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 [1]:
```
!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/aa32dad071ce52b5
502266b5c659451cfd6ffcbf14e6c8c4f16c0ff5aaab/Keras-2.2.4-py2.py3-none-any.wh
l (312kB)
     |████████████████████████████████| 317kB 8.1MB/s eta 0:00:01
Requirement already satisfied: keras-preprocessing>=1.0.5 in /opt/conda/lib/
python3.6/site-packages (from keras==2.2.4) (1.1.0)
Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.6/site-p
ackages (from keras==2.2.4) (1.12.0)
Requirement already satisfied: h5py in /opt/conda/lib/python3.6/site-package
s (from keras==2.2.4) (2.9.0)
Requirement already satisfied: keras-applications>=1.0.6 in /opt/conda/lib/p
ython3.6/site-packages (from keras==2.2.4) (1.0.8)
Requirement already satisfied: numpy>=1.9.1 in /opt/conda/lib/python3.6/site
-packages (from keras==2.2.4) (1.16.4)
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.6/site-packa
ges (from keras==2.2.4) (5.1.2)
Requirement already satisfied: scipy>=0.14 in /opt/conda/lib/python3.6/site-
packages (from keras==2.2.4) (1.2.1)
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 [2]:
```
!pip freeze > kaggle_image_requirements.txt
```

In [3]:
```
# 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 [4]:
```
# 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

In [5]:
```
# Params for bert model and tokenization
Nsamp = 1000 # number of samples to generate in each class - 'spam', 'not sp
maxtokens = 230 # the maximum number of tokens per document
maxtokenlen = 200 # the maximum length of each token
```

Tokenization

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

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 [7]:
```
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 [8]:
```
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
# of a sentence - but we didn't find it to make a difference for this proble
# stopwords.remove("no")
# stopwords.remove("nor")
# stopwords.remove("not")
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [9]:
```
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 [10]:
```
!wget -q "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
!tar xzf aclImdb_v1.tar.gz
```

Shuffle and preprocess data

In [11]:
```
# 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 [12]:
```
# Subsample required number of samples
random_indices = np.random.choice(range(len(raw_header)),size=(Nsamp*2,),rep
data_train = raw_data[random_indices]
header = raw_header[random_indices]

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

```
DEBUG::data_train::
[list(['really', 'amazing', 'pile', 'pap', 'br', 'br', 'a', 'predictable',
'slow', 'moving', 'soul', 'destroying', 'mind', 'numbing', 'movie', 'which',
'slitting', 'wrists', 'rusty', 'bread', 'knife', 'seems', 'well', 'almost',
'necessarybr', 'br', 'the', 'acting', 'done', 'thin', 'dialogue', 'every',
'scene', 'least', 'twice', 'long', 'needs', 'be', 'intricate', 'details', 'c
areer', 'collapsing', 'career', 'rising', 'far', 'dreary', 'mundane', 'word
s', 'the', 'music', 'would', 'good', 'sit', 'movie', 'really', 'three', 'goo
d', 'songs', 'enough', 'reward', 'effort', 'required', 'watch', 'moviebr',
'br', 'watching', 'film', 'i', 'prayed', 'god', 'narcolepsy', 'someone', 'sh
oot', 'mebr', 'br', 'never', 'ever', 'ever', 'again'])
 list(['after', 'reading', 'book', 'i', 'loved', 'story', 'watching', 'movi
e', 'i', 'disappointed', 'many', 'changes', 'made', 'it', 'understandable',
'books', 'movies', 'differ', 'two', 'different', 'stories', 'names', 'book
s', 'story', 'remained', 'read', 'book', 'better', 'understanding', 'movie',
'the', 'book', 'gives', 'better', 'development', 'characters', 'these', 'cha
racters', 'extremely', 'interesting', 'make', 'care', 'them', 'the', 'locati
ons', 'indeed', 'line', 'books', 'descriptions', 'some', 'characters', 'incl
uded', 'television', 'microwaved', 'many', 'great', 'books', 'stories', 'per
fect', 'example', 'that', 'input', 'author', 'always', 'insure', 'good', 'mo
vie', 'help', 'sometimes'])
 list(['i', 'personally', 'watched', 'see', 'footage', 's', 's', 'it', 'fasc
inating', 'learn', 'drug', 'movement', 'essentially', 'started', 'became',
'pop', 'culture', 'eventual', 'uncompromising', 'force', 'life', 'the', 'int
erviews', 'classic', 'rock', 'stars', 'titillating', 'humorous', 'you', 'fee
l', 'like', 'secret', 'nodding', 'head', 'timebecause', 'feels', 'good', 'fa
miliar', 'i', 'loved', 'it', 'segments', 'spresent', 'day', 'i', 'highly',
'recommend', 'aspects', 'including', 'rock', 'music', 'hipper', 'movement',
'politics', 'good', 'ol', 'history', 'i', 'check', 'marked', 'box', 'sayin
g', 'contains', 'spoiler', 'i', 'idea', 'might', 'consider', 'spoiler', 'reg
ards', 'since', 'i', 'discussed', 'whats', '', 'segments', 'wanted', 'saf
e'])
 ...
 list(['ive', 'seen', 'lot', 'tv', 'movies', 'time', 'student', 'majority',
'normal', 'waste', 'time', 'us', 'television', 'throws', 'out', 'this', 'on
e', 'however', 'well', 'crafted', 'plotted', 'nice', 'twist', 'end', 'havin
g', 'seen', 'richard', 'dean', 'anderson', 'macgyver', 'stargate', 'i', 'sur
prised', 'excellent', 'performance', 'rather', 'rather', 'gamut', 'expressio
ns', 'ab', 'normally', 'gives', 'it', 'pleasant', 'surprise', 'see', 'daphn
e', 'zuniga', 'quite', 'long', 'time', 'dating', 'back', 'the', 'fly', 'ii',
'also', 'nice', 'see', 'robert', 'guillaumme', 'leading', 'role', 'again',
'i', 'cant', 'say', 'i', 'ever', 'take', 'jane', 'leeves', 'seriously', 'ben
ny', 'hill', 'days', 'managed', 'cope', 'well', 'role', 'all', 'highly', 're
commended', 'film'])
 list(['when', 'the', 'magic', 'of', 'lassie', 'opened', 'radio', 'city', 'm
usic', 'hall', 'i', 'foolish', 'enough', 'believe', 'would', 'heartwarming',
'first', 'lassie', 'films', 'were', 'notbr', 'br', 'the', 'story', 'abysma
l', 'songs', 'sherman', 'brothers', 'way', 'usual', 'level', 'characters',
'uninspired', 'james', 'stewart', 'mickey', 'rooney', 'seen', 'much', 'bette
r', 'daysbr', 'br', 'then', 'too', 'i', 'interested', 'seeing', 'alice', 'fa
yes', 'contribution', 'would', 'like', 'since', 'shed', 'absent', 'screen',
'many', 'years', 'always', 'fetching', 'earlier', 'roles', 'fox', 'alice',
'too', 'letdown', 'foolish', 'script', 'unflattering', 'photography', 'anoth
er', 'disappointmentbr', 'br', 'nothing', 'original', 'here', 'nothing', 'ev
en', 'remotely', 'interesting', 'adult', 'enjoyand', 'clearly', 'magic', 'pr
esent', 'anyone', 'you', 'skip', 'one', 'without', 'missing', 'thing'])
```

```
 list(['great', 'movie', '', 'especially', 'music', '', 'etta', 'james', '',
'at', 'last', 'this', 'speaks', 'volumes', 'finally', 'found', 'special', 's
omeone'])]
```

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

In [13]:
```
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]
[1003  997]
```

In [14]:
```
# function for converting data into the right format, due to the difference
# we expect a single string per email here, versus a list of tokens for the
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
[['a series shorts spoofing dumb tv shows groove tube hits misses lot overal
l i really like movie unfortunately couple segments totally boring a really
great clips make this a predecessor classics like kentucky fried movie']
 ['damon runyons world times square new york prior disneyfication basis musi
cal joseph l mankiewicz man knew movies directed nostalgic tribute crossroad
s world show us underside new york past frank loessers music sounds great we
watch magnificent cast characters typical area people edges society tended g
ravitate toward area lights action possibilities part town this underbelly c
ity made living street life intensebr br some songs original production incl
uded film we know whether makes sense unusual hollywood musical change alter
worked stage that original cast included wonderful vivian blaine stubby kaye
wonder decision letting robert alda sam levene isabel bigley repeat original
roles these distinguished actors could made amazing contributionbr br the fi
lm visually amazing the look follows closely fashions times as far casting m
arlon brando otherwise known singing abilities frank sinatra jean simmons se
em work']
 ['this gorgeous movie visually the images mexican desert old mansion charac
ters picturesque costumesall amount real work artbr br the story seems bit l
oose thats meant realistic it taken book called one hundred years solitude s
upposed evocation isolated otherworldly atmosphere latin america so far god
close united states the tremendous debt erendira owes grandmother symbolic l
atin americas international debt burden although many layers meaningbr br if
appreciate slowmoving richlytextured movie one you']
 ...
 ['michael kallio gives strong convincing performance eric seaver troubled y
oung man horribly mistreated little boy monstrous abusive alcoholic stepfath
er barry a genuinely frightening portrayal gunnar hansen eric compassionate
fiancé sweetly played lovely tracee newberry job transcribing autopsy report
s local morgue haunted bleak past egged bald beaming jack demon a truly cree
py michael robert brandon sent edge recent death mother eric goes deep end e
mbarks brutal killing spree capably directed kallio who also wrote tight ast
ute script uniformly fine acting sound noname cast jeff steiger especially g
ood erics wannabe helpful guardian angel michael rather rough overall polish
ed cinematography george lieber believable truetolife characters jolting out
bursts raw shocking unflinchingly ferocious violence moody spooky score dan
kolton uncompromisingly downbeat ending grungy detroit michigan locations gr
imly serious tone taut gripping narrative stays steady track throughout extr
emely potent gritty psychological horror thriller makes often absorbing dist
urbing viewing a real sleeper']
 ['genie zoe trilling arrives egypt visit hypocritical biblequoting archeolo
gist father william finley attracts attention group cultists led descendant
marquis de sade robert englund englund also plays de sade flashbacks ranting
cell genie led astray mohammed juliano merr rides around naked horse sabina
alona kamhi bisexual introduces opium smoking leads wild hallucination featu
ring topless harem dancers woman simulating oral sex snake orgy father preac
hing background meanwhile black hooded cult members decapitate gouge eyeball
s slit throats when genie slipped drugs tea imagines de sade hanging cross g
oldpainted woman leafy gstring bloody bed covered snakes its reincarnation d
e sades lost lovebr br this typically sleazy harry alan towers production re
dundant seedy pretty senseless sets costumes cinematography location work ex
cellent least theres always something going onbr br score  ']
 ['its spelled slashers i happy main character flashed boobs that pretty tig
ht before movie pretty much blows the acting like elist shown well movie not
mention low budget preacherman chainsaw charlie played person the whole movi
```

```
e looks like shot camcorder instead half way decent film the reason i liked
movie chainsaw charlie doctor ripper funny they said many stupid things made
laugh other see movie blockbuster everyone favor hide behind lawnmowerman  a
nybody thinks movie good mentally evaluated']]
[1 1 1 1 1]
(1400,)
```

Build, Train and Evaluate BERT Model

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

In [15]:
```
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=T
    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, labe
        )
    return InputExamples
```

Next, we define a custom tf hub BERT layer

In [16]:
```
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
            )

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

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

        # Remove unused layers
        trainable_vars = self.bert.variables
        if self.pooling == "first":
            trainable_vars = [var for var in trainable_vars if not "/cls/" i
            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
            )

        # 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_
        )
        if self.pooling == "first":
            pooled = self.bert(inputs=bert_inputs, signature="tokens", as_di
                "pooled_output"
            ]
        elif self.pooling == "mean":
            result = self.bert(inputs=bert_inputs, signature="tokens", as_di
                "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),
                    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 c

        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 [17]:
```
# 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_mas
    in_segment = tf.keras.layers.Input(shape=(max_seq_length,), name="segmer
    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=["ac
    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 [18]:

```
# 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=maxtc
(test_input_ids,test_input_masks,test_segment_ids,test_labels) = \
convert_examples_to_features(tokenizer, test_examples, max_seq_length=maxtok

# 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],
                    validation_data=([test_input_ids, test_input_masks, test
                    epochs=5,batch_size=32)
```

```
Converting examples to features: 100%|██████████| 1400/1400 [00:02<00:00, 49
1.41it/s]
Converting examples to features: 100%|██████████| 600/600 [00:01<00:00, 504.
37it/s]
```

```
WARNING: Entity <bound method BertLayer.call of <__main__.BertLayer object a
t 0x7bf1e15645f8>> could not be transformed and will be executed as-is. Plea
se report this to the AutgoGraph team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full outp
ut. Cause: converting <bound method BertLayer.call of <__main__.BertLayer ob
ject at 0x7bf1e15645f8>>: AttributeError: module 'gast' has no attribute 'Nu
m'
Model: "model"
_________________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #     Connected t
o
=========================================================================
=====================
input_ids (InputLayer)       [(None, 230)]             0
_________________________________________________________________________
_____________________
input_masks (InputLayer)     [(None, 230)]             0
_________________________________________________________________________
_____________________
segment_ids (InputLayer)     [(None, 230)]             0
_________________________________________________________________________
_____________________
bert_layer (BertLayer)       (None, 768)               110104890   input_ids
[0][0]
                                                                   input_masks
[0][0]
                                                                   segment_ids
[0][0]
_________________________________________________________________________
_____________________
dense (Dense)                (None, 256)               196864      bert_layer
[0][0]
_________________________________________________________________________
_____________________
dense_1 (Dense)              (None, 1)                 257         dense[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 [==============================] - 20s 14ms/sample - loss: 0.5573
- acc: 0.7014 - val_loss: 0.4451 - val_acc: 0.8133
Epoch 2/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.4715
- acc: 0.7736 - val_loss: 0.5096 - val_acc: 0.7367
Epoch 3/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.4406
- acc: 0.7950 - val_loss: 0.4016 - val_acc: 0.8150
Epoch 4/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.4205
- acc: 0.8107 - val_loss: 0.4130 - val_acc: 0.8150
```

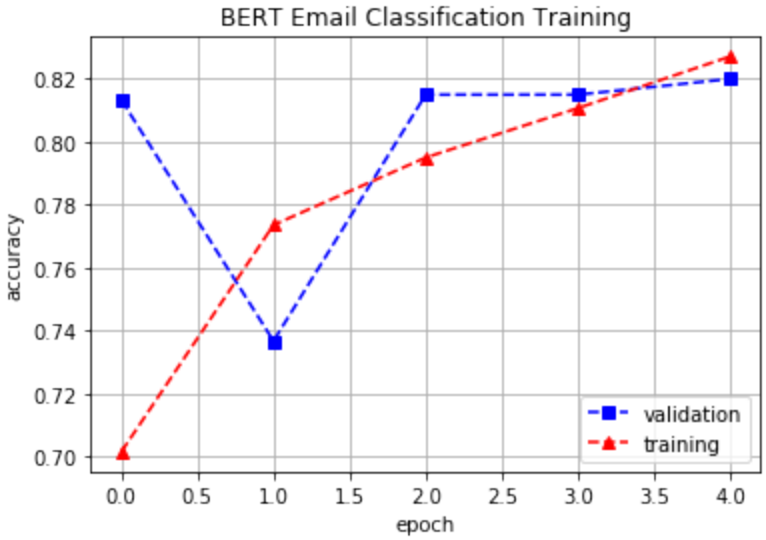
```
Epoch 5/5
1400/1400 [==============================] - 18s 13ms/sample - loss: 0.3901
- acc: 0.8271 - val_loss: 0.3938 - val_acc: 0.8200
```

Visualize Convergence

In [19]:
```
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='vali
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 [20]:
```
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[20]: Download file

In [21]:
```
!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
```