Multi-Label, Multi-Class Text Classification with BERT, Transformers and Keras

In this article, I'll show how to do a multi-label, multi-class text classification task using Huggingface Transformers library and Tensorflow Keras API. In doing so, you'll learn how to use a BERT model from Transformer as a layer in a Tensorflow model built using the Keras API.

Emil Lykke Jensen Aug 25, 2020 · 8 min read ★

```
# Load the MainLayer
bert = transformer_model.layers[0]

# Build your model input
input_ids = Input(shape=(max_length,), name='input_ids', dtype='int32')
inputs = {'input_ids': input_ids}      You, a few seconds ago • Uncommitted changes

# Load the Transformers BERT model as a layer in a Keras model
bert_model = bert(inputs)[1]
dropout = Dropout(config.hidden_dropout_prob, name='pooled_output')
pooled_output = dropout(bert_model, training=False)

# Then build your model output
issue = Dense(units=len(data.Issue_label.value_counts()), kernel_initializer=TruncatedNormal(stddev=config.initializer_range), name='issue')(pooled_output)
product = Dense(units=len(data.Product_label.value_counts()), kernel_initializer=TruncatedNormal(stddev=config.initializer_range), name='product')(pooled_output)
outputs = {'issue': issue, 'product': product}

# And combine it all in a model object
model = Model(inputs=inputs, outputs=outputs, name='BERT_MultiLabel_MultiClass')

# Take a look at the model
model.summary()
```

Multi-Label, Multi-Class Text Classification with BERT, Transformers and Keras

The internet is full of text classification articles, most of which are BoW-models combined with some kind of ML-model typically solving a binary text classification problem. With the rise of NLP, and in particular BERT (take a look here, if you are not familiar with BERT) and other multilingual transformer based models, more and more text classification problems can now be solved.

However, when it comes to solving a multi-label, multi-class text classification problem using Huggingface Transformers, BERT, and Tensorflow Keras, the number of articles are indeed very limited and I for one, haven't found any… Yet!

Therefore, with the help and inspiration of a great deal of blog posts, tutorials and GitHub code snippets all relating to either BERT, multi-label classification in Keras or other useful information I will show you how to build a working model, solving exactly that problem.

And why use Huggingface Transformers instead of Googles own BERT solution? Because with Transformers it is extremely easy to switch between different models, that being BERT, ALBERT, XLnet, GPT-2 etc. Which means, that you more or less 'just' replace one model for another in your code.

Where to start

With data. Looking for text data I could use for a multi-label multi-class text classification task, I stumbled upon the 'Consumer Complaint Database' from data.gov. Seems to do the trick, so that's what we'll use.

Next up is the exploratory data analysis. This is obviously crucial to get a proper understanding of what your data looks like, what pitfalls there might be, the quality of your data, and so on. But I'm skipping this step for now, simply because the aim of this article is purely how to build a model.

If you don't like googling around take a look at these two articles on the subject: NLP Part 3 | Exploratory Data Analysis of Text Data and A Complete Exploratory Data Analysis and Visualization for Text Data.

Get on with it

We have our data and now comes the coding part.

First, we'll load the required libraries.

```
#######################################
### -------- Load libraries ------- ###

# Load Huggingface transformers
from transformers import TFBertModel,  BertConfig, BertTokenizerFast

# Then what you need from tensorflow.keras
from tensorflow.keras.layers import Input, Dropout, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.utils import to_categorical

# And pandas for data import + sklearn because you allways need
sklearn
import pandas as pd
from sklearn.model_selection import train_test_split
```

Then we will import our data and wrangle it around so it fits our needs. Nothing fancy there. Note that we will only use the columns 'Consumer complaint narrative', 'Product' and 'Issue' from our dataset. 'Consumer complaint narrative' will serve as our input for the model and 'Product' and 'Issue' as our two outputs.

```
#######################################
### --------- Import data --------- ###

# Import data from csv
data = pd.read_csv('dev/Fun with BERT/complaints.csv')

# Select required columns
data = data[['Consumer complaint narrative', 'Product', 'Issue']]

# Remove a row if any of the three remaining columns are missing
data = data.dropna()

# Remove rows, where the label is present only ones (can't be split)
data = data.groupby('Issue').filter(lambda x : len(x) > 1)
data = data.groupby('Product').filter(lambda x : len(x) > 1)

# Set your model output as categorical and save in new label col
data['Issue_label'] = pd.Categorical(data['Issue'])
data['Product_label'] = pd.Categorical(data['Product'])
```

```
# Transform your output to numeric
data['Issue'] = data['Issue_label'].cat.codes
data['Product'] = data['Product_label'].cat.codes

# Split into train and test - stratify over Issue
data, data_test = train_test_split(data, test_size = 0.2, stratify =
data[['Issue']])
```

Next we will load a number of different Transformers classes.

```
#######################################
### --------- Setup BERT ---------- ###

# Name of the BERT model to use
model_name = 'bert-base-uncased'

# Max length of tokens
max_length = 100

# Load transformers config and set output_hidden_states to False
config = BertConfig.from_pretrained(model_name)
config.output_hidden_states = False

# Load BERT tokenizer
tokenizer =
BertTokenizerFast.from_pretrained(pretrained_model_name_or_path =
model_name, config = config)

# Load the Transformers BERT model
transformer_model = TFBertModel.from_pretrained(model_name, config =
config)
```

Here we first load a BERT config object that controls the model, tokenizer and so on.

Then, a tokenizer that we will use later in our script to transform our text input into BERT tokens and then pad and truncate them to our max length. The tokenizer is pretty well documented so I won't get into that here.

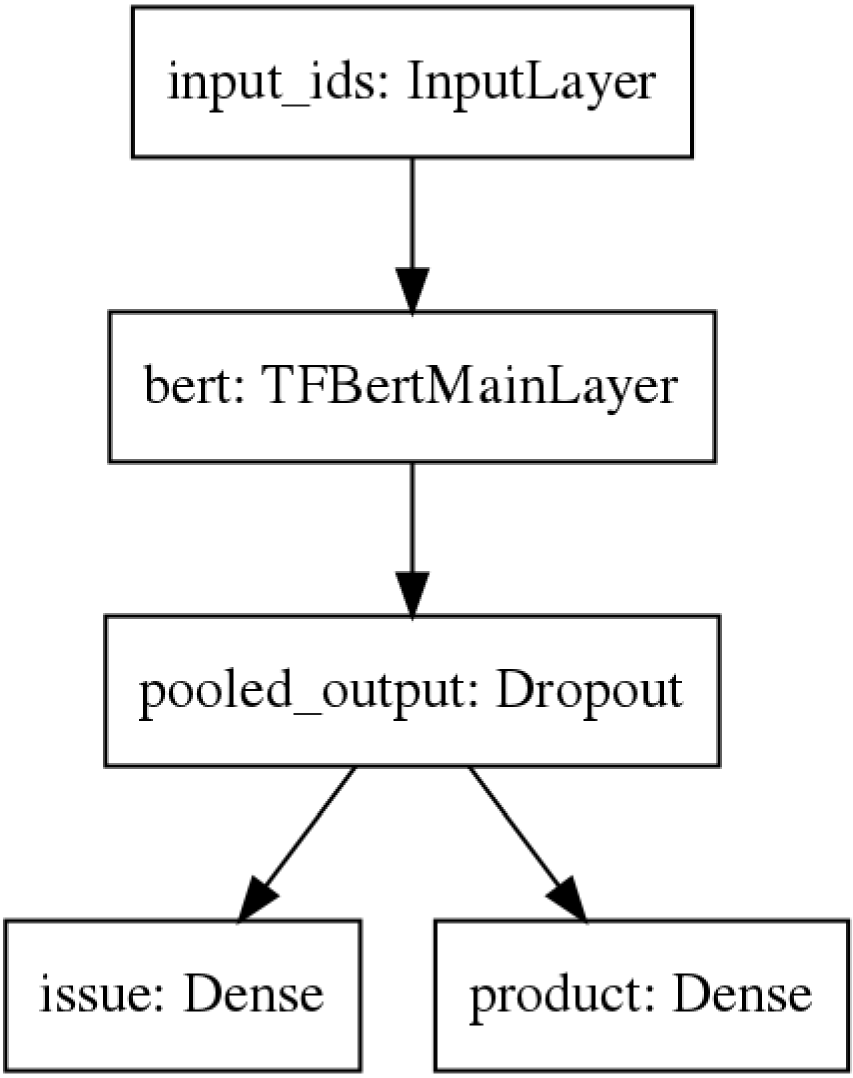
Lastly, we will load the BERT model itself as a BERT Transformers TF 2.0 Keras model (here we use the 12-layer bert-base-uncased).

Now for the fun part

We are ready to build our model. In the Transformers library, there are a number of different BERT classification models to use. The mother of all models is the one simply called 'BertModel' (PyTorch) or 'TFBertModel' (TensorFlow) and thus the one we want.

The Transformers library also comes with a prebuilt BERT model for sequence classification called 'TFBertForSequenceClassification'. If you take a look at the code found here you'll see, that they start by loading a clean BERT model and then they simply add a dropout and a dense layer to it. Therefore, what we'll do is simply to add two dense layers instead of just one.

Here what our model looks like:



The Multi-Label, Multi-Class Text Classification with BERT, Transformer and Keras model

And a more detailed view of the model:

```
Model: "BERT_MultiLabel_MultiClass"
```

```
Layer (type)        Output Shape          Param #    Connected to
==================================================================
input_ids           [(None, 100)]         0
(InputLayer)
__________________________________________________________________
bert                (                     109482240  input_ids[0][0]
(TFBertMainLayer)    (None, 100, 768),
                     (None, 768)
                    )
__________________________________________________________________
pooled_output       (None, 768)           0          bert[1][1]
(Dropout)
__________________________________________________________________
issue               (None, 159)           122271     pooled_output[0][0]
(Dense)
__________________________________________________________________
product             (None, 18)            13842      pooled_output[0][0]
(Dense)
==================================================================
Total params: 109,618,353
Trainable params: 109,618,353
Non-trainable params: 0
__________________________________________________________________
```

If you want to know more about BERTs architecture itself, take a look here.

Now that we have our model architecture, all we need to do is write it in code.

```
#######################################
### ------- Build the model ------- ###

# TF Keras documentation:
https://www.tensorflow.org/api_docs/python/tf/keras/Model

# Load the MainLayer
bert = transformer_model.layers[0]

# Build your model input
input_ids = Input(shape=(max_length,), name='input_ids',
dtype='int32')
inputs = {'input_ids': input_ids}

# Load the Transformers BERT model as a layer in a Keras model
bert_model = bert(inputs)[1]
dropout = Dropout(config.hidden_dropout_prob, name='pooled_output')
pooled_output = dropout(bert_model, training=False)

# Then build your model output
issue = Dense(units=len(data.Issue_label.value_counts()),
kernel_initializer=TruncatedNormal(stddev=config.initializer_range),
name='issue')(pooled_output)
product = Dense(units=len(data.Product_label.value_counts()),
```

```
kernel_initializer=TruncatedNormal(stddev=config.initializer_range),
name='product')(pooled_output)
outputs = {'issue': issue, 'product': product}

# And combine it all in a model object
model = Model(inputs=inputs, outputs=outputs,
name='BERT_MultiLabel_MultiClass')

# Take a look at the model
model.summary()
```

Let the magic begin

Then all there is left to do is to compile our new model and fit it on our data.

```
#######################################
### ------- Train the model ------- ###

# Set an optimizer
optimizer = Adam(
    learning_rate=5e-05,
    epsilon=1e-08,
    decay=0.01,
    clipnorm=1.0)

# Set loss and metrics
loss = {'issue': CategoricalCrossentropy(from_logits = True),
'product': CategoricalCrossentropy(from_logits = True)}
metric = {'issue': CategoricalAccuracy('accuracy'), 'product':
CategoricalAccuracy('accuracy')}

# Compile the model
model.compile(
    optimizer = optimizer,
    loss = loss,
    metrics = metric)

# Ready output data for the model
y_issue = to_categorical(data['Issue'])
y_product = to_categorical(data['Product'])

# Tokenize the input (takes some time)
x = tokenizer(
    text=data['Consumer complaint narrative'].to_list(),
    add_special_tokens=True,
    max_length=max_length,
    truncation=True,
    padding=True,
```

```
    return_tensors='tf',
    return_token_type_ids = False,
    return_attention_mask = False,
    verbose = True)

# Fit the model
history = model.fit(
    x={'input_ids': x['input_ids']},
    y={'issue': y_issue, 'product': y_product},
    validation_split=0.2,
    batch_size=64,
    epochs=10)
```

Once the model is fitted, we can evaluate it on our test data to see how it performs.

```
#######################################
### ----- Evaluate the model ------ ###

# Ready test data
test_y_issue = to_categorical(data_test['Issue'])
test_y_product = to_categorical(data_test['Product'])
test_x = tokenizer(
    text=data_test['Consumer complaint narrative'].to_list(),
    add_special_tokens=True,
    max_length=max_length,
    truncation=True,
    padding=True,
    return_tensors='tf',
    return_token_type_ids = False,
    return_attention_mask = False,
    verbose = True)

# Run evaluation
model_eval = model.evaluate(
    x={'input_ids': test_x['input_ids']},
    y={'issue': test_y_issue, 'product': test_y_product}
)
```

As it turns out, our model performs fairly okay and has a relatively good accuracy. Especially considering the fact that our output 'Product' consists of 18 labels and 'Issue' consists of 159 different labels.

```
#####################################################################
Classification metrics for Product

                           precision    recall  f1-score   support
    Bank account or service       0.63      0.36      0.46      2977
Checking or savings account       0.60      0.75      0.67      4685
```

```
              Consumer Loan       0.48      0.29      0.36      1876
                Credit card       0.56      0.42      0.48      3765
Credit card or prepaid card       0.63      0.71      0.67      8123
           Credit reporting       0.64      0.37      0.47      6318
   Credit reporting, credit
  repair services, or other
  personal consumer reports       0.81      0.85      0.83     38529
            Debt collection       0.80      0.85      0.82     23848
    Money transfer, virtual
 currency, or money service       0.59      0.65      0.62      1966
            Money transfers       0.50      0.01      0.01       305
                   Mortgage       0.89      0.93      0.91     13502
    Other financial service       0.00      0.00      0.00        60
                Payday loan       0.57      0.01      0.02       355
Payday loan, title loan, or
              personal loan       0.46      0.40      0.43      1523
               Prepaid card       0.82      0.14      0.24       294
               Student loan       0.83      0.87      0.85      5332
      Vehicle loan or lease       0.49      0.51      0.50      1963
           Virtual currency       0.00      0.00      0.00         3

                   accuracy                           0.76    115424
                  macro avg       0.57      0.45      0.46    115424
               weighted avg       0.75      0.76      0.75    115424

####################################################################
Classification metrics for Issue (only showing summarized metrics)

precision    recall  f1-score   support
                   accuracy                           0.41    115424
                  macro avg       0.09      0.08      0.06    115424
```

What to do next?

There are, however, plenty of things you could do to increase performance of this model. Here I have tried to do it as simple as possible, but if you are looking for better performance consider the following:

- Fiddle around with the hyperparameters set in the optimizer or change the optimizer itself
- Train a language model using the Consumer Complaint Database data- either from scratch or by fine-tuning an existing BERT model (have a look here to see how). Then load that model instead of the 'bert-base-uncased' used here.

- Use multiple inputs. In our current setup, we only use token id's as input. However, we could (probably) gain some performance increase if we added attention masks to our input. It is pretty straightforward and looks something like this:

```
# Build your model input

input_ids = Input(shape=(max_length,), name='input_ids',
dtype='int32')

attention_mask = Input(shape=(max_length,), name='attention_mask',
dtype='int32')

inputs = {'input_ids': input_ids, 'attention_mask': attention_mask}
```

(remember to add attention_mask when fitting your model and set return_attention_mask to True in your tokenizer. For more info on attention masks, look here. Also I have added attention_mask to the gist below and commented it out for your inspiration.*)*

- Try another model such as ALBERT, RoBERTa, XLM or even an autoregressive model such as GPT-2 or XLNet — all of them easily imported into your framework though the Transformers library. You can find an overview of all the directly available models here.

That's it — hope you like this little walk-through of how to do a 'Multi-Label, Multi-Class Text Classification with BERT, Transformer and Keras'. If you have any feedback or questions, fire away in the comments below.

```
#######################################
### -------- Load libraries ------- ###

# Load Huggingface transformers
from transformers import TFBertModel,  BertConfig, BertTokenizerFast

# Then what you need from tensorflow.keras
from tensorflow.keras.layers import Input, Dropout, Dense
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.utils import to_categorical

# And pandas for data import + sklearn because you allways need sklearn
import pandas as pd
from sklearn.model_selection import train_test_split


#######################################
### --------- Import data --------- ###

# Import data from csv
data = pd.read_csv('dev/Fun with BERT/complaints.csv')

# Select required columns
data = data[['Consumer complaint narrative', 'Product', 'Issue']]

# Remove a row if any of the three remaining columns are missing
data = data.dropna()

# Remove rows, where the label is present only ones (can't be split)
data = data.groupby('Issue').filter(lambda x : len(x) > 1)
data = data.groupby('Product').filter(lambda x : len(x) > 1)

# Set your model output as categorical and save in new label col
data['Issue_label'] = pd.Categorical(data['Issue'])
data['Product_label'] = pd.Categorical(data['Product'])

# Transform your output to numeric
data['Issue'] = data['Issue_label'].cat.codes
data['Product'] = data['Product_label'].cat.codes

# Split into train and test - stratify over Issue
data, data_test = train_test_split(data, test_size = 0.2, stratify = data[['Issue']])


#######################################
### --------- Setup BERT ---------- ###

# Name of the BERT model to use
model_name = 'bert-base-uncased'

# Max length of tokens
```

```
max_length = 100

# Load transformers config and set output_hidden_states to False
config = BertConfig.from_pretrained(model_name)
config.output_hidden_states = False

# Load BERT tokenizer
tokenizer = BertTokenizerFast.from_pretrained(pretrained_model_name_or_path = model_na

# Load the Transformers BERT model
transformer_model = TFBertModel.from_pretrained(model_name, config = config)


#######################################
### ------- Build the model ------- ###

# TF Keras documentation: https://www.tensorflow.org/api_docs/python/tf/keras/Model

# Load the MainLayer
bert = transformer_model.layers[0]

# Build your model input
input_ids = Input(shape=(max_length,), name='input_ids', dtype='int32')
# attention_mask = Input(shape=(max_length,), name='attention_mask', dtype='int32')
# inputs = {'input_ids': input_ids, 'attention_mask': attention_mask}
inputs = {'input_ids': input_ids}

# Load the Transformers BERT model as a layer in a Keras model
bert_model = bert(inputs)[1]
dropout = Dropout(config.hidden_dropout_prob, name='pooled_output')
pooled_output = dropout(bert_model, training=False)

# Then build your model output
issue = Dense(units=len(data.Issue_label.value_counts()), kernel_initializer=Truncated
product = Dense(units=len(data.Product_label.value_counts()), kernel_initializer=Trunc
outputs = {'issue': issue, 'product': product}

# And combine it all in a model object
model = Model(inputs=inputs, outputs=outputs, name='BERT_MultiLabel_MultiClass')

# Take a look at the model
model.summary()


#######################################
### ------- Train the model ------- ###

```

```
# Set an optimizer
optimizer = Adam(
    learning_rate=5e-05,
    epsilon=1e-08,
    decay=0.01,
    clipnorm=1.0)

# Set loss and metrics
loss = {'issue': CategoricalCrossentropy(from_logits = True), 'product': CategoricalCr
metric = {'issue': CategoricalAccuracy('accuracy'), 'product': CategoricalAccuracy('ac

# Compile the model
model.compile(
    optimizer = optimizer,
    loss = loss,
    metrics = metric)

# Ready output data for the model
y_issue = to_categorical(data['Issue'])
y_product = to_categorical(data['Product'])

# Tokenize the input (takes some time)
x = tokenizer(
    text=data['Consumer complaint narrative'].to_list(),
    add_special_tokens=True,
    max_length=max_length,
    truncation=True,
    padding=True,
    return_tensors='tf',
    return_token_type_ids = False,
    return_attention_mask = True,
    verbose = True)

# Fit the model
history = model.fit(
    # x={'input_ids': x['input_ids'], 'attention_mask': x['attention_mask']},
    x={'input_ids': x['input_ids']},
    y={'issue': y_issue, 'product': y_product},
    validation_split=0.2,
    batch_size=64,
    epochs=10)


#######################################
### ----- Evaluate the model ------ ###

# Ready test data
test_y_issue = to_categorical(data_test['Issue'])
```

```
test_y_product = to_categorical(data_test['Product'])
test_x = tokenizer(
    text=data_test['Consumer complaint narrative'].to_list(),
    add_special_tokens=True,
    max_length=max_length,
    truncation=True,
    padding=True,
    return_tensors='tf',
    return_token_type_ids = False,
    return_attention_mask = False,
    verbose = True)

# Run evaluation
model_eval = model.evaluate(
    x={'input_ids': test_x['input_ids']},
    y={'issue': test_y_issue, 'product': test_y_product}
)
```

Thanks to Michael Armanious.

TensorFlow Keras Bert Transformers NLP