Building a Multi-label Text Classifier using BERT and TensorFlow

Javaid Nabi Follow
May 11, 2019 · 8 min read

In a **multi-label classification** problem, the training set is composed of instances each can be assigned with multiple categories represented as a set of target labels and the task is to predict the label set of test data e.g.,

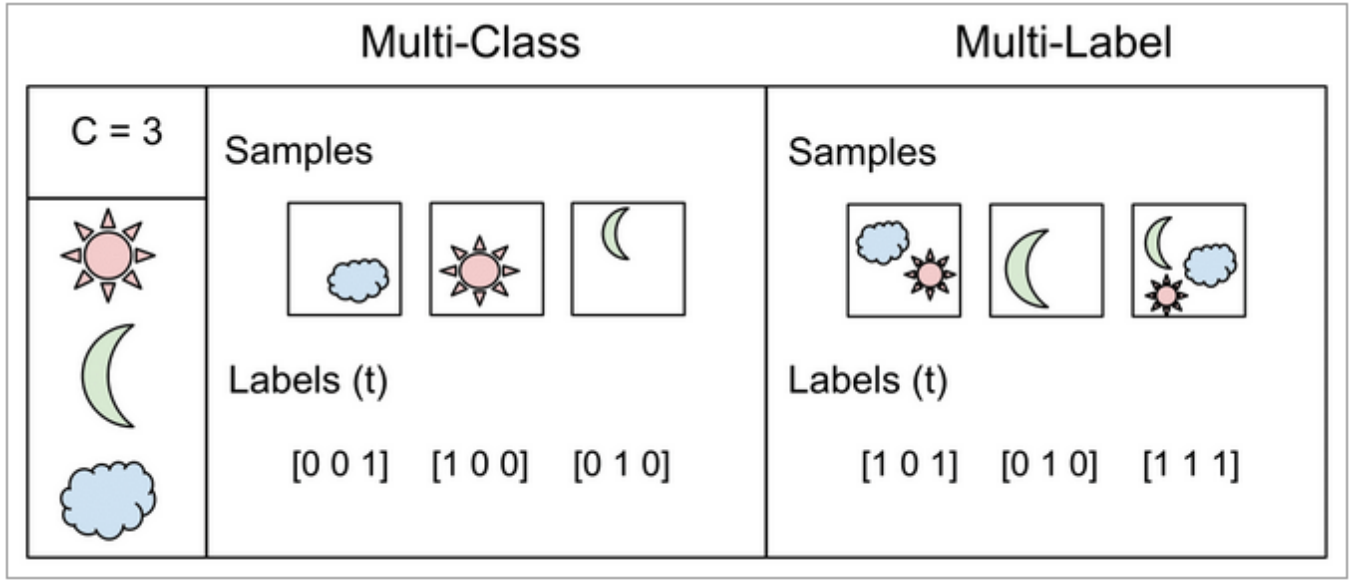
- A text might be about any of religion, politics, finance or education at the same time or none of these.
- A movie can be categorized into action, comedy and romance genre based on its summary content. There is possibility that a movie falls into multiple genres like romcoms [romance & comedy].


MARCH 27 1978 MONDAY 8:00 P.M. JACKSON BROWNE
OCT 8 1978 SUNDAY 8:00 P.M. AN EVENING WITH GENESIS
Globetrotters 1983 WORLD TOUR MUNICIPAL AUDITORIUM JAN 2 1983 NASHVILLE, TN SUNDAY 2:30 P.M.
JUNE 28 1987
DIMENSIONS UNLIMITED Presents PARTY WITH THE STARS FEATURING KOOL & THE GANG / CAMEO MUNICIPAL AUDITORIUM DEC 28 1980 NASHVILLE, TN SUNDAY 7:00 P.M.
DIMENSIONS UNLIMITED, INC. In Association With SOUND SEVENTY PRODUCTIONS, INC. Present RUFUS / CHAKA KHAN MUNICIPAL AUDITORIUM MAR 14 1980 NASHVILLE, TN FRIDAY 7:30 P.M.
INTERNATIONAL TOUR CONSULTANTS & DICK GRIFFEY PRODUCTIONS Present TEDDY PENDERGRASS MUNICIPAL AUDITORIUM AUG 1 1979 NASHVILLE, TN WEDNESDAY 7:30 P.M.
JUNE 9 1985 7:30 P.M.
MUNICIPAL AUDITORIUM NASHVILLE, TENN. DEC 16 1978 SATURDAY 7:00 P.M. Sound Seventy Productions, Inc. in Association with Monarch Entertainment Presents Grateful Dead
MUNICIPAL AUDITORIUM NASHVILLE, TENN. NOV 22 1979 THURSDAY 8:00 P.M. Sound Seventy Productions, Inc. Presents BILLY JOEL
MUNICIPAL AUDITORIUM NASHVILLE, TENNESSEE AUG 17 1982 TUESDAY 8:00 P.M. Cumberland Concerts Presents POLICE
MUNICIPAL AUDITORIUM NASHVILLE, TENNESSEE NOV 28 1978 TUESDAY 8:00 P.M. Sound Seventy Productions, Inc. PRESENTS THE DOOBIE BROTHERS
MUNICIPAL AUDITORIUM NASHVILLE, TENNESSEE FRIDAY 8:00 P.M. SOUND SEVENTY PRODUCTIONS, INC. presents ALICE COOPER TAX INCLUDED JAN 2 1987
CHESAPEAKE CONCERTS Presents James Taylor MUNICIPAL AUDITORIUM AUG 6 1980 NASHVILLE, TN WEDNESDAY 8:00 P.M.
JERRY WEINTRAUB & CONCERTS WEST Present Bob Dylan MUNICIPAL AUDITORIUM DEC 2 1978 NASHVILLE, TN SATURDAY 8:00 P.M.
MUNICIPAL AUDITORIUM NASHVILLE, TENNESSEE JAN 11 1984 WEDNESDAY 7:30 P.M. Sound Seventy Productions, Inc. Present KISS — $10.50 —
SOUND SEVENTY PRODUCTIONS, INC. Presents BLACK SABBATH MUNICIPAL AUDITORIUM NOV 9 1978 NASHVILLE, TN THURSDAY 8:00 P.M.
MUNICIPAL AUDITORIUM NASHVILLE, TENNESSEE JAN 14 1978 SATURDAY 7:00 P.M. Sound Seventy Productions presents The Charlie Daniels Band VOLUNTEER JAM IV
IN CONCERT ★ ELVIS ★ MUNICIPAL AUDITORIUM JUL 01 1973 Nashville, Tenn. SUNDAY Doors Open-7 PM SHOWTIME-8:30 PM
MUNICIPAL AUDITORIUM NASHVILLE, TENN. DEC'BR 29 Tues. Eve. at 8:00 ADM. $6.00 DICK CLARK PRODUCTIONS PRESENTS JACKSON 5 1970
NPGONLINELTD.COM PRESENTS PRINCE HIT N' RUN TOUR 2000 NASHVILLE MUNICIPAL AUD. SAT NOV 25 2000 8:00PM
BEAVER PRODUCTIONS Presents ZZ TOP EXPECTORATION SUNDAY NOV. 25 1979 8:00 P.M. MUNICIPAL AUDITORIUM NASHVILLE, TENNESSEE
MUNICIPAL AUDITORIUM NASHVILLE, TENN. NOV 4 1977 FRIDAY 8:00 P.M. Sound Seventy Productions, Inc. presents CHICAGO
BEAVER PRODUCTIONS PRESENTS LOVERBOY MUNICIPAL AUDITORIUM JAN 27 1986 NASHVILLE, TN MONDAY 8:00 P.M.
TOM JONES SHOW Municipal Auditorium - Nashville, Tenn. JULY WEDNESDAY
SHOWDOWN PRODUCTIONS & W.G. ENTERTAINMENT PRESENT THE GAP BAND
BLUE OYSTER CULT

source

How is it different from **multi-class** classification problem?

In **Multi-class classification** each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time. Let us consider an example of three classes C= [“Sun, “Moon, Cloud”]. In multi-class each sample can belong to only one of C classes. In multi-label case each sample can belong to one or more than one class.



Source

Dataset

For our discussion we will use Kaggle's ***Toxic Comment Classification Challenge*** dataset consisting of a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

```
toxic, severe_toxic, obscene, threat, insult, identity_hate
```

Example:

```
"Hi! I am back again! Last warning! Stop undoing my edits or die!"
```

is labelled as [1,0,0,1,0,0]. Meaning it is both `toxic` and `threat`.

Please refer here for detailed EDA of the dataset.

Let us briefly discuss BERT

In Oct 2018, Google released a new language representation model called **BERT**, which stands for **Bidirectional Encoder Representations from Transformers.** BERT builds upon recent work in pre-training contextual representations — including Semi-supervised Sequence Learning, Generative Pre-Training, ELMo, and ULMFit. However, unlike these

previous models, BERT is the first *deeply bidirectional*, *unsupervised* language representation, pre-trained using only a plain text corpus (Wikipedia).

Pre-trained representations can either be ***context-free*** or ***contextual***

1. **Context-free** models such as word2vec or GloVe generate a single word embedding representation for each word in the vocabulary. For example, the word "*bank*" would have the same context-free representation in "*bank account*" and "*bank of the river.*"
2. **Contextual** models instead generate a representation of each word that is based on the other words in the sentence. Contextual representations can further be *unidirectional* or *bidirectional*. For example, in the sentence "*I accessed the bank account*," a unidirectional contextual model would represent "*bank*" based on "*I accessed the*" but not "*account*." However, BERT represents "*bank*" using both its previous and next context — "*I accessed the … account*" — starting from the very bottom of a deep neural network, making it deeply bidirectional.

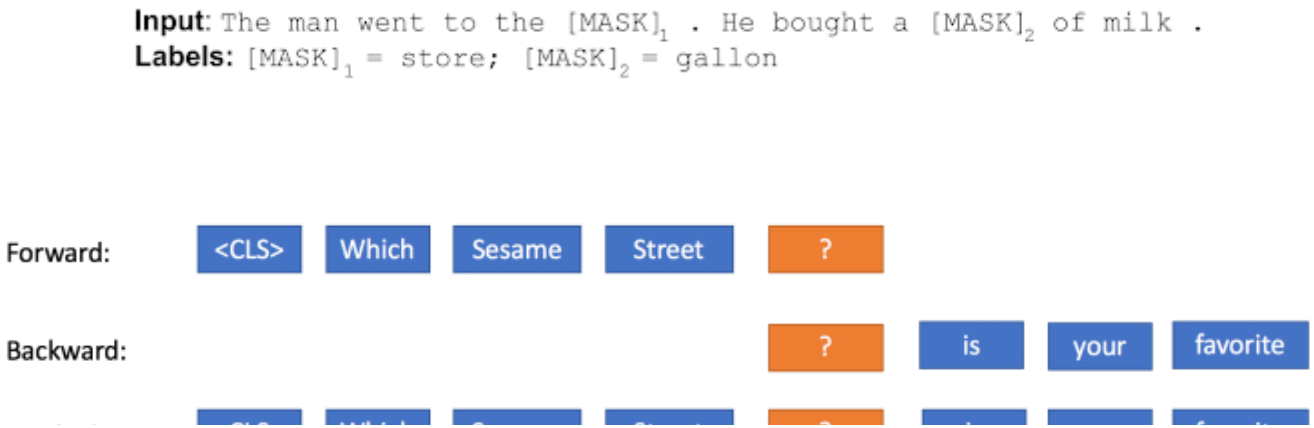
Bidirectional LSTM based language models train a standard left-to-right language model and also train a right-to-left (reverse) language model that predicts previous words from subsequent words like in ELMO. In ELMo,

there is a single LSTM for the forward language model and backward language model each. The crucial difference is that neither LSTM takes both the previous and subsequent tokens into account at the same time.

Why BERT is superior to other Bidirectional models?

Intuitively, a deep bidirectional model is strictly more powerful than either a left-to-right model or the concatenation of a left-to-right and right-to left model. Unfortunately, standard conditional language models can only be trained left-to-right or right-to-left, since bidirectional conditioning would allow each word to indirectly "see itself" in a multi-layered context.

To solve this problem, BERT uses "MASKING" technique to mask out some of the words in the input and then condition each word bidirectionally to predict the masked words. For example:



Forward, Backward, and Masked Language Modeling

BERT also learns to model relationships between sentences by pre-training on a very simple task that can be generated from any text corpus: Given two sentences A and B, is B the actual next sentence that comes after A in the corpus, or just a random sentence? For example:

Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

This is just a very basic overview of what BERT is. For details please refer to the original paper and some references[1], and [2].

Good News: Google has uploaded BERT to TensorFlow Hub which means we can directly use the pre-trained models for our NLP problems be it text classification or sentence similarity etc.

The example of predicting movie review, a binary classification problem is provided as an example code in the repository. In this article, we will focus on application of BERT to the problem of **multi-label text classification**.

So we will be basically modifying the example code and applying changes necessary to make it work for multi-label scenario.

Setup

Install the BERT using `!pip install bert-tensorflow`

Downloading pre-trained BERT models: These are the weights and other necessary files to represent the information BERT learned in pre-training. You'll need to pick which BERT pre-trained weights you want. There are two ways to download and use the pre-trained BERT model:

1. Directly using from the tensorflow-hub:

Following pre-trained models are available to choose from.

1. `BERT-Base, Uncased` : 12-layer, 768-hidden, 12-heads, 110M parameters
2. `BERT-Large, Uncased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
3. `BERT-Base, Cased` : 12-layer, 768-hidden, 12-heads , 110M parameters
4. `BERT-Large, Cased` : 24-layer, 1024-hidden, 16-heads, 340M parameters

5. `BERT-Base, Multilingual Case` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
6. `BERT-Base, Chinese` : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

We will use basic model: **'uncased_L-12_H-768_A-12'**

```
BERT_MODEL_HUB = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"
```

2. Manually Download the BERT model files : Download and save into a directory and unzip it. Here are links to the files for English:

- `BERT-Base, Uncased, BERT-Base, Cased,`
- `BERT-Large, Cased, BERT-Large, Uncased`

You can use either way, but let us see what are the files actually in the pre-trained models. When I download `BERT-Base, Uncased,` these are 3 important files as follows:

```
BERT_VOCAB= 'uncased-l12-h768-a12/vocab.txt'
BERT_INIT_CHKPNT = 'uncased-l12-h768-a12/bert_model.ckpt'
```

`BERT_CONFIG = 'uncased_l12_h768_a12/bert_config.json'`

`BERT_VOCAB :` Contains model vocabulary [words to indexes mapping]

`BERT_INIT_CHKPNT` : Contains weights of the pre-trained model

`BERT_CONFIG` : Contains BERT model architecture.

Tokenization

Tokenization involves breaking up of input text into its individual words. In order to do so, the first step is to create the tokenizer object. Two ways we can do that:

1. Directly from the tensorflow-hub

```
BERT_MODEL_HUB = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"
tokenizer = run_classifier_with_tfhub.create_tokenizer_from_hub_module(BERT_MODEL_HUB)
```

2. From manually downloaded files:

Using **`BERT_INIT_CHKPNT & BERT_VOCAB files`**

```
tokenization.validate_case_matches_checkpoint(True,BERT_INIT_CHKPNT)
tokenizer = tokenization.FullTokenizer(
      vocab_file=BERT_VOCAB, do_lower_case=True)
```

After you have created the tokenizer, it is time to use it. Let us tokenize sentence: “This here’s an example of using the BERT tokenizer”

```
tokenizer.tokenize("This here's an example of using the BERT tokenizer")
```

```
['this',
 'here',
 "'",
 's',
 'an',
 'example',
 'of',
 'using',
 'the',
 'bert',
 'token',
 '##izer']
```

Size of the vocabulary: ~30K

```
len(tokenizer.vocab)
```

```
30522
```

Data preprocessing:

Let us first read the data set provided :

```
train_data_path='../input/jigsaw-toxic-comment-classification-challenge/train.csv'
train = pd.read_csv(train_data_path)
```

train.head()

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

We need to convert our data into a format that BERT understands. Some utility functions are provided to do that.

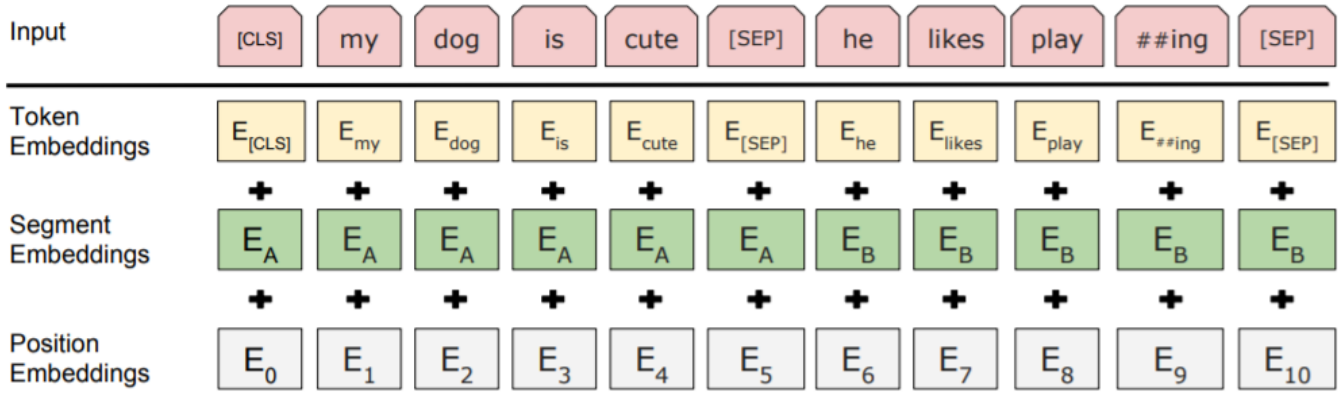
```
class InputExample(object):
```

```
    """A single training/test example for simple sequence classification."""

    def __init__(self, guid, text_a, text_b=None, labels=None):
        """Constructs a InputExample.

        Args:
            guid: Unique id for the example.
            text_a: string. The untokenized text of the first sequence. For single
            sequence tasks, only this sequence must be specified.
            text_b: (Optional) string. The untokenized text of the second sequence.
            Only must be specified for sequence pair tasks.
            labels: (Optional) [string]. The label of the example. This should be
            specified for train and dev examples, but not for test examples.
        """
        self.guid = guid
        self.text_a = text_a
        self.text_b = text_b
        self.labels = labels
```

```
def create_examples(df, labels_available=True):
    """Creates examples for the training and dev sets."""
    examples = []
    for (i, row) in enumerate(df.values):
        guid = row[0]
        text_a = row[1]
        if labels_available:
            labels = row[2:]
        else:
            labels = [0,0,0,0,0,0]
        examples.append(
            InputExample(guid=guid, text_a=text_a, labels=labels))
    return examples
```

`create_examples()`, reads data-frame and loads input text and corresponding target labels into `InputExample` objects.

Using tokenizer, we'll call `convert_examples_to_features` method on our examples to convert them into features BERT understands. This method adds the special "CLS" and "SEP" tokens used by BERT to identify sentence start and end. It also appends "index" and "segment" tokens to each input. So all the job of formatting input as per the BERT is done by this function.



BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Creating Model

Here we use the pre-trained BERT model and fine-tune it for our classification task. Basically we load the pre-trained model and then train

the last layer for classification task.

```
def create_model(bert_config, is_training, input_ids, input_mask, segment_ids,
                 labels, num_labels, use_one_hot_embeddings):
    """Creates a classification model."""
    model = modeling.BertModel(
        config=bert_config,
        is_training=is_training,
        input_ids=input_ids,
        input_mask=input_mask,
        token_type_ids=segment_ids,
        use_one_hot_embeddings=use_one_hot_embeddings)

    output_layer = model.get_pooled_output()

    hidden_size = output_layer.shape[-1].value

    output_weights = tf.get_variable(
        "output_weights", [num_labels, hidden_size],
        initializer=tf.truncated_normal_initializer(stddev=0.02))

    output_bias = tf.get_variable(
        "output_bias", [num_labels], initializer=tf.zeros_initializer())

    with tf.variable_scope("loss"):
        if is_training:
            # I.e., 0.1 dropout
            output_layer = tf.nn.dropout(output_layer, keep_prob=0.9)

        logits = tf.matmul(output_layer, output_weights, transpose_b=True)
        logits = tf.nn.bias_add(logits, output_bias)

        probabilities = tf.nn.sigmoid(logits)
        labels = tf.cast(labels, tf.float32)

        per_example_loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=labels, logits=logits)
        loss = tf.reduce_mean(per_example_loss)

        return (loss, per_example_loss, logits, probabilities)
```

In multi-label classification instead of `softmax()`, we use `sigmoid()` to get the probabilities.

- In simple binary classification, there's no big difference between the two, however in case of multinational classification, sigmoid allows to deal with non-exclusive labels (a.k.a. *multi-labels*), while softmax deals with exclusive classes.
- A *logit* (also called a score) is a raw unscaled value associated with a class, before computing the probability. In terms of neural network architecture, this means that a logit is an output of a dense (fully-connected) layer [3].

So, to compute probabilities, we make the following change:

```
### multi-class case: probabilities = tf.nn.softmax(logits)
### multi-label case: probabilities = tf.nn.sigmoid(logits)
```

To compute per example loss, tensorflow provides another method:

tf.nn.sigmoid_cross_entropy_with_logits Measures the probability error in discrete classification tasks in which each class is independent and not mutually

exclusive. This is suitable for multi-label classification problems[4].

Rest of the code is mostly from the BERT reference[5]. The complete code is available at github.

Kaggle Submission Score:

Submission and Description	Private Score	Public Score
sample_submission.csv a few seconds ago by Javaid Nabi epochs 2	0.98475	0.98509

Just by running 2 epochs, got very good results. This is the power of **transfer learning**: using pre-trained model which has been trained on a huge dataset and then fine-tuning it for a specific task. Kaggle code here.

So try it out on some other dataset and run for few epochs[3–4] and see the results.

Thanks for reading.

References

[1] https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

[2] https://mlexplained.com/2019/01/07/paper-dissected-bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding-explained/

[3] https://stackoverflow.com/questions/47034888/how-to-choose-cross-entropy-loss-in-tensorflow

[4]https://www.tensorflow.org/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits

[5] https://github.com/google-research/bert/blob/master/run_classifier.py

[6] https://www.depends-on-the-definition.com/guide-to-multi-label-classification-with-neural-networks/

[7] https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

[8] https://gombru.github.io/2018/05/23/cross_entropy_loss/

Machine Learning Text Classification Multi Label Bert Deep Learning