**Title: BERT Analysis For Classification**

**Introduction:**

BERT (Bidirectional Encoder Representations from Transformers) is a language representation model developed by Google Research. BERT is a successful example from the most recent generation of deep learning-based.

Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

**Literature survey:**

BERT was built upon recent work in pre-training contextual representations but crucially these models are all *unidirectional* or *shallowly bidirectional*. The following are some of the major pre-trained models of BERT which have been developed till now:

* [BERT-Large, Uncased (Whole Word Masking)](https://storage.googleapis.com/bert_models/2019_05_30/wwm_uncased_L-24_H-1024_A-16.zip): 24-layer, 1024-hidden, 16-heads, 340M parameters
* [BERT-Large, Cased (Whole Word Masking)](https://storage.googleapis.com/bert_models/2019_05_30/wwm_cased_L-24_H-1024_A-16.zip): 24-layer, 1024-hidden, 16-heads, 340M parameters
* [BERT-Base, Uncased](https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip): 12-layer, 768-hidden, 12-heads, 110M parameters
* [BERT-Large, Uncased](https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-24_H-1024_A-16.zip): 24-layer, 1024-hidden, 16-heads, 340M parameters
* [BERT-Base, Cased](https://storage.googleapis.com/bert_models/2018_10_18/cased_L-12_H-768_A-12.zip): 12-layer, 768-hidden, 12-heads , 110M parameters
* [BERT-Large, Cased](https://storage.googleapis.com/bert_models/2018_10_18/cased_L-24_H-1024_A-16.zip): 24-layer, 1024-hidden, 16-heads, 340M parameters
* [BERT-Base, Multilingual Cased (New, recommended)](https://storage.googleapis.com/bert_models/2018_11_23/multi_cased_L-12_H-768_A-12.zip): 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
* [BERT-Base, Multilingual Uncased (Orig, not recommended)](https://storage.googleapis.com/bert_models/2018_11_03/multilingual_L-12_H-768_A-12.zip) (Not recommended, use Multilingual Cased instead): 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
* [BERT-Base, Chinese](https://storage.googleapis.com/bert_models/2018_11_03/chinese_L-12_H-768_A-12.zip): Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

Following is the [GLUE Score](https://mccormickml.com/2019/11/05/GLUE/) analysis of these models:

| Model | Score | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI-m | MNLI-mm | QNLI(v2) | RTE | WNLI | AX |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BERT-Tiny | 64.2 | 0.0 | 83.2 | 81.1/71.1 | 74.3/73.6 | 62.2/83.4 | 70.2 | 70.3 | 81.5 | 57.2 | 62.3 | 21.0 |
| BERT-Mini | 65.8 | 0.0 | 85.9 | 81.1/71.8 | 75.4/73.3 | 66.4/86.2 | 74.8 | 74.3 | 84.1 | 57.9 | 62.3 | 26.1 |
| BERT-Small | 71.2 | 27.8 | 89.7 | 83.4/76.2 | 78.8/77.0 | 68.1/87.0 | 77.6 | 77.0 | 86.4 | 61.8 | 62.3 | 28.6 |
| BERT-Medium | 73.5 | 38.0 | 89.6 | 86.6/81.6 | 80.4/78.4 | 69.6/87.9 | 80.0 | 79.1 | 87.8 | 62.2 | 62.3 | 30.5 |

We use HuggingFace Algorithm for our analysis which is based on [BERT-Base, Uncased](https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip)

**Data set:**

<https://drive.google.com/file/d/1tsMRKA9LuMbXUSORigAUDsP0AXjsAixU/view?usp=sharing>

The Youtube video dataset contains almost 4000 videos with the following format for classification:

| link | title | description | category |
| --- | --- | --- | --- |
| Video ID | Title of the video | Description of the video | Category of the video |

The category/classes are:

1. Travel Vlogs
2. Food
3. Art and Music
4. History

**Method:**

We can use a pre-trained BERT model and then leverage [transfer learning](https://en.wikipedia.org/wiki/Transfer_learning) as a technique to solve specific NLP tasks.

Transfer learning is key here because training BERT from scratch is very hard. The original BERT model was pre-trained with a combined text corpus containing about 3.3 billion words. The pre-training takes about 4 days to complete on 16 TPU chips, whereas most fine-tuning procedures from pre-trained models will take about one to few hours to run on a single GPU.

This process can be implemented with the following tasks:

1. **Choose a pre-trained BERT model according to the language needs for our task.**

For our task we choose the [*distilbert-base-uncased*](https://github.com/huggingface/transformers/tree/master/examples/distillation), which is pre-trained on the same data used to pre-train BERT using a technique known as [knowledge distillation](https://medium.com/neuralmachine/knowledge-distillation-dc241d7c2322) with the supervision of the bert-base-uncased version of BERT.

**The model has 6 layers**, 768 dimensions and 12 heads, totalizing 66M parameters. It can be trained 60% faster than the original **uncased base BERT,** which **has 12 layers** and approximately 110M parameters, while preserving 97% of the model performance.

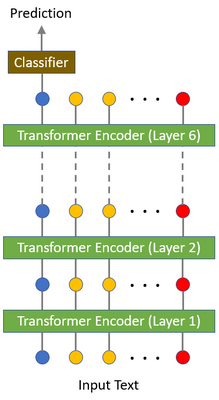
1. **Modify the pre-trained model architecture to fit our specific task.**

BERT was designed to be pre-trained in an unsupervised way to perform two tasks: masked language modeling and next sentence prediction.

In the masked language modeling, some percentage of the input tokens are masked at random and the model is trained to predict those masked tokens at the output. For the next sentence prediction task, the model is trained for a binary classification task by choosing pairs of sentences A and B.

In our specific task, we need to modify the base BERT model to perform text classification. This can be done by feeding the first output token of the last transformer layer into a classifier of our choice. That first token at the output layer is an aggregate sequence representation of an entire sequence that is fed as input to the model.

The package we use in our implementation already has several modified BERT models to perform different tasks, including one for text classification, so we don’t need to plug a custom classifier.

 High-level overview of the modified BERT model to perform text classification

1. **Prepare the training data according to our specific task.**

To work with BERT, we also need to prepare our data according to what the model architecture expects. For the text classification task, the input text needs to be prepared as following:

1. Tokenize text sequences according to the [WordPiece](https://medium.com/@makcedward/how-subword-helps-on-your-nlp-model-83dd1b836f46). In this specification, tokens can represent words, sub-words, or even single characters. For example, the word 'requisitions' is tokenized as ['re', '##qui', '##sit', '##ions']. Here, the two hash signs preceding some sub-words denote that a sub-word is part of a larger word and preceded by another sub-word.
2. Truncate and pad your sequences to the maximum sequence length suitable for your task, respecting the hard limit of 512 tokens per sequence according to the BERT specification.
3. Annotate your tokenized sequences with the special tokens '[CLS]' and '[SEP]' to mark the beginning and end of each sequence, respectively.
4. Convert your tokenized sequences into sequences of indices that are specific for the BERT vocabulary.
5. Create a sequence mask to indicate which elements in a sequence are tokens and which are paddings.
6. Create the numeric sequential array to be used for the positional embeddings, which is required by the [transformer](https://arxiv.org/abs/1706.03762).
7. **Fine-tune the modified pre-trained model by further training it using our own dataset.**

After choosing and instantiating a pre-trained BERT model and preparing our data for model training and validation, we can finally perform the model fine-tuning. This is very similar to training a model from scratch, except usually for fine-tuning we have far less training data, less hyperparameters to tune, and we can train for a couple of epochs only in order to get good results.

**Result analysis:**

We split the data into 80% for training, 10% for validation, and 10% for testing.

The following is the obtained confusion matrix for YouTube dataset:

|  | **precision** | **recall** | **f1-score** | **support** |
| --- | --- | --- | --- | --- |
| **art\_music** | 0.90 | 0.96 | 0.93 | 101 |
| **food** | 0.94 | 0.83 | 0.88 | 96 |
| **history** | 0.98 | 0.81 | 0.88 | 57 |
| **travel** | 0.82 | 0.93 | 0.87 | 105 |
|  |  |  |  |  |
| **accuracy** |  |  | 0.89 | 359 |
| **Macro avg** | 0.91 | 0.88 | 0.89 | 359 |
| **Weighted avg** | 0.90 | 0.89 | 0.89 | 359 |

**Test Accuracy: 0.8941504178272981**

**Reference:**

<https://arxiv.org/abs/1810.04805>

<https://towardsdatascience.com/a-review-of-bert-based-models-4ffdc0f15d58>

<https://mccormickml.com/2019/11/05/GLUE/>