WARNING

Please make sure to "COPY AND EDIT NOTEBOOK" to use compatible library dependencies! DO NOT CREATE A NEW NOTEBOOK AND COPY+PASTE THE CODE - this will use latest Kaggle dependencies at the time you do that, and the code will need to be modified to make it work. Also make sure internet connectivity is enabled on your notebook

Preliminaries

First install critical dependencies not already on the Kaggle docker image. **NOTE THAT THIS NOTEBOOK USES TENSORFLOW 1.14 IN ORDER TO BE COMPARED WITH ELMo, WHICH WAS NOT PORTED TO TENSORFLOW 2.X. To see equivalent Tensorflow 2.X BERT Code for the Spam problem, see https://www.kaggle.com/azunre/tlfornlp-chapters2-3-spam-bert-tf2**

In [2]:
```
!pip install keras==2.2.4 # critical dependency
!pip install -q bert-tensorflow==1.0.1
```

```
Collecting keras==2.2.4
  Downloading https://files.pythonhosted.org/packages/5e/10/aa32dad071ce52b550226
6b5c659451cfd6ffcbf14e6c8c4f16c0ff5aaab/Keras-2.2.4-py2.py3-none-any.whl (312kB)
     |████████████████████████████████| 317kB 4.3MB/s eta 0:00:01
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.6/site-packages
(from keras==2.2.4) (5.1.2)
Requirement already satisfied: scipy>=0.14 in /opt/conda/lib/python3.6/site-packa
ges (from keras==2.2.4) (1.2.1)
Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.6/site-packag
es (from keras==2.2.4) (1.12.0)
Requirement already satisfied: numpy>=1.9.1 in /opt/conda/lib/python3.6/site-pack
ages (from keras==2.2.4) (1.16.4)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /opt/conda/lib/pytho
n3.6/site-packages (from keras==2.2.4) (1.1.0)
Requirement already satisfied: h5py in /opt/conda/lib/python3.6/site-packages (fr
om keras==2.2.4) (2.9.0)
Requirement already satisfied: keras-applications>=1.0.6 in /opt/conda/lib/python
3.6/site-packages (from keras==2.2.4) (1.0.8)
Installing collected packages: keras
  Found existing installation: Keras 2.3.0
    Uninstalling Keras-2.3.0:
      Successfully uninstalled Keras-2.3.0
Successfully installed keras-2.2.4
```

Write requirements to file, anytime you run it, in case you have to go back and recover Kaggle dependencies. **MOST OF THESE REQUIREMENTS WOULD NOT BE NECESSARY FOR LOCAL INSTALLATION**

Latest known such requirements are hosted for each notebook in the companion github repo, and can be pulled down and installed here if needed. Companion github repo is located at https://github.com/azunre/transfer-learning-for-nlp

In [3]:
```
!pip freeze > kaggle_image_requirements.txt
```

In [4]:
```
# Import neural network libraries
import tensorflow as tf
import tensorflow_hub as hub
from bert.tokenization import FullTokenizer
from tensorflow.keras import backend as K

# Initialize session
sess = tf.Session()
```

In [5]:
```
# Some other key imports
import os
import re
import pandas as pd
import numpy as np
from tqdm import tqdm
```

Define Tokenization, Stop-word and Punctuation Removal Functions

Before proceeding, we must decide how many samples to draw from each class. We must also decide the maximum number of tokens per email, and the maximum length of each token. This is done by setting the following overarching hyperparameters

Tokenization

In [6]:
```
def tokenize(row):
    if row is None or row is '':
        tokens = ""
    else:
        try:
            tokens = row.split(" ")[:maxtokens]
        except:
            tokens=""
    return tokens
```

In [23]:
```
# Params for bert model and tokenization
Nsamp = 1000 # number of samples to generate in each class - 'spam', 'not spam'
maxtokens = 230 # the maximum number of tokens per document
maxtokenlen = 200 # the maximum length of each token
```

Use regular expressions to remove unnecessary characters

Next, we define a function to remove punctuation marks and other nonword characters (using regular expressions) from the emails with the help of the ubiquitous python regex library. In the same step, we truncate all tokens to hyperparameter maxtokenlen defined above.

In [24]:
```
def reg_expressions(row):
    tokens = []
    try:
```

```
            for token in row:
                token = token.lower()
                token = re.sub(r'[\W\d]', "", token)
                token = token[:maxtokenlen] # truncate token
                tokens.append(token)
        except:
            token = ""
            tokens.append(token)
        return tokens
```

Stop-word removal

Let's define a function to remove stopwords - words that occur so frequently in language that they offer no useful information for classification. This includes words such as "the" and "are", and the popular library NLTK provides a heavily-used list that will employ.

In [25]:
```
import nltk

nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords = stopwords.words('english')

# print(stopwords) # see default stopwords
# it may be beneficial to drop negation words from the removal list, as they can
# of a sentence - but we didn't find it to make a difference for this problem
# stopwords.remove("no")
# stopwords.remove("nor")
# stopwords.remove("not")
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [26]:
```
def stop_word_removal(row):
    token = [token for token in row if token not in stopwords]
    token = filter(None, token)
    return token
```

Download and Assemble IMDB Review Dataset

Download the labeled IMDB reviews

In [27]:
```
!wget -q "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
!tar xzf aclImdb_v1.tar.gz
```

In [11]:
```
!wget -q "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
!tar xzf aclImdb_v1.tar.gz
```

Shuffle and preprocess data

In [28]:
```
# function for shuffling data
def unison_shuffle(data, header):
    p = np.random.permutation(len(header))
    data = data[p]
    header = np.asarray(header)[p]
```

```
    return data, header

def load_data(path):
    data, sentiments = [], []
    for folder, sentiment in (('neg', 0), ('pos', 1)):
        folder = os.path.join(path, folder)
        for name in os.listdir(folder):
            with open(os.path.join(folder, name), 'r') as reader:
                  text = reader.read()
            text = tokenize(text)
            text = stop_word_removal(text)
            text = reg_expressions(text)
            data.append(text)
            sentiments.append(sentiment)
    data_np = np.array(data)
    data, sentiments = unison_shuffle(data_np, sentiments)

    return data, sentiments

train_path = os.path.join('aclImdb', 'train')
test_path = os.path.join('aclImdb', 'test')
raw_data, raw_header = load_data(train_path)

print(raw_data.shape)
print(len(raw_header))
```

```
(25000,)
25000
```

In [29]:
```
# Subsample required number of samples
random_indices = np.random.choice(range(len(raw_header)),size=(Nsamp*2,),replace
data_train = raw_data[random_indices]
header = raw_header[random_indices]

print("DEBUG::data_train::")
print(data_train)
```

```
DEBUG::data_train::
[list(['in', 'spirit', 'classic', 'the', 'sting', 'movie', 'hits', 'truly', 'hurt
s', 'heart', 'a', 'prim', 'proper', 'female', 'psychiatrist', 'hungry', 'adventur
e', 'meets', 'dirtiest', 'rottenest', 'scoundrals', 'the', 'vulnerable', 'docto
r', 'falls', 'career', 'badman', 'begs', 'involved', 'operation', 'while', 'movi
e', 'moves', 'kind', 'slow', 'climax', 'ending', 'stunning', 'youll', 'especiall
y', 'enjoy', 'doctor', 'forgives', 'herself'])
 list(['this', 'truly', 'terrible', 'filmbr', 'br', 'im', 'writing', 'people', 's
omewhere', 'put', 'watching', 'it', 'if', 'i', 'stopped', 'one', 'person', 'wasti
ng', 'precious', 'life', 'watching', 'film', 'i', 'shall', 'die', 'happybr', 'b
r', 'unutterably', 'dull', 'although', 'since', 'stars', 'al', 'pacino', 'i', 'fo
oled', 'thinking', 'moment', 'something', 'interesting', 'going', 'happen', 'the
n', 'credits', 'rolled', 'i', 'realised', 'i', 'completely', 'fooled', 'watchin
g', 'unbelievable', 'drivelbr', 'br', 'i', 'cannot', 'believe', 'film', 'achieve
d', 'high', 'score', 'imdb', 'over', '', 'stars', 'i', 'last', 'saw', 'voting',
'are', 'people', 'voting', 'ironicallybr', 'br', 'please', 'please', 'please', 'w
atch', 'film'])
 list(['show', 'boobies', 'funny', 'certainly', 'channel', 'shows', 'cartoons',
'understand', 'im', 'coming', 'from', 'i', 'want', '', '', 'years', 'old', 'daugh
ters', 'thinking', 'like', 'hearing', 'that', 'i', 'find', 'sad', 'nick', 'hype
d', 'crap', 'that', 'much', 'thats', 'get', 'stupid', 'little', 'kids', 'acting',
'like', 'stupid', 'adults', 'i', 'know', 'meant', 'humorous', 'consider', 'swee
t', 'little', 'innocent', 'girls', 'k', 'st', 'grade', 'cant', 'wait', 'see', 'th
is', 'i', 'comment', 'disappointed', 'i', 'i', 'saw', 'it', 'my', 'daughters', 'w
atching', 'it', 'id', 'love', 'block', 'nick', 'heart', 'point', 'nick', 'keeps',
'putting', 'kind', 'crap', 'ill', 'to'])
 ...
 list(['this', 'movie', 'bad', 'i', 'cannot', 'even', 'begin', 'describe', 'it',
'what', 'blazing', 'pit', 'wrong', 'writers', 'producers', 'director', 'how', 'ea
rth', 'get', 'funding', 'abomination', 'the', 'plot', 'laughable', 'acting', 'poo
r', 'best', 'story', 'what', 'story', 'the', 'first', 'fight', 'movie', 'ok', 'ke
eps', 'repeating', 'want', 'turn', 'offbr', 'br', 'i', 'guess', 'im', 'biggest',
'looser', 'turning', 'stupid', 'movie', 'first', 'minutebr', 'br', '', 'spoler',
'alert', 'br', 'br', 'i', 'saw', 'movie', 'scott', 'adkins', 'it', 'it', '', 'sec
ondsbr', 'br', 'i', 'give', '', '', 'lowest', 'grade', 'imdb', 'offerbr', 'br',
'do', 'favour', 'see', 'uwe', 'boll', 'movie', 'instead', 'twice', 'worthy', 'tim
e'])
 list(['one', 'favorite', 'scenes', 'beginning', 'guests', 'private', 'yacht', 'd
ecide', 'take', 'impromptu', 'swim', '', 'underwear', 'rather', 'risqué', ''])
 list(['let', 'start', 'saying', 'war', 'inc', 'everyones', 'cup', 'tea', 'it',
'is', 'however', 'enjoyable', 'and', 'gets', 'thinking', '', 'oh', 'crap', 'the',
'comedy', 'involved', 'film', 'obvious', '', 'quite', 'subtle', 'tamerlane', 'tan
ks', 'drycleaning', 'service', 'etc', 'changes', 'twists', '', 'turns', 'plotbr',
'br', 'i', 'may', 'one', 'i', 'compare', 'grosse', 'point', 'blank', 'because',
'different', 'john', 'cusack', '', 'i', 'say', 'amazing', 'brilliant', '', 'goo
d', 'on', 'hand', 'sister', 'joan', 'cusack', 'incredible', 'delivery', 'lines',
'', 'comedic', 'timing', '', 'even', 'though', 'hardly', 'film', 'id', 'say', 'be
n', 'kingsleybr', 'br', 'marisa', 'tomei', 'plays', 'convincing', 'reporter', 'ma
nages', 'pull', 'off', 'hilary', 'duff', 'commendable', 'role', 'central', 'asia
n', 'pop', 'star', 'yonica', 'babyyeah', 'duffs', 'development', 'actress', 'noti
ceable', 'film', 'good', 'job', 'even', 'though', 'accent', 'tad', 'unrealbr', 'b
r', 'overall', 'film', 'i', 'would', 'call', 'entertaining', 'it', 'particular',
'storyline', 'quite', 'silly', 'times', 'subtle', 'message', 'id', 'say', 'wort
h', 'watch'])]
```

Display sentiments and their frequencies in the dataset, to ensure it is roughly balanced between classes

In [30]:
```
unique_elements, counts_elements = np.unique(header, return_counts=True)
print("Sentiments and their frequencies:")
print(unique_elements)
print(counts_elements)
```

```
Sentiments and their frequencies:
[0 1]
[ 979 1021]
```

In [31]:
```
# function for converting data into the right format, due to the difference in r
# we expect a single string per email here, versus a list of tokens for the skle
def convert_data(raw_data,header):
    converted_data, labels = [], []
    for i in range(raw_data.shape[0]):
        # combine list of tokens representing each email into single string
        out = ' '.join(raw_data[i])
        converted_data.append(out)
        labels.append(header[i])
    converted_data = np.array(converted_data, dtype=object)[:, np.newaxis]

    return converted_data, np.array(labels)

data_train, header = unison_shuffle(data_train, header)

# split into independent 70% training and 30% testing sets
idx = int(0.7*data_train.shape[0])
# 70% of data for training
train_x, train_y = convert_data(data_train[:idx],header[:idx])
# remaining 30% for testing
test_x, test_y = convert_data(data_train[idx:],header[idx:])

print("train_x/train_y list details, to make sure it is of the right form:")
print(len(train_x))
print(train_x)
print(train_y[:5])
print(train_y.shape)
```

```
train_x/train_y list details, to make sure it is of the right form:
1400
[['i think part reason movie madeand aimed us gamers actually play nancy drew pc
games theres lot movies lately based video games i think one thembr br so movie f
ollow book but follow parts games i buy play every nancy drew games soon comes ou
t and games herinteractive girls afraid mouse and games actually parents choice g
old awards they fun actually learn thing two playingbr br i took two step childre
n go see loved it the  yr old started playing first nancy drew game day i took se
e movie much fun playing game i thought would enjoy movie well and i rightshe lov
ed movie wait get home finish first game start another onebr br my step daughter
also loved movie still little young play games yet enjoys']
 ['i know film meager rating imdb this film accompanied i curious blue masterwork
br br the thing let film like process film like psychology expecting hardcore por
nographic rammingbr br this film want watch unwind film want see like masterpiece
time attention carebr br summaries may contain a spoiler or twobr br the main thi
ng film blends whole film within film thing way sometimes forget fictions realbr
br the film like many films onebr br  a political documentary social system swede
n time which lot ways still relevant today interviews done young woman named lena
br br  a narrative filmmaker vilgot sjoman making film deals relationship star fi
lm never got involved people hes supposed work withbr br  the film vilgot making
its young woman named lenaie  young']
 ['or least one best i think fun cool game n bowser usual shenanigans yeah dumb w
ord one i think of mario must stop again this game fun play contains lots nostalg
ia me the bad thing graphics awful todays standards everything else pretty good e
specially little minigames unlock its second best n game the first best conkers b
ad fur day i recommend mario fan fan platform games it beats mediocre super mario
sunshine daybr br  or a']
 ...
 ['ok i really much say film this i seen  films life  horror films while i diffic
ulty deciding best as opposed favourite i tell george a romeros dawn of the dead
i tell without slightest hesitation todd sheets zombie bloodbath absolute worst h
orror film i ever seenbr br there simply nothing positive i say film the acting d
ialogue directing makeup music every aspect film simply far acceptable boggles mi
nd ever even releasedbr br even horror zombie movie completist please heed warnin
g do not waste time garbage there pleasure gotten viewing this you even get laugh
s utter ineptitude display trust me please']
 ['nahhh leila grace mills teenager turned satan and lsd archaeologist fiancé ric
hard theres neighborhood hippie demon cult hanging local decrepit ancient castle
leila richard drink blood drop drugs join sex orgies dance lame psychedelic rock
participate black mass ceremonies guys wear pants masks capes women wear anything
all unfortunately kind extracurricular activities left leila open demonic possess
ion dreaded spirit evil leila also comes screwed family help either her older bro
ther john recluse seems love her her mother patricia maria perschy depressed thin
ks responsible fathers death to top off sister maria kosti slutty semi pro golfer
named gasp debbie gibson theres plenty hired help around also waste time two maid
s a young one takes clothes lot old one spies everyone plus udo luis induni bald
voyeuristic handyman spies leila changing clothes takes nude pictures sneaks pool
house take sniff freshly used bathing suit oh yeah borg pet german shepherdbr br
paul naschy']
 ['a gritty gutsy portrayal part world war  history us us hadhave idea ever occur
red i would love video it shown tv one time far i know back or  i asked around mo
vie video stores even know it great actresses around wish i could see again top n
otch series']]
[1 1 1 1 0]
(1400,)
```

Build, Train and Evaluate BERT Model

First define critical functions that define various components of the BERT model

In [32]:
```
class InputExample(object):
    """A single training/test example for simple sequence classification."""

    def __init__(self, guid, text_a, text_b=None, label=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.
      label: (Optional) string. The label of the example. This should be
        specified for train examples, but not for test examples.
    """
        self.guid = guid
        self.text_a = text_a
        self.text_b = text_b
        self.label = label


def create_tokenizer_from_hub_module(bert_path):
    """Get the vocab file and casing info from the Hub module."""
    bert_module = hub.Module(bert_path)
    tokenization_info = bert_module(signature="tokenization_info", as_dict=True)
    vocab_file, do_lower_case = sess.run(
        [tokenization_info["vocab_file"], tokenization_info["do_lower_case"]]
    )

    return FullTokenizer(vocab_file=vocab_file, do_lower_case=do_lower_case)


def convert_single_example(tokenizer, example, max_seq_length=256):
    """Converts a single `InputExample` into a single `InputFeatures`."""

    tokens_a = tokenizer.tokenize(example.text_a)
    if len(tokens_a) > max_seq_length - 2:
        tokens_a = tokens_a[0 : (max_seq_length - 2)]

    tokens = []
    segment_ids = []
    tokens.append("[CLS]")
    segment_ids.append(0)
    for token in tokens_a:
        tokens.append(token)
        segment_ids.append(0)
    tokens.append("[SEP]")
    segment_ids.append(0)

    input_ids = tokenizer.convert_tokens_to_ids(tokens)

    # The mask has 1 for real tokens and 0 for padding tokens. Only real
    # tokens are attended to.
    input_mask = [1] * len(input_ids)

    # Zero-pad up to the sequence length.
    while len(input_ids) < max_seq_length:
        input_ids.append(0)
```

```
        input_mask.append(0)
        segment_ids.append(0)

    assert len(input_ids) == max_seq_length
    assert len(input_mask) == max_seq_length
    assert len(segment_ids) == max_seq_length

    return input_ids, input_mask, segment_ids, example.label


def convert_examples_to_features(tokenizer, examples, max_seq_length=256):
    """Convert a set of `InputExample`s to a list of `InputFeatures`."""

    input_ids, input_masks, segment_ids, labels = [], [], [], []
    for example in tqdm(examples, desc="Converting examples to features"):
        input_id, input_mask, segment_id, label = convert_single_example(
            tokenizer, example, max_seq_length
        )
        input_ids.append(input_id)
        input_masks.append(input_mask)
        segment_ids.append(segment_id)
        labels.append(label)
    return (
        np.array(input_ids),
        np.array(input_masks),
        np.array(segment_ids),
        np.array(labels).reshape(-1, 1),
    )


def convert_text_to_examples(texts, labels):
    """Create InputExamples"""
    InputExamples = []
    for text, label in zip(texts, labels):
        InputExamples.append(
            InputExample(guid=None, text_a=" ".join(text), text_b=None, label=la
        )
    return InputExamples
```

Next, we define a custom tf hub BERT layer

In [33]:

```
class BertLayer(tf.keras.layers.Layer):
    def __init__(
        self,
        n_fine_tune_layers=10,
        pooling="mean",
        bert_path="https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1",
        **kwargs,
    ):
        self.n_fine_tune_layers = n_fine_tune_layers
        self.trainable = True
        self.output_size = 768
        self.pooling = pooling
        self.bert_path = bert_path
        if self.pooling not in ["first", "mean"]:
            raise NameError(
                f"Undefined pooling type (must be either first or mean, but is {
            )
```

```
        super(BertLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.bert = hub.Module(
            self.bert_path, trainable=self.trainable, name=f"{self.name}_module"
        )

        # Remove unused layers
        trainable_vars = self.bert.variables
        if self.pooling == "first":
            trainable_vars = [var for var in trainable_vars if not "/cls/" in va
            trainable_layers = ["pooler/dense"]

        elif self.pooling == "mean":
            trainable_vars = [
                var
                for var in trainable_vars
                if not "/cls/" in var.name and not "/pooler/" in var.name
            ]
            trainable_layers = []
        else:
            raise NameError(
                f"Undefined pooling type (must be either first or mean, but is {
            )

        # Select how many layers to fine tune
        for i in range(self.n_fine_tune_layers):
            trainable_layers.append(f"encoder/layer_{str(11 - i)}")

        # Update trainable vars to contain only the specified layers
        trainable_vars = [
            var
            for var in trainable_vars
            if any([l in var.name for l in trainable_layers])
        ]

        # Add to trainable weights
        for var in trainable_vars:
            self._trainable_weights.append(var)

        for var in self.bert.variables:
            if var not in self._trainable_weights:
                self._non_trainable_weights.append(var)

        super(BertLayer, self).build(input_shape)

    def call(self, inputs):
        inputs = [K.cast(x, dtype="int32") for x in inputs]
        input_ids, input_mask, segment_ids = inputs
        bert_inputs = dict(
            input_ids=input_ids, input_mask=input_mask, segment_ids=segment_ids
        )
        if self.pooling == "first":
            pooled = self.bert(inputs=bert_inputs, signature="tokens", as_dict=T
                "pooled_output"
            ]
        elif self.pooling == "mean":
            result = self.bert(inputs=bert_inputs, signature="tokens", as_dict=T
                "sequence_output"
            ]
```

```
            mul_mask = lambda x, m: x * tf.expand_dims(m, axis=-1)
            masked_reduce_mean = lambda x, m: tf.reduce_sum(mul_mask(x, m), axis
                    tf.reduce_sum(m, axis=1, keepdims=True) + 1e-10)
            input_mask = tf.cast(input_mask, tf.float32)
            pooled = masked_reduce_mean(result, input_mask)
        else:
            raise NameError(f"Undefined pooling type (must be either first or me

        return pooled

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_size)
```

We now use the custom TF hub BERT embedding layer within a higher-level function to define the overall model. More specifically, we put a dense trainable layer of output dimension 256 on top of the BERT embedding.

In [34]:
```
# Function to build overall model
def build_model(max_seq_length):
    in_id = tf.keras.layers.Input(shape=(max_seq_length,), name="input_ids")
    in_mask = tf.keras.layers.Input(shape=(max_seq_length,), name="input_masks")
    in_segment = tf.keras.layers.Input(shape=(max_seq_length,), name="segment_id
    bert_inputs = [in_id, in_mask, in_segment]

    # just extract BERT features, don't fine-tune
    bert_output = BertLayer(n_fine_tune_layers=0)(bert_inputs)
    # train dense classification layer on top of extracted features
    dense = tf.keras.layers.Dense(256, activation="relu")(bert_output)
    pred = tf.keras.layers.Dense(1, activation="sigmoid")(dense)

    model = tf.keras.models.Model(inputs=bert_inputs, outputs=pred)
    model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accura
    model.summary()

    return model

# Function to initialize variables correctly
def initialize_vars(sess):
    sess.run(tf.local_variables_initializer())
    sess.run(tf.global_variables_initializer())
    sess.run(tf.tables_initializer())
    K.set_session(sess)
```

In [35]:
```
# tf hub bert model path
bert_path = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"

# Instantiate tokenizer
tokenizer = create_tokenizer_from_hub_module(bert_path)

# Convert data to InputExample format
train_examples = convert_text_to_examples(train_x, train_y)
test_examples = convert_text_to_examples(test_x, test_y)

# Convert to features
(train_input_ids,train_input_masks,train_segment_ids,train_labels) = \
convert_examples_to_features(tokenizer, train_examples, max_seq_length=maxtokens
(test_input_ids,test_input_masks,test_segment_ids,test_labels) = \
```

```
convert_examples_to_features(tokenizer, test_examples, max_seq_length=maxtokens)

# Build model
model = build_model(maxtokens)

# Instantiate variables
initialize_vars(sess)

# Train model
history = model.fit([train_input_ids, train_input_masks, train_segment_ids],trai
                    validation_data=([test_input_ids, test_input_masks, test_seg
                    epochs=5,batch_size=32)
```

```
Converting examples to features: 100%|██████████| 1400/1400 [00:02<00:00, 513.99i
t/s]
Converting examples to features: 100%|██████████| 600/600 [00:01<00:00, 512.07it/
s]
```

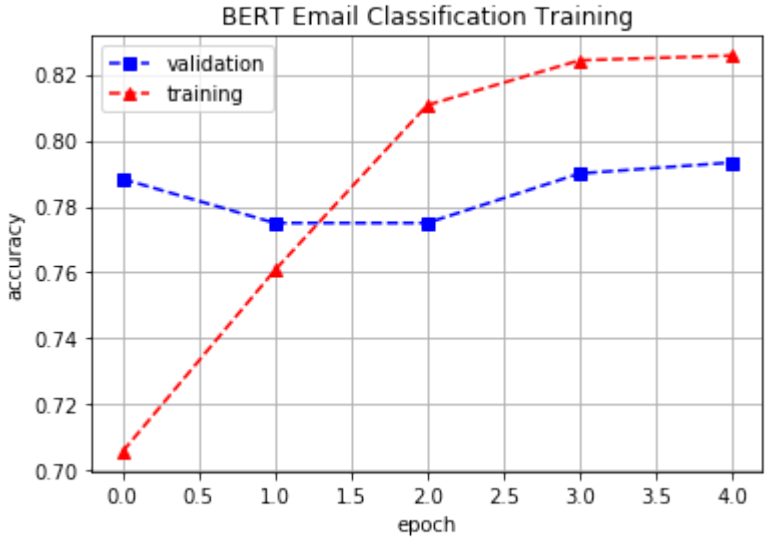
```
WARNING: Entity <bound method BertLayer.call of <__main__.BertLayer object at 0x7
8cfd58989b0>> could not be transformed and will be executed as-is. Please report
this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on Lin
ux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: convertin
g <bound method BertLayer.call of <__main__.BertLayer object at 0x78cfd58989b0>>:
AttributeError: module 'gast' has no attribute 'Num'
Model: "model_1"
_________________________________________________________________________________
_________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================
=================
input_ids (InputLayer)       [(None, 230)]             0
_________________________________________________________________________________
_________________
input_masks (InputLayer)     [(None, 230)]             0
_________________________________________________________________________________
_________________
segment_ids (InputLayer)     [(None, 230)]             0
_________________________________________________________________________________
_________________
bert_layer_1 (BertLayer)     (None, 768)               110104890   input_ids[0][0]
                                                                   input_masks[0]
[0]
                                                                   segment_ids[0]
[0]
_________________________________________________________________________________
_________________
dense_2 (Dense)              (None, 256)               196864      bert_layer_1[0]
[0]
_________________________________________________________________________________
_________________
dense_3 (Dense)              (None, 1)                 257         dense_2[0][0]
=================================================================================
=================
Total params: 110,302,011
Trainable params: 197,121
Non-trainable params: 110,104,890
_________________________________________________________________________________
_________________
Train on 1400 samples, validate on 600 samples
Epoch 1/5
1400/1400 [==============================] - 44s 31ms/sample - loss: 0.5673 - ac
c: 0.7057 - val_loss: 0.4951 - val_acc: 0.7883
Epoch 2/5
1400/1400 [==============================] - 39s 28ms/sample - loss: 0.4730 - ac
c: 0.7607 - val_loss: 0.4809 - val_acc: 0.7750
Epoch 3/5
1400/1400 [==============================] - 40s 28ms/sample - loss: 0.4132 - ac
c: 0.8107 - val_loss: 0.4723 - val_acc: 0.7750
Epoch 4/5
1400/1400 [==============================] - 39s 28ms/sample - loss: 0.3966 - ac
c: 0.8243 - val_loss: 0.4556 - val_acc: 0.7900
Epoch 5/5
1400/1400 [==============================] - 39s 28ms/sample - loss: 0.3848 - ac
c: 0.8257 - val_loss: 0.4507 - val_acc: 0.7933
```

Visualize Convergence

In [39]:
```
import matplotlib.pyplot as plt

df_history = pd.DataFrame(history.history)
fig,ax = plt.subplots()
plt.plot(range(df_history.shape[0]),df_history['val_acc'],'bs--',label='validati
plt.plot(range(df_history.shape[0]),df_history['acc'],'r^--',label='training')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('BERT Email Classification Training')
plt.legend(loc='best')
plt.grid()
plt.show()

fig.savefig('BERTConvergence.eps', format='eps')
fig.savefig('BERTConvergence.pdf', format='pdf')
fig.savefig('BERTConvergence.png', format='png')
fig.savefig('BERTConvergence.svg', format='svg')
```



Make figures downloadable to local system in interactive mode

In [40]:
```
from IPython.display import HTML
def create_download_link(title = "Download file", filename = "data.csv"):
    html = '<a href={filename}>{title}</a>'
    html = html.format(title=title,filename=filename)
    return HTML(html)

create_download_link(filename='BERTConvergence.svg')
```

Out[40]: Download file

In [38]:
```
!ls
!rm -rf aclImdb
!rm aclImdb_v1.tar.gz
```

```
BERTConvergence.eps  BERTConvergence.svg  kaggle_image_requirements.txt
BERTConvergence.pdf  aclImdb
BERTConvergence.png  aclImdb_v1.tar.gz
```

Pipeline para con BERT en TensorFlow 1.14s.

1. Configuración Preliminar

El primer paso consiste en configurar el entorno instalando las dependencias necesarias. Debido a que el proyecto utiliza TensorFlow 1.14 para compatibilidad con BERT y ELMo, se requiere instalar versiones específicas de librerías

- `keras=2..4` y `bert-tensorflow=1.0.1`.

Además, se crea un archivo con las dependencias insaladas (`kaggle_image_requireents.txt`) para asegurar reproducibilidad del entorno. Se importan las librerías clve como `tnsrflow` `anday,` nmpy `, y` tensoflow_hub`, y se inicializa una sesión de Tensorow.

2. Procesamiento de Texto

El procesamiento de texto es crítico para preparar los datos antes de utilizarlos con el modelo BERT. Se realizan las siguientes etapas:

1. **Tokenización**: Se divide el texto en palabras individuales, asegurando que el número de tokens por documento no exceda un límite predefinido.
2. **Eliminación de puntuación**: Se remueven caracteres no alfanuméricos y se truncan las palabras más largas.
3. **Remoción de palabras vacías**: Se eliminan palabras comunes que no aportan valor semántico (como "the", "are").

El conjunto de datos IMDB se descarga directamente desde su fuente, y se preprocesa para tokenizar el texto, eliminar puntuaciones y filtrar palabras vacías. Posteriormente, se realiza un submuestreo para obtener una cantidad equilibrada de muestras de correos electrónicos positivoy negativos.

3. Conversión de Datos a Formato BERT

Para utilizar los datos con BERT, se convierten a un format específico lamado `InputExample`. Este formato define las características necesarias para cada muestra, incluyendo el texto tokenizado y las etiquetas asociadas.

Los textos y etiquetas se convierten e matricesdeentrada (`ipu_ids` , `inpt_masks` , `segment_ids`) utilizando un tokenizador basado en un modelo BERT preentrenado. Esto asegura que los datos cumplan con los requisitos de trada del modelo.

4. Definición y Entrenamiento del Modelo

El modelo se define utilizando una capa personalizada de BERT, que extrae características de las secuencias de texto y las pasa a una capa densa entrenable para la clasificación. Las principales características del modelo incluyen:

- Una capa de salida densa de dimensión 256 con activación ReLU.
- Una capa final con activación sigmoidal para predecir la probabilidad de clasificación.

Las variables del modelo se inicializan adecuadamente antes de comenzar el entrenamiento. El modelo se entrena durante cinco épocas utilizando un 70% de los datos para entrenamienty un 30% para validación.

5. Visualización del Entrenamiento

Se visualiza la convergencia del modelo mediante una gráfica que muestra la precisión en los datos de entrenamiento y validación a través de las épocas. La gráficase gara enmútipls formatos (`.png` , `.pdf` , `.svg`) y se hacdescargable para el usuario.

6. Limpieza del Entorno

Finalmente, se eliminan los datos temporales descargados durante el procesamiento (como los archivos del conjunto IMDB) para manter en un entorno controlado como Kaggle.

In [ ]: