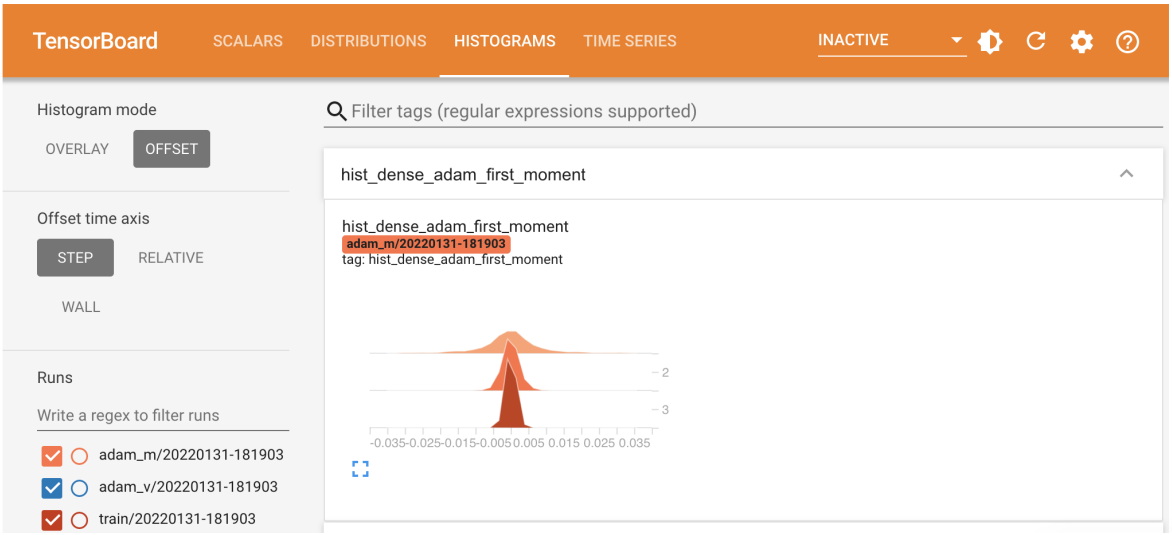
In [ ]:
```
%tensorboard --logdir ./logs
```

Unfortunately, Tensorboard visualizations are not exported when downloading an ipynb-file, hence I have screenshotted the distributions and histograms of a prior run for both approximate gradients of the dense layer.


train/20220131-181903
val/20220131-181903
TOGGLE ALL RUNS
./logs
hist_dense_adam_second_moment
hist_dense_adam_second_moment
adam_m/20220131-181903
tag: hist_dense_adam_second_moment
2
3
0.00010
0.00030
0.00050
0.00070
0.00090


./logs
hist_dense_adam_second_moment
hist_dense_adam_second_moment
adam_m/20220131-181903
tag: hist_dense_adam_second_moment
8e-4
6e-4
4e-4
2e-4
0
1
2
3





We can see that the distribution of first approximate gradients of the dense layer gets more centered around its mean with increasing epochs showing signs of conversion.

Exporting the model

In [ ]:
```
model = build_classifier_model(num_classes)
model.load_weights(checkpoint_path)
```

Out[ ]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f217f1bc690>
```

Let's concatenate both models, preprocessing and classification, together into one model, save it, zip it, and download it, so it can be used with tensorflow serving.

In [ ]:
```
#tf serve requires a version number, hence the 0001
saved_model_path = os.path.join('./bertbc', '0001')

preprocess_inputs = test_preprocess_model.inputs
bert_encoder_inputs = test_preprocess_model(preprocess_inputs)
bert_outputs = model(bert_encoder_inputs)
model_for_export = tf.keras.Model(preprocess_inputs, bert_outputs)
```

```
print('Saving', saved_model_path)

model_for_export.save(saved_model_path)
```

```
Saving ./bertbc/0001
WARNING:absl:Found untraced functions such as restored_function_body, restored_functi
on_body, restored_function_body, restored_function_body, restored_function_body while
saving (showing 5 of 310). These functions will not be directly callable after loadin
g.
```

In [ ]:
```
!zip -r bertbc.zip ./bertbc/
```

```
  adding: bertbc/ (stored 0%)
  adding: bertbc/0001/ (stored 0%)
  adding: bertbc/0001/saved_model.pb (deflated 92%)
  adding: bertbc/0001/variables/ (stored 0%)
  adding: bertbc/0001/variables/variables.data-00000-of-00001 (deflated 7%)
  adding: bertbc/0001/variables/variables.index (deflated 71%)
  adding: bertbc/0001/keras_metadata.pb (deflated 85%)
  adding: bertbc/0001/assets/ (stored 0%)
  adding: bertbc/0001/assets/vocab.txt (deflated 53%)
```

One quick test.

In [ ]:
```
model_for_export(np.array(["Germany have beaten England again in football by 3-0"]))
```

Out[ ]:
```
<tf.Tensor: shape=(1, 5), dtype=float32, numpy=
array([[-1.121467 , -1.9104604, -1.1305437,  5.552042 , -0.9230069]],
      dtype=float32)>
```

Appendix

All variables, their approximate gradients, and their shape can seen with the following lines. Helpful with debugging.

In [ ]:
```
for var in optimizer.variables():
    print(var.name, var.shape)
```

```
Adam/iter:0 ()
Adam/word_embeddings/embeddings/m:0 (30522, 512)
Adam/position_embedding/embeddings/m:0 (512, 512)
Adam/type_embeddings/embeddings/m:0 (2, 512)
Adam/embeddings/layer_norm/gamma/m:0 (512,)
Adam/embeddings/layer_norm/beta/m:0 (512,)
Adam/transformer/layer_0/self_attention/query/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_0/self_attention/query/bias/m:0 (8, 64)
Adam/transformer/layer_0/self_attention/key/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_0/self_attention/key/bias/m:0 (8, 64)
Adam/transformer/layer_0/self_attention/value/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_0/self_attention/value/bias/m:0 (8, 64)
Adam/transformer/layer_0/self_attention/attention_output/kernel/m:0 (8, 64, 512)
Adam/transformer/layer_0/self_attention/attention_output/bias/m:0 (512,)
Adam/transformer/layer_0/self_attention_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_0/self_attention_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_0/intermediate/kernel/m:0 (512, 2048)
Adam/transformer/layer_0/intermediate/bias/m:0 (2048,)
Adam/transformer/layer_0/output/kernel/m:0 (2048, 512)
Adam/transformer/layer_0/output/bias/m:0 (512,)
Adam/transformer/layer_0/output_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_0/output_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_1/self_attention/query/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_1/self_attention/query/bias/m:0 (8, 64)
Adam/transformer/layer_1/self_attention/key/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_1/self_attention/key/bias/m:0 (8, 64)
Adam/transformer/layer_1/self_attention/value/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_1/self_attention/value/bias/m:0 (8, 64)
Adam/transformer/layer_1/self_attention/attention_output/kernel/m:0 (8, 64, 512)
Adam/transformer/layer_1/self_attention/attention_output/bias/m:0 (512,)
Adam/transformer/layer_1/self_attention_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_1/self_attention_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_1/intermediate/kernel/m:0 (512, 2048)
Adam/transformer/layer_1/intermediate/bias/m:0 (2048,)
Adam/transformer/layer_1/output/kernel/m:0 (2048, 512)
Adam/transformer/layer_1/output/bias/m:0 (512,)
Adam/transformer/layer_1/output_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_1/output_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_2/self_attention/query/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_2/self_attention/query/bias/m:0 (8, 64)
Adam/transformer/layer_2/self_attention/key/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_2/self_attention/key/bias/m:0 (8, 64)
Adam/transformer/layer_2/self_attention/value/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_2/self_attention/value/bias/m:0 (8, 64)
Adam/transformer/layer_2/self_attention/attention_output/kernel/m:0 (8, 64, 512)
Adam/transformer/layer_2/self_attention/attention_output/bias/m:0 (512,)
Adam/transformer/layer_2/self_attention_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_2/self_attention_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_2/intermediate/kernel/m:0 (512, 2048)
Adam/transformer/layer_2/intermediate/bias/m:0 (2048,)
Adam/transformer/layer_2/output/kernel/m:0 (2048, 512)
Adam/transformer/layer_2/output/bias/m:0 (512,)
Adam/transformer/layer_2/output_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_2/output_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_3/self_attention/query/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_3/self_attention/query/bias/m:0 (8, 64)
Adam/transformer/layer_3/self_attention/key/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_3/self_attention/key/bias/m:0 (8, 64)
Adam/transformer/layer_3/self_attention/value/kernel/m:0 (512, 8, 64)
Adam/transformer/layer_3/self_attention/value/bias/m:0 (8, 64)
Adam/transformer/layer_3/self_attention/attention_output/kernel/m:0 (8, 64, 512)
```

```
Adam/transformer/layer_3/self_attention/attention_output/bias/m:0 (512,)
Adam/transformer/layer_3/self_attention_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_3/self_attention_layer_norm/beta/m:0 (512,)
Adam/transformer/layer_3/intermediate/kernel/m:0 (512, 2048)
Adam/transformer/layer_3/intermediate/bias/m:0 (2048,)
Adam/transformer/layer_3/output/kernel/m:0 (2048, 512)
Adam/transformer/layer_3/output/bias/m:0 (512,)
Adam/transformer/layer_3/output_layer_norm/gamma/m:0 (512,)
Adam/transformer/layer_3/output_layer_norm/beta/m:0 (512,)
Adam/pooler_transform/kernel/m:0 (512, 512)
Adam/pooler_transform/bias/m:0 (512,)
Adam/prediction/dense/kernel/m:0 (512, 5)
Adam/prediction/dense/bias/m:0 (5,)
Adam/word_embeddings/embeddings/v:0 (30522, 512)
Adam/position_embedding/embeddings/v:0 (512, 512)
Adam/type_embeddings/embeddings/v:0 (2, 512)
Adam/embeddings/layer_norm/gamma/v:0 (512,)
Adam/embeddings/layer_norm/beta/v:0 (512,)
Adam/transformer/layer_0/self_attention/query/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_0/self_attention/query/bias/v:0 (8, 64)
Adam/transformer/layer_0/self_attention/key/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_0/self_attention/key/bias/v:0 (8, 64)
Adam/transformer/layer_0/self_attention/value/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_0/self_attention/value/bias/v:0 (8, 64)
Adam/transformer/layer_0/self_attention/attention_output/kernel/v:0 (8, 64, 512)
Adam/transformer/layer_0/self_attention/attention_output/bias/v:0 (512,)
Adam/transformer/layer_0/self_attention_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_0/self_attention_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_0/intermediate/kernel/v:0 (512, 2048)
Adam/transformer/layer_0/intermediate/bias/v:0 (2048,)
Adam/transformer/layer_0/output/kernel/v:0 (2048, 512)
Adam/transformer/layer_0/output/bias/v:0 (512,)
Adam/transformer/layer_0/output_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_0/output_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_1/self_attention/query/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_1/self_attention/query/bias/v:0 (8, 64)
Adam/transformer/layer_1/self_attention/key/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_1/self_attention/key/bias/v:0 (8, 64)
Adam/transformer/layer_1/self_attention/value/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_1/self_attention/value/bias/v:0 (8, 64)
Adam/transformer/layer_1/self_attention/attention_output/kernel/v:0 (8, 64, 512)
Adam/transformer/layer_1/self_attention/attention_output/bias/v:0 (512,)
Adam/transformer/layer_1/self_attention_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_1/self_attention_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_1/intermediate/kernel/v:0 (512, 2048)
Adam/transformer/layer_1/intermediate/bias/v:0 (2048,)
Adam/transformer/layer_1/output/kernel/v:0 (2048, 512)
Adam/transformer/layer_1/output/bias/v:0 (512,)
Adam/transformer/layer_1/output_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_1/output_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_2/self_attention/query/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_2/self_attention/query/bias/v:0 (8, 64)
Adam/transformer/layer_2/self_attention/key/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_2/self_attention/key/bias/v:0 (8, 64)
Adam/transformer/layer_2/self_attention/value/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_2/self_attention/value/bias/v:0 (8, 64)
Adam/transformer/layer_2/self_attention/attention_output/kernel/v:0 (8, 64, 512)
Adam/transformer/layer_2/self_attention/attention_output/bias/v:0 (512,)
Adam/transformer/layer_2/self_attention_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_2/self_attention_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_2/intermediate/kernel/v:0 (512, 2048)
```

```
Adam/transformer/layer_2/intermediate/bias/v:0 (2048,)
Adam/transformer/layer_2/output/kernel/v:0 (2048, 512)
Adam/transformer/layer_2/output/bias/v:0 (512,)
Adam/transformer/layer_2/output_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_2/output_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_3/self_attention/query/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_3/self_attention/query/bias/v:0 (8, 64)
Adam/transformer/layer_3/self_attention/key/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_3/self_attention/key/bias/v:0 (8, 64)
Adam/transformer/layer_3/self_attention/value/kernel/v:0 (512, 8, 64)
Adam/transformer/layer_3/self_attention/value/bias/v:0 (8, 64)
Adam/transformer/layer_3/self_attention/attention_output/kernel/v:0 (8, 64, 512)
Adam/transformer/layer_3/self_attention/attention_output/bias/v:0 (512,)
Adam/transformer/layer_3/self_attention_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_3/self_attention_layer_norm/beta/v:0 (512,)
Adam/transformer/layer_3/intermediate/kernel/v:0 (512, 2048)
Adam/transformer/layer_3/intermediate/bias/v:0 (2048,)
Adam/transformer/layer_3/output/kernel/v:0 (2048, 512)
Adam/transformer/layer_3/output/bias/v:0 (512,)
Adam/transformer/layer_3/output_layer_norm/gamma/v:0 (512,)
Adam/transformer/layer_3/output_layer_norm/beta/v:0 (512,)
Adam/pooler_transform/kernel/v:0 (512, 512)
Adam/pooler_transform/bias/v:0 (512,)
Adam/prediction/dense/kernel/v:0 (512, 5)
Adam/prediction/dense/bias/v:0 (5,)
```

In [ ]:
```
for var in model.trainable_variables:
    print(var.name, var.shape, var.numpy().mean())
```

```
word_embeddings/embeddings:0 (30522, 512) -0.0066453097
position_embedding/embeddings:0 (512, 512) 3.2817425e-06
type_embeddings/embeddings:0 (2, 512) -0.00046012126
embeddings/layer_norm/gamma:0 (512,) 0.99336314
embeddings/layer_norm/beta:0 (512,) -0.022806492
transformer/layer_0/self_attention/query/kernel:0 (512, 8, 64) -4.5812565e-05
transformer/layer_0/self_attention/query/bias:0 (8, 64) 0.008242253
transformer/layer_0/self_attention/key/kernel:0 (512, 8, 64) 1.2599877e-05
transformer/layer_0/self_attention/key/bias:0 (8, 64) -0.00065185595
transformer/layer_0/self_attention/value/kernel:0 (512, 8, 64) 7.231177e-06
transformer/layer_0/self_attention/value/bias:0 (8, 64) 0.003585639
transformer/layer_0/self_attention/attention_output/kernel:0 (8, 64, 512) -5.1669695e
-06
transformer/layer_0/self_attention/attention_output/bias:0 (512,) -0.0025170534
transformer/layer_0/self_attention_layer_norm/gamma:0 (512,) 0.8360751
transformer/layer_0/self_attention_layer_norm/beta:0 (512,) 0.008211255
transformer/layer_0/intermediate/kernel:0 (512, 2048) 1.748698e-05
transformer/layer_0/intermediate/bias:0 (2048,) -0.06786728
transformer/layer_0/output/kernel:0 (2048, 512) 2.767684e-05
transformer/layer_0/output/bias:0 (512,) 0.0012120833
transformer/layer_0/output_layer_norm/gamma:0 (512,) 1.0267036
transformer/layer_0/output_layer_norm/beta:0 (512,) 0.014142748
transformer/layer_1/self_attention/query/kernel:0 (512, 8, 64) -7.8840945e-05
transformer/layer_1/self_attention/query/bias:0 (8, 64) -0.0015058867
transformer/layer_1/self_attention/key/kernel:0 (512, 8, 64) -8.0022855e-06
transformer/layer_1/self_attention/key/bias:0 (8, 64) -0.0012124626
transformer/layer_1/self_attention/value/kernel:0 (512, 8, 64) -9.154148e-06
transformer/layer_1/self_attention/value/bias:0 (8, 64) -0.003475386
transformer/layer_1/self_attention/attention_output/kernel:0 (8, 64, 512) -7.3189362e
-06
transformer/layer_1/self_attention/attention_output/bias:0 (512,) -0.00062419544
transformer/layer_1/self_attention_layer_norm/gamma:0 (512,) 0.8742993
transformer/layer_1/self_attention_layer_norm/beta:0 (512,) 0.015244308
transformer/layer_1/intermediate/kernel:0 (512, 2048) -0.00015509104
transformer/layer_1/intermediate/bias:0 (2048,) -0.05378834
transformer/layer_1/output/kernel:0 (2048, 512) 2.715422e-05
transformer/layer_1/output/bias:0 (512,) 0.00019617332
transformer/layer_1/output_layer_norm/gamma:0 (512,) 1.0965862
transformer/layer_1/output_layer_norm/beta:0 (512,) 0.015992833
transformer/layer_2/self_attention/query/kernel:0 (512, 8, 64) -3.0990122e-08
transformer/layer_2/self_attention/query/bias:0 (8, 64) 0.0013554958
transformer/layer_2/self_attention/key/kernel:0 (512, 8, 64) 2.672428e-06
transformer/layer_2/self_attention/key/bias:0 (8, 64) -0.0008049298
transformer/layer_2/self_attention/value/kernel:0 (512, 8, 64) -2.2756183e-05
transformer/layer_2/self_attention/value/bias:0 (8, 64) 0.0015726285
transformer/layer_2/self_attention/attention_output/kernel:0 (8, 64, 512) -3.0396759e
-05
transformer/layer_2/self_attention/attention_output/bias:0 (512,) -0.0023096022
transformer/layer_2/self_attention_layer_norm/gamma:0 (512,) 0.82641155
transformer/layer_2/self_attention_layer_norm/beta:0 (512,) 0.012189545
transformer/layer_2/intermediate/kernel:0 (512, 2048) -0.00016035879
transformer/layer_2/intermediate/bias:0 (2048,) -0.06222543
transformer/layer_2/output/kernel:0 (2048, 512) 0.000105424915
transformer/layer_2/output/bias:0 (512,) 0.0012809397
transformer/layer_2/output_layer_norm/gamma:0 (512,) 1.11973
transformer/layer_2/output_layer_norm/beta:0 (512,) 0.0327286
transformer/layer_3/self_attention/query/kernel:0 (512, 8, 64) -6.074895e-05
transformer/layer_3/self_attention/query/bias:0 (8, 64) -0.002055696
transformer/layer_3/self_attention/key/kernel:0 (512, 8, 64) -8.961315e-06
transformer/layer_3/self_attention/key/bias:0 (8, 64) -0.00230591
transformer/layer_3/self_attention/value/kernel:0 (512, 8, 64) 5.9269318e-05
```

```
transformer/layer_3/self_attention/value/bias:0 (8, 64) -0.0015618418
transformer/layer_3/self_attention/attention_output/kernel:0 (8, 64, 512) -1.3399658e
-06
transformer/layer_3/self_attention/attention_output/bias:0 (512,) -0.00046601822
transformer/layer_3/self_attention_layer_norm/gamma:0 (512,) 0.7120942
transformer/layer_3/self_attention_layer_norm/beta:0 (512,) 0.032199815
transformer/layer_3/intermediate/kernel:0 (512, 2048) -0.00042257633
transformer/layer_3/intermediate/bias:0 (2048,) -0.045447063
transformer/layer_3/output/kernel:0 (2048, 512) -4.4578896e-06
transformer/layer_3/output/bias:0 (512,) -0.0008670932
transformer/layer_3/output_layer_norm/gamma:0 (512,) 0.78690046
transformer/layer_3/output_layer_norm/beta:0 (512,) -0.02228038
pooler_transform/kernel:0 (512, 512) 4.4187773e-06
pooler_transform/bias:0 (512,) 0.005290821
prediction/dense_1/kernel:0 (512, 5) 0.0025037632
prediction/dense_1/bias:0 (5,) -4.818695e-05
```

In [ ]: