bert-disaster-tweet_lower_acc

May 23, 2025

[23]:
```
#Importing important libraires
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow_hub as hub
!pip install bert-for-tf2
from bert import bert_tokenization
#!wget --quiet https://raw.githubusercontent.com/tensorflow/models/master/
 ↪official/nlp/bert/tokenization.py
#import tokenization
import matplotlib.pyplot as plt

#Gpu usage
tf.config.list_physical_devices('GPU')
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

```
Requirement already satisfied: bert-for-tf2 in /opt/conda/lib/python3.10/site-
packages (0.14.9)
Requirement already satisfied: py-params>=0.9.6 in
/opt/conda/lib/python3.10/site-packages (from bert-for-tf2) (0.10.2)
Requirement already satisfied: params-flow>=0.8.0 in
/opt/conda/lib/python3.10/site-packages (from bert-for-tf2) (0.8.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages
(from params-flow>=0.8.0->bert-for-tf2) (1.23.5)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages
(from params-flow>=0.8.0->bert-for-tf2) (4.66.1)
```

[23]:
```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
```

```
 incarnation: 5391614925084781990
 xla_global_id: -1,
 name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 16266690560
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 9198000694050359603
 physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id:
0000:00:04.0, compute capability: 6.0"
 xla_global_id: 416903419]
```

[24]:
```
#setting a seed: The answer to the life
SEED = 42
def seed_everything(seed):
    np.random.seed(seed)
    tf.random.set_seed(seed)

seed_everything(SEED)
```

Reading in the data using pandas. We will tokenize the text later in the notebook.

[25]:
```
#reading input data with pandas, this is for kaggle, you can change it for your␣
 ↪local folders
train = pd.read_csv("/kaggle/input/nlp-getting-started/train.csv")
test = pd.read_csv("/kaggle/input/nlp-getting-started/test.csv")
submission = pd.read_csv("/kaggle/input/nlp-getting-started/sample_submission.
 ↪csv")

#visualizing some of the tweets
for i, val in enumerate(train.iloc[:2]["text"].to_list()):
    print("Tweet {}: {}".format(i+1, val))
```

```
Tweet 1: Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all
Tweet 2: Forest fire near La Ronge Sask. Canada
```

[26]:
```
#To see how many labels are positive or negative
print(sum(train.target==1))
print(sum(train.target==0))
```

```
3271
4342
```

[27]:
```
#To see what we do have in the pandas file
train.head(5)
```

[27]:
```
   id keyword location                                              text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M…
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are …
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or…
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as …

   target
0       1
1       1
2       1
3       1
4       1
```

[28]:
```
#You can use the following code to drop if you have duplicates

#train = train.drop_duplicates(subset=['text'], keep='first')
#train.info()
```

[29]:
```
# The following functions are defined for preprocessing.


def remove_URL(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'', text)


def remove_emoji(text):
    emoji_pattern = re.compile(
        '['
        u'\U0001F600-\U0001F64F'  # emoticons
        u'\U0001F300-\U0001F5FF'  # symbols & pictographs
        u'\U0001F680-\U0001F6FF'  # transport & map symbols
        u'\U0001F1E0-\U0001F1FF'  # flags (iOS)
        u'\U00002702-\U000027B0'
        u'\U000024C2-\U0001F251'
        ']+',
        flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)


def remove_html(text):
    html = re.compile(r'<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')
    return re.sub(html, '', text)


def remove_punct(text):
```

```
    table = str.maketrans('', '', string.punctuation)
    return text.translate(table)

def remove_non_ascii(text):
    return ''.join(i for i in text if ord(i) < 128)
```

[30]:
```
import re
import string
```

[31]:
```
#We will create new series from train["text"]

train['text_clean'] = train['text'].apply(lambda x: x.lower())
train['text_clean'] = train['text_clean'].apply(lambda x: remove_URL(x))
train['text_clean'] = train['text_clean'].apply(lambda x: remove_punct(x))
train['text_clean'] = train['text_clean'].apply(lambda x: remove_non_ascii(x))
train['text_clean'] = train['text_clean'].apply(lambda x: remove_emoji(x))
train['text_clean'] = train['text_clean'].apply(lambda x: remove_html(x))


train.head()
```

[31]:

	id	keyword	location	text	\
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M…	
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	
2	5	NaN	NaN	All residents asked to 'shelter in place' are …	
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or…	
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as …	

	target	text_clean
0	1	our deeds are the reason of this earthquake ma…
1	1	forest fire near la ronge sask canada
2	1	all residents asked to shelter in place are be…
3	1	13000 people receive wildfires evacuation orde…
4	1	just got sent this photo from ruby alaska as s…

[32]:
```
#Same for test data

test['text_clean'] = test['text'].apply(lambda x: x.lower())
test['text_clean'] = test['text_clean'].apply(lambda x: remove_URL(x))
test['text_clean'] = test['text_clean'].apply(lambda x: remove_punct(x))
test['text_clean'] = test['text_clean'].apply(lambda x: remove_non_ascii(x))
test['text_clean'] = test['text_clean'].apply(lambda x: remove_emoji(x))
test['text_clean'] = test['text_clean'].apply(lambda x: remove_html(x))


test.head()
```

[32]:
```
   id keyword location                                              text  \
0   0     NaN      NaN                 Just happened a terrible car crash
1   2     NaN      NaN  Heard about #earthquake is different cities, s…
2   3     NaN      NaN  there is a forest fire at spot pond, geese are…
3   9     NaN      NaN           Apocalypse lighting. #Spokane #wildfires
4  11     NaN      NaN      Typhoon Soudelor kills 28 in China and Taiwan

                                       text_clean
0                 just happened a terrible car crash
1  heard about earthquake is different cities sta…
2  there is a forest fire at spot pond geese are …
3              apocalypse lighting spokane wildfires
4      typhoon soudelor kills 28 in china and taiwan
```

[33]:
```
test['text_clean']
```

[33]:
```
0                      just happened a terrible car crash
1       heard about earthquake is different cities sta…
2       there is a forest fire at spot pond geese are …
3                   apocalypse lighting spokane wildfires
4           typhoon soudelor kills 28 in china and taiwan
                              …
3258    earthquake safety los angeles  safety fastener…
3259    storm in ri worse than last hurricane my citya…
3260                     green line derailment in chicago
3261            meg issues hazardous weather outlook hwo
3262    cityofcalgary has activated its municipal emer…
Name: text_clean, Length: 3263, dtype: object
```

[34]:
```
#What is the max length of any text? This will be important for tokenization

print(min(train["text"].apply(len)))
max(train["text"].apply(len))
```

```
7
```

[34]:
```
157
```

##

bert_encode function

0.0.1 tokenizer

We are using the Tensorflow Research's BERT tokenization method. This tokenization method can be thought of as three steps.

- Text Normalization
  - The first part of the tokenizer converts the text to lowercase (given that we are using the uncased version of roBERTa), converts whitespace to spaces, and strips out accent

markers.

`"Alex Pättason's, "  -> "alex pattason's,"`

- Punctuation splitting
  - This next step adds spaces on each side of all "punctuation". Note that this includes any non-letter/number/space ASCII characters (ie including $, @). See more of this in the Docs.

`"Alex Pättason's, "  -> "alex pattason ' s ,"`

- WordPiece tokenization
  - This step applies what is called whitespace tokenization to the output of the process above, and apply's WordPiece tokenization to each word separately. See the example below.

`"Alex Pättason's, "  -> "alex pat ##ta ##son ' s ,"`

0.0.2 tags

The next part of the function reduces the length of the text by the max_length that we have specified and adds [CLS] and [SEP] tags to the end of the array. The [CLS] tag is short for classification and indicates the start of the sentence. Similarly, the [SEP] tag indicates the end of the sentence.

0.0.3 convert_tokens_to_ids + pad_masks

We then use the tokenizer method to replace the string representation of words with integers. We also create the input mask (AKA pad_masks), and the segment id's. Note that we are not fulfilling the segment_ids full benefits below as we are only passing an array of zeros. More on the tokens, pad_masks, and segment_ids further in the notebook.

[35]:
```
#This is encoder for Bert. You can choose max-length of text for this.

def bert_encode(texts, tokenizer, max_len=512):
    all_tokens = []
    all_masks = []
    all_segments = []

    for text in texts:
        text = tokenizer.tokenize(text)

        text = text[:max_len-2]
        input_sequence = ["[CLS]"] + text + ["[SEP]"]
        pad_len = max_len - len(input_sequence)

        tokens = tokenizer.convert_tokens_to_ids(input_sequence)
        tokens += [0] * pad_len
        pad_masks = [1] * len(input_sequence) + [0] * pad_len
        segment_ids = [0] * max_len

        all_tokens.append(tokens)
        all_masks.append(pad_masks)
```

```
        all_segments.append(segment_ids)

    return np.array(all_tokens), np.array(all_masks), np.array(all_segments)
```

##

build_model function

The first three components of the function are basically preprocessing plain text inputs into the input format expected by the roBERTa model.

0.0.4 input_word_ids

- Basically maps each word to its token id. There can be multiple different values that correspond with the same word. For example, "smell" could be encoded both as 883 and 789.

```
text = "I love this notebook. It is Great."
input_word_ids = [10, 235, 123, 938, 184, 301, 567]
```

0.0.5 the input_mask

- Shows where the sentence begins, and where it ends using an array. All input tokens that are not padding are given a value of 1, and all values that are padding are given 0. If the sentence exceeds that max_length, then the entire vector will be of 1's.

```
text = "I love this notebook. It is Great."
input_mask = [1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
```

0.0.6 segment_ids

- This component is still a little vague for me, but from my understanding, it is recognizing segments of the text. The start of each segment has a 1 in the array, and other components and padding all have a zero. I am unsure as to whether this corresponds to the end of sentences or paragraphs, but if you can explain this better please do so in the comments below!

```
text = "I love this notebook. It is Great."
segment_ids = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

[36]:
```
def build_model(bert_layer, max_len=512):
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32,␣
 ↪name="input_word_ids")
    input_mask = Input(shape=(max_len,), dtype=tf.int32, name="input_mask")
    segment_ids = Input(shape=(max_len,), dtype=tf.int32, name="segment_ids")

    #could be pooled_output, sequence_output yet sequence output provides for␣
 ↪each input token (in context)
    _, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])
    clf_output = sequence_output[:, 0, :]
    #new_output = Dense(256, activation='relu')(clf_output)
    #c_output = Dense(256, activation='relu')(new_output)
```

```
    out = Dense(1, activation='sigmoid')(clf_output)

    model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=out)

    #specifying optimizer
    model.compile(Adam(learning_rate=6*10**(-6)), loss='binary_crossentropy',␣
↪metrics=['accuracy'])

    return model
```

0.1 Build Model

The first cell below is basically loading in the version of the BERTa model that we want to use. We are using a Large uncased model. The most simple way to use a BERTa model and modify it to a specific use case is to set it as a KerasLayer.

Note there are many different variations of BERT models that you can look through here –> TFhub Bert

[46]:
```
#load uncased bert model
module_url = "https://tfhub.dev/tensorflow/bert_en_uncased_L-24_H-1024_A-16/2"
bert_layer = hub.KerasLayer(module_url, trainable=True)
```

In the next cell, we are setting up the tokenizer that will be used to preprocess our input data to what BERT understands. We have to specify a vocab file so that the tokenizer knows what number to encode each word as then we have to specify whether we want uncased or cased text. We will use the same vocab_file that the pre-trained model was trained on (Google's SentencePiece in this case) and we will also use the same case that the model was built for (uncased).

Finally, once we have these two variables, we create the tokenizer and tokenize the training and testing data using the bert_encode function that we created above.

[47]:
```
max_len = 157
#vocab file from pre-trained BERT for tokenization
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()

#returns true/false depending on if we selected cased/uncased bert layer
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()

#Create the tokenizer
tokenizer = bert_tokenization.FullTokenizer(vocab_file, do_lower_case)

#tokenizing the training and testing data
train_input = bert_encode(train.text.values, tokenizer, max_len=max_len)
test_input = bert_encode(test.text.values, tokenizer, max_len=max_len)
train_labels = train.target.values
```

Having a look at the model summary. We can see the three input layers that we created followed by the roBERTa model which is in the keras_layer. We have the final dense layer which predicts the sentiment of the tweet on a scale of 0-1.

[48]:
```
model = build_model(bert_layer, max_len=max_len)

model.summary()
```

```
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_word_ids (InputLayer)    [(None, 157)]        0           []

 input_mask (InputLayer)        [(None, 157)]        0           []

 segment_ids (InputLayer)       [(None, 157)]        0           []

 keras_layer_6 (KerasLayer)     [(None, 1024),       335141889
['input_word_ids[0][0]',
                                 (None, 157, 1024)]
'input_mask[0][0]',
'segment_ids[0][0]']

 tf.__operators__.getitem (Slic  (None, 1024)        0
['keras_layer_6[0][1]']
 ingOpLambda)

 dense (Dense)                  (None, 1)            1025
['tf.__operators__.getitem[0][0]'
                                                                  ]

==================================================================================
==================
Total params: 335,142,914
Trainable params: 335,142,913
Non-trainable params: 1
__________________________________________________________________________________
__________________
```

0.2 Training Model

The next cell is a simple way of training the model using Keras. We have included the built-in ModelCheckpoint callback to only save the model that has the highest validation accuracy. This ensures we are only saving the best models.

We could decrease the randomness of the split by doing some sort of a stratified split, or cross-validation, but this will do for now.

[49]:
```
#checkpoint = ModelCheckpoint('model.h5', monitor='val_accuracy',␣
 ↪save_best_only=True, patience=3)
checkpoint = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',␣
 ↪patience=2)
#FixedEarlyStopping()

#You can change hyperparameters with Grid-Search or Bayesian Search

train_history = model.fit(
    train_input, train_labels,
    validation_split=0.2,
    epochs=100,
    callbacks=[checkpoint],
    batch_size=16,
    shuffle=True
)

# # max_len with 1: 200
# # Epoch 1/3
# 381/381 [==============================] - 507s 1s/step - loss: 0.4273 -␣
 ↪accuracy: 0.8158 - val_loss: 0.3778 - val_accuracy: 0.8431
# Epoch 2/3
# 381/381 [==============================] - 472s 1s/step - loss: 0.2881 -␣
 ↪accuracy: 0.8834 - val_loss: 0.3937 - val_accuracy: 0.8332
```

```
Epoch 1/100
381/381 [==============================] - 474s 1s/step - loss: 0.4370 -
accuracy: 0.8102 - val_loss: 0.3820 - val_accuracy: 0.8418
Epoch 2/100
381/381 [==============================] - 395s 1s/step - loss: 0.3376 -
accuracy: 0.8662 - val_loss: 0.3915 - val_accuracy: 0.8418
Epoch 3/100
381/381 [==============================] - 395s 1s/step - loss: 0.2770 -
accuracy: 0.8947 - val_loss: 0.4261 - val_accuracy: 0.8378
```

0.3 Make Prediction

Using the model to make predictions on the testing set. We round the prediction to 1 or 0. 1 is a disaster tweet, and 0 is a regular tweet.

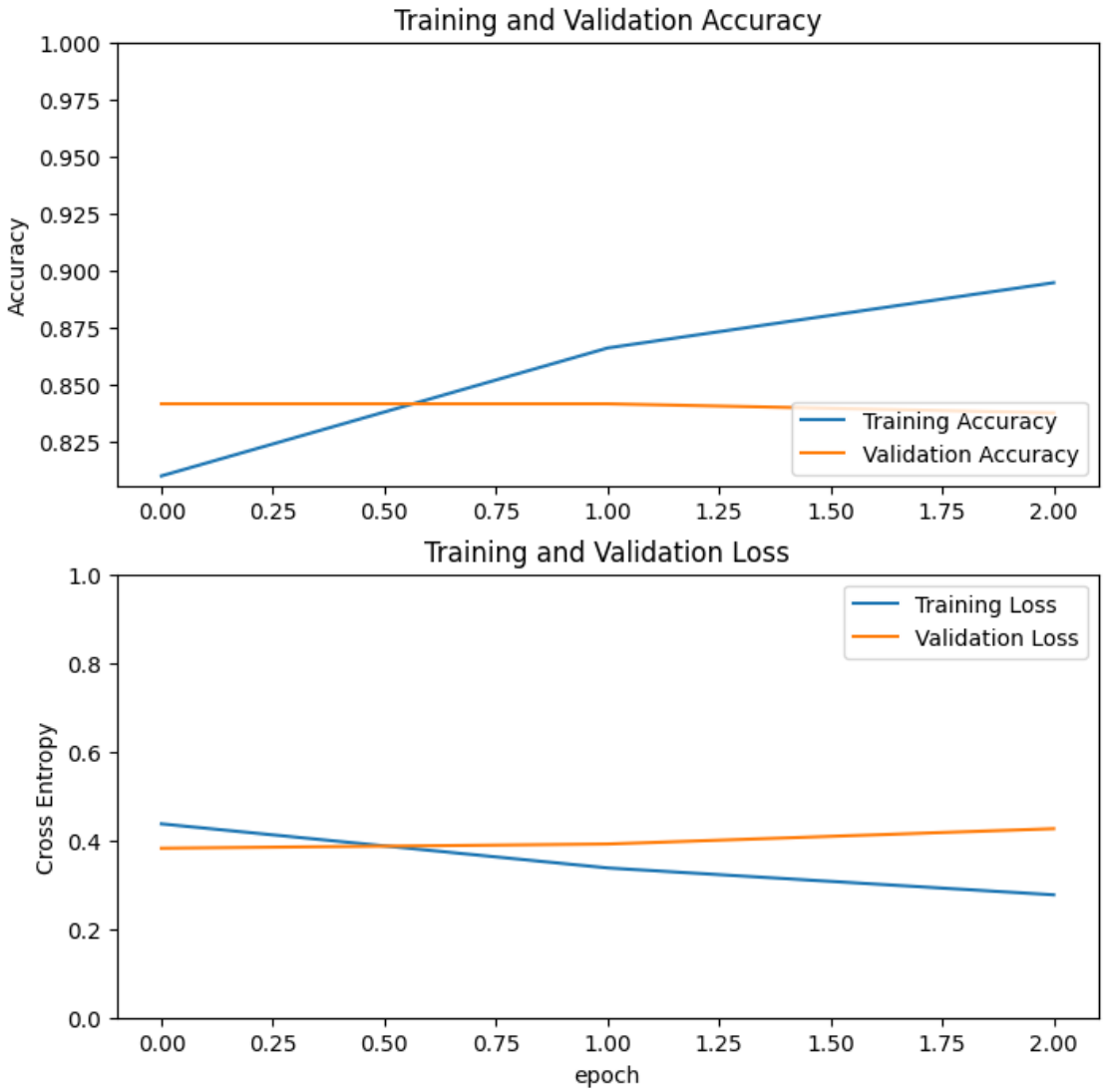
[50]:
```
acc = train_history.history['accuracy']
val_acc = train_history.history['val_accuracy']

loss = train_history.history['loss']
val_loss = train_history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
```

```
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



[51]:
```
# If you don't overfit or underfit, you can submit
test_pred = model.predict(test_input)

submission['target'] = test_pred.round().astype(int)
submission.to_csv('submission.csv', index=False)
```

```
102/102 [==============================] - 61s 596ms/step
```

[ ]:
```

```