Yu chien (Calvin) Ma

In this project, we will use TensorFlow and TF-Hub.

The pretrained BERT model used in this project is available on TensorFlow Hub

Setting up TensorFlow and Colab Runtime.

```
# Checking GPU availability
!nvidia-smi

Sun Dec  4 05:57:50 2022
+---------------------------------------------------------------------
--------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version:
11.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|                               |                      |
MIG M. |
|
===============================+======================+===============
=======|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |
0 |
| N/A   56C    P0    30W /  70W |      0MiB / 15109MiB |      0%
Default |
|                               |                      |
N/A |
+-------------------------------+----------------------
+----------------------+


+---------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU
Memory |
|        ID   ID
Usage      |
|
======================================================================
=======|
|  No running processes found
```

```
|
+---------------------------------------------------------------------
--------+
```

Installing TensorFlow and TensorFlow Model Garden

```
import tensorflow as tf
print(tf.version.VERSION)
```

```
2.9.2
```

```
!git clone --depth 1 -b v2.3.0
https://github.com/tensorflow/models.git
```

```
Cloning into 'models'...
remote: Enumerating objects: 2650, done.ote: Counting objects: 100%
(2650/2650), done.ote: Compressing objects: 100% (2311/2311),
done.ote: Total 2650 (delta 505), reused 1389 (delta 306), pack-reused
0ake experimental
changes and commit them, and you can discard any commits you make in
this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you
may
do so (now or later) by using -b with the checkout command again.
Example:

  git checkout -b <new-branch-name>
```

```
# installing requirements to use tensorflow/models repository
!pip install -Uqr models/official/requirements.txt
```

```
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
ipython 7.9.0 requires jedi>=0.10, which is not installed.
```

Restart the runtime

Download and Import the Quora Insincere Questions Dataset

```
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub
import sys
sys.path.append('models')
```

```
from official.nlp.data import classifier_data_lib
from official.nlp.bert import tokenization
from official.nlp import optimization

print("TF Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if
tf.config.experimental.list_physical_devices("GPU") else "NOT
AVAILABLE")
```

```
TF Version:  2.9.2
Eager mode:  True
Hub version:  0.12.0
GPU is available
```

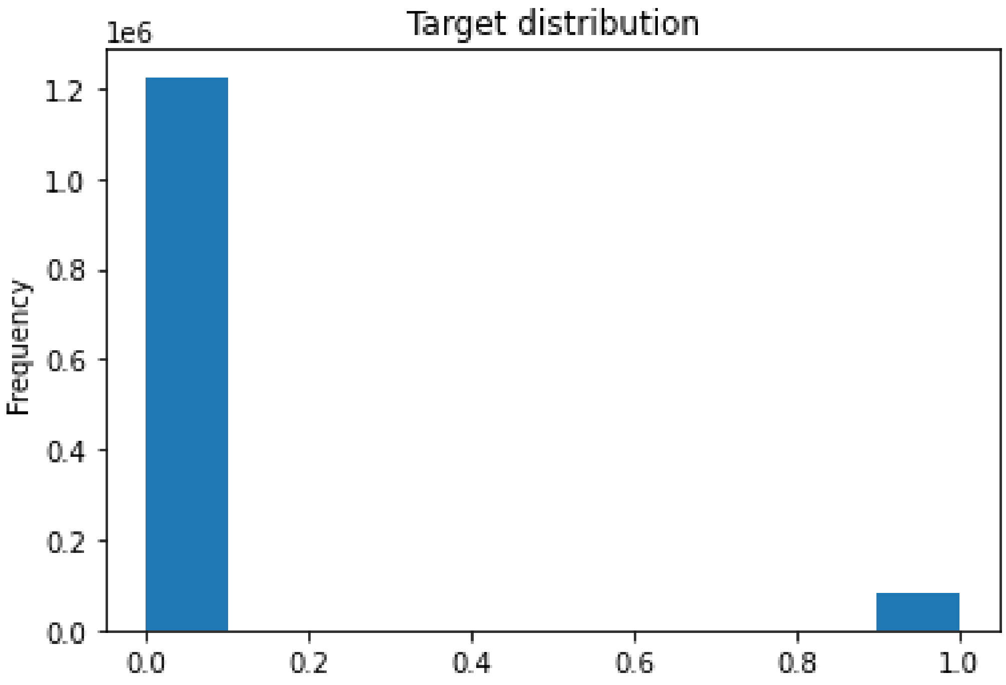
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('https://archive.org/download/fine-tune-bert-
tensorflow-train.csv/train.csv.zip',
                 compression='zip', low_memory=False)
df.shape
```

```
(1306122, 3)
```

```
df.tail(20)
```

```
                          qid  \
1306102  ffff3778790af9baae76
1306103  ffff3f0a2449ffe4b9ff
1306104  ffff41393389d4206066
1306105  ffff42493fc203cd9532
1306106  ffff48dd47bee89fff79
1306107  ffff5fd051a032f32a39
1306108  ffff6d528040d3888b93
1306109  ffff8776cd30cdc8d7f8
1306110  ffff94d427ade3716cd1
1306111  ffffa382c58368071dc9
1306112  ffffa5b0fa76431c063f
1306113  ffffae5dbda3dc9e9771
1306114  ffffba7c4888798571c1
1306115  ffffc0c7158658a06fd9
1306116  ffffc404da586ac5a08f
1306117  ffffcc4e2331aaf1e41e
1306118  ffffd431801e5a2f4861
1306119  ffffd48fb36b63db010c
1306120  ffffec519fa37cf60c78
1306121  ffffed09fedb5088744a
```

```
                                             question_text  target
1306102  What steps can I take to live a normal life if...       0
1306103  Isn't Trump right after all? Why should the US...       1
1306104  Is 33 too late for a career in creative advert...       0
1306105  What is difference between the filteration wor...       0
1306106  If the universe "popped" into existence from n...       0
1306107  How does a shared service technology team meas...       0
1306108                    How is DSATM civil engineering?       0
1306109  Do you know any problem that depends solely on...       0
1306110  What are some comic ideas for you Tube videos ...       0
1306111  If you had $10 million of Bitcoin, could you s...       0
1306112                Are you ashamed of being an Indian?       1
1306113  What are the methods to determine fossil ages ...       0
1306114                          What is your story today?       0
1306115  How do I consume 150 gms protein daily both ve...       0
1306116  What are the good career options for a msc che...       0
1306117  What other technical skills do you need as a c...       0
1306118  Does MS in ECE have good job prospects in USA ...       0
1306119                          Is foam insulation toxic?       0
1306120  How can one start a research project based on ...       0
1306121  Who wins in a battle between a Wolverine and a...       0

# Plotting the distribution of sincere vs insincere questions; as can
be seen, there is a lot of imbalance
df.target.plot(kind='hist', title='Target distribution');
```



Creating tf.data.Datasets for Training and Evaluation

```
train_df, remaining = train_test_split(df, random_state=42,
train_size=0.0075, stratify=df.target.values)
valid_df, _ = train_test_split(remaining, random_state=42,
train_size=0.00075, stratify=remaining.target.values)
train_df.shape, valid_df.shape
```

```
((9795, 3), (972, 3))
```

```
with tf.device('/cpu:0'):
  train_data =
tf.data.Dataset.from_tensor_slices((train_df.question_text.values,
train_df.target.values))
  valid_data =
tf.data.Dataset.from_tensor_slices((valid_df.question_text.values,
valid_df.target.values))

  for text, label in train_data.take(1):
    print(text)
    print(label)
```

```
tf.Tensor(b'Why are unhealthy relationships so desirable?', shape=(),
dtype=string)
tf.Tensor(0, shape=(), dtype=int64)
```

Task 5: Download a Pre-trained BERT Model from TensorFlow Hub

```
"""
Each line of the dataset is composed of the review text and its label
- Data preprocessing consists of transforming text to BERT input
features:
input_word_ids, input_mask, segment_ids
- In the process, tokenizing the text is done with the provided BERT
model tokenizer
"""

label_list = [0, 1] # Label categories
max_seq_length = 128 # maximum length of (token) input sequences
train_batch_size = 32

# Get BERT layer and tokenizer:

bert_layer =
hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-
768_A-12/2",
                            trainable=True)
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
```

```
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
tokenizer = tokenization.FullTokenizer(vocab_file, do_lower_case)

tokenizer.wordpiece_tokenizer.tokenize('hi, how are you doing?')

['hi', '##,', 'how', 'are', 'you', 'doing', '##?']

tokenizer.convert_tokens_to_ids(tokenizer.wordpiece_tokenizer.tokenize
('hi, how are you doing?'))

[7632, 29623, 2129, 2024, 2017, 2725, 29632]
```

Task 6: Tokenize and Preprocess Text for BERT

We'll need to transform our data into a format BERT understands. This involves two steps. First, we create InputExamples using `classifier_data_lib`'s constructor `InputExample` provided in the BERT library.

```
# This provides a function to convert row to input features and label

def to_feature(text, label, label_list=label_list,
max_seq_length=max_seq_length, tokenizer=tokenizer):
  example = classifier_data_lib.InputExample(guid = None,
                                             text_a = text.numpy(),
                                             text_b = None,
                                             label = label.numpy())
  feature = classifier_data_lib.convert_single_example(0, example,
label_list,
                                    max_seq_length, tokenizer)

  return (feature.input_ids, feature.input_mask, feature.segment_ids,
feature.label_id)
```

We want to use `Dataset.map` to apply this function to each element of the dataset. `Dataset.map` runs in graph mode.

- Graph tensors do not have a value.
- In graph mode we can only use TensorFlow Ops and functions.

Therefore, we cannot `.map` this function directly: we need to wrap it in a `tf.py_function`. The `tf.py_function` will pass regular tensors (with a value and a `.numpy()` method to access it), to the wrapped python function.

Wrap a Python Function into a TensorFlow Ops for Eager Execution

```
def to_feature_map(text, label):
  input_ids, input_mask, segment_ids, label_id =
tf.py_function(to_feature, inp=[text, label],
                                Tout=[tf.int32, tf.int32, tf.int32,
tf.int32])

  # py_func doesn't set the shape of the returned tensors.
  input_ids.set_shape([max_seq_length])
  input_mask.set_shape([max_seq_length])
  segment_ids.set_shape([max_seq_length])
  label_id.set_shape([])

  x = {
        'input_word_ids': input_ids,
        'input_mask': input_mask,
        'input_type_ids': segment_ids
    }
  return (x, label_id)
```

Create a TensorFlow Input Pipeline with `tf.data`

```
with tf.device('/cpu:0'):
  # train
  train_data = (train_data.map(to_feature_map,

num_parallel_calls=tf.data.experimental.AUTOTUNE)
                          #.cache()
                          .shuffle(1000)
                          .batch(32, drop_remainder=True)
                          .prefetch(tf.data.experimental.AUTOTUNE))

  # valid
  valid_data = (valid_data.map(to_feature_map,

num_parallel_calls=tf.data.experimental.AUTOTUNE)
                          .batch(32, drop_remainder=True)
                          .prefetch(tf.data.experimental.AUTOTUNE))
```

The resulting `tf.data.Datasets` return `(features, labels)` pairs, as expected by `keras.Model.fit`:

```
# data spec
train_data.element_spec
```

```
({'input_word_ids': TensorSpec(shape=(32, 128), dtype=tf.int32,
name=None),
  'input_mask': TensorSpec(shape=(32, 128), dtype=tf.int32,
name=None),
  'input_type_ids': TensorSpec(shape=(32, 128), dtype=tf.int32,
name=None)},
 TensorSpec(shape=(32,), dtype=tf.int32, name=None))
```

```
# data spec
valid_data.element_spec

({'input_word_ids': TensorSpec(shape=(32, 128), dtype=tf.int32,
name=None),
  'input_mask': TensorSpec(shape=(32, 128), dtype=tf.int32,
name=None),
  'input_type_ids': TensorSpec(shape=(32, 128), dtype=tf.int32,
name=None)},
 TensorSpec(shape=(32,), dtype=tf.int32, name=None))
```

Task 9: Add a Classification Head to the BERT Layer

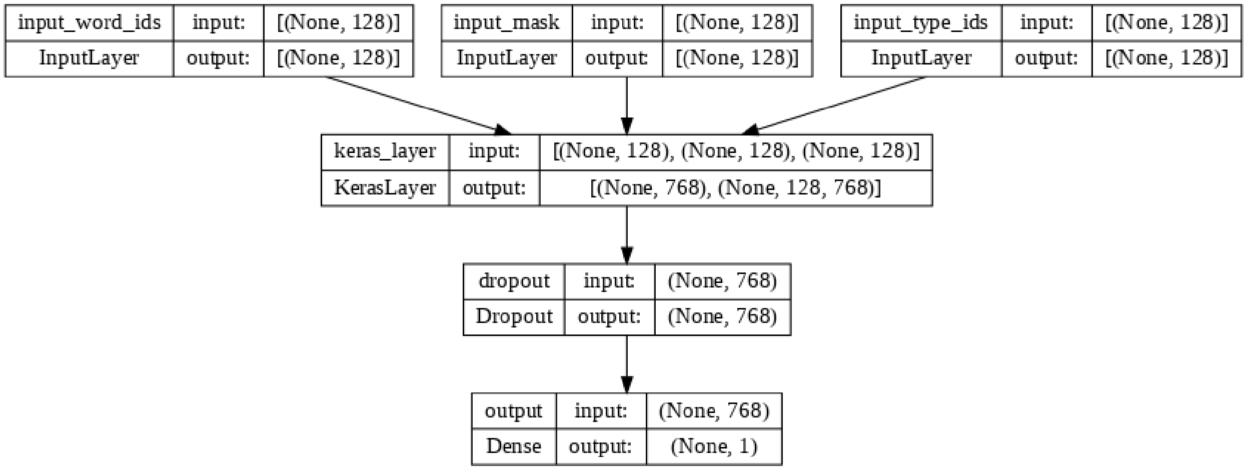
```
# Building the model
def create_model():
  input_word_ids = tf.keras.layers.Input(shape=(max_seq_length,),
dtype=tf.int32,
                                      name="input_word_ids")
  input_mask = tf.keras.layers.Input(shape=(max_seq_length,),
dtype=tf.int32,
                                  name="input_mask")
  input_type_ids = tf.keras.layers.Input(shape=(max_seq_length,),
dtype=tf.int32,
                                  name="input_type_ids")

  pooled_output, sequence_output = bert_layer([input_word_ids,
input_mask, input_type_ids])

  drop = tf.keras.layers.Dropout(0.4)(pooled_output)
  output = tf.keras.layers.Dense(1, activation="sigmoid",
name="output")(drop)

  model = tf.keras.Model(
    inputs={
        'input_word_ids': input_word_ids,
        'input_mask': input_mask,
        'input_type_ids': input_type_ids
    },
    outputs=output)
  return model
```

Task 10: Fine-Tune BERT for Text Classification

```
model = create_model()
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-5),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy()])
model.summary()
```

```
Model: "model"
______________________________________________________________________
____________________________
 Layer (type)                   Output Shape         Param #
Connected to
======================================================================
============================
 input_word_ids (InputLayer)    [(None, 128)]        0           []



 input_mask (InputLayer)        [(None, 128)]        0           []



 input_type_ids (InputLayer)    [(None, 128)]        0           []



 keras_layer (KerasLayer)       [(None, 768),        109482241
['input_word_ids[0][0]',
                                 (None, 128, 768)]
'input_mask[0][0]',

'input_type_ids[0][0]']


 dropout (Dropout)              (None, 768)          0
['keras_layer[0][0]']


 output (Dense)                 (None, 1)            769
['dropout[0][0]']


======================================================================
============================
Total params: 109,483,010
Trainable params: 109,483,009
Non-trainable params: 1
______________________________________________________________________
____________________________
```

```
tf.keras.utils.plot_model(model=model, show_shapes=True, dpi=76, )
```


input_word_ids | input: | [(None, 128)]
InputLayer | output: | [(None, 128)]
input_mask | input: | [(None, 128)]
InputLayer | output: | [(None, 128)]
input_type_ids | input: | [(None, 128)]
InputLayer | output: | [(None, 128)]
keras_layer | input: | [(None, 128), (None, 128), (None, 128)]
KerasLayer | output: | [(None, 768), (None, 128, 768)]
dropout | input: | (None, 768)
Dropout | output: | (None, 768)
output | input: | (None, 768)
Dense | output: | (None, 1)

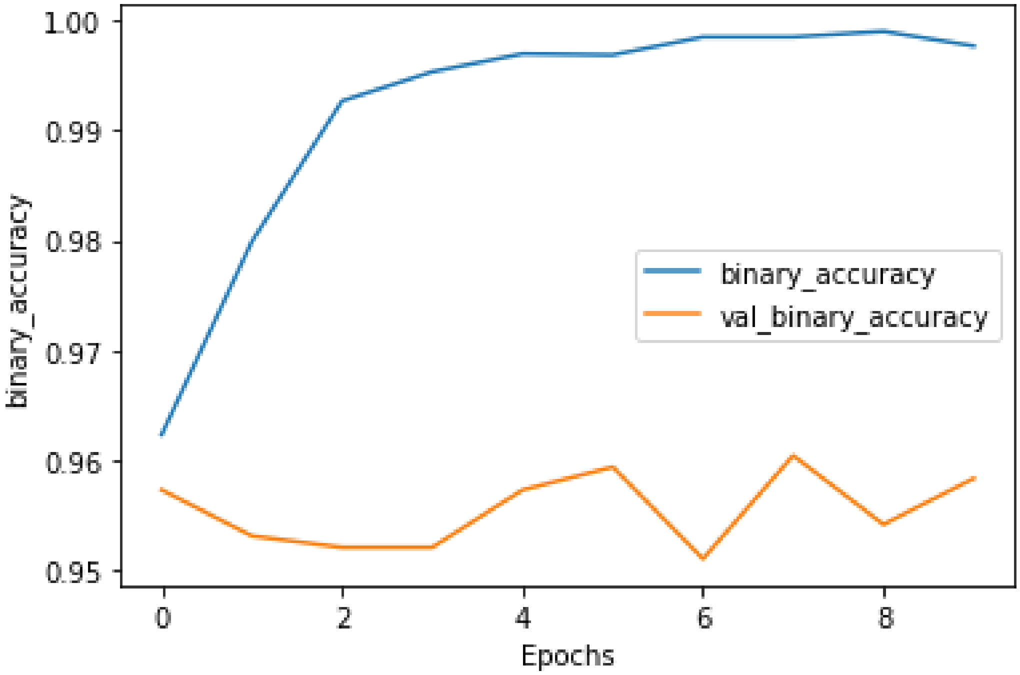
```
# Train model
epochs = 10
history = model.fit(train_data,
                    validation_data=valid_data,
                    epochs=epochs,
                    verbose=1)
```

```
Epoch 1/10
306/306 [==============================] - 269s 872ms/step - loss:
0.0972 - binary_accuracy: 0.9623 - val_loss: 0.1300 -
val_binary_accuracy: 0.9573
Epoch 2/10
306/306 [==============================] - 268s 873ms/step - loss:
0.0520 - binary_accuracy: 0.9799 - val_loss: 0.1676 -
val_binary_accuracy: 0.9531
Epoch 3/10
306/306 [==============================] - 268s 872ms/step - loss:
0.0228 - binary_accuracy: 0.9926 - val_loss: 0.2387 -
val_binary_accuracy: 0.9521
Epoch 4/10
306/306 [==============================] - 267s 870ms/step - loss:
0.0134 - binary_accuracy: 0.9953 - val_loss: 0.2012 -
val_binary_accuracy: 0.9521
Epoch 5/10
306/306 [==============================] - 268s 873ms/step - loss:
0.0082 - binary_accuracy: 0.9969 - val_loss: 0.2619 -
val_binary_accuracy: 0.9573
Epoch 6/10
306/306 [==============================] - 268s 874ms/step - loss:
0.0087 - binary_accuracy: 0.9968 - val_loss: 0.2250 -
val_binary_accuracy: 0.9594
Epoch 7/10
306/306 [==============================] - 267s 870ms/step - loss:
0.0039 - binary_accuracy: 0.9985 - val_loss: 0.2739 -
val_binary_accuracy: 0.9510
```

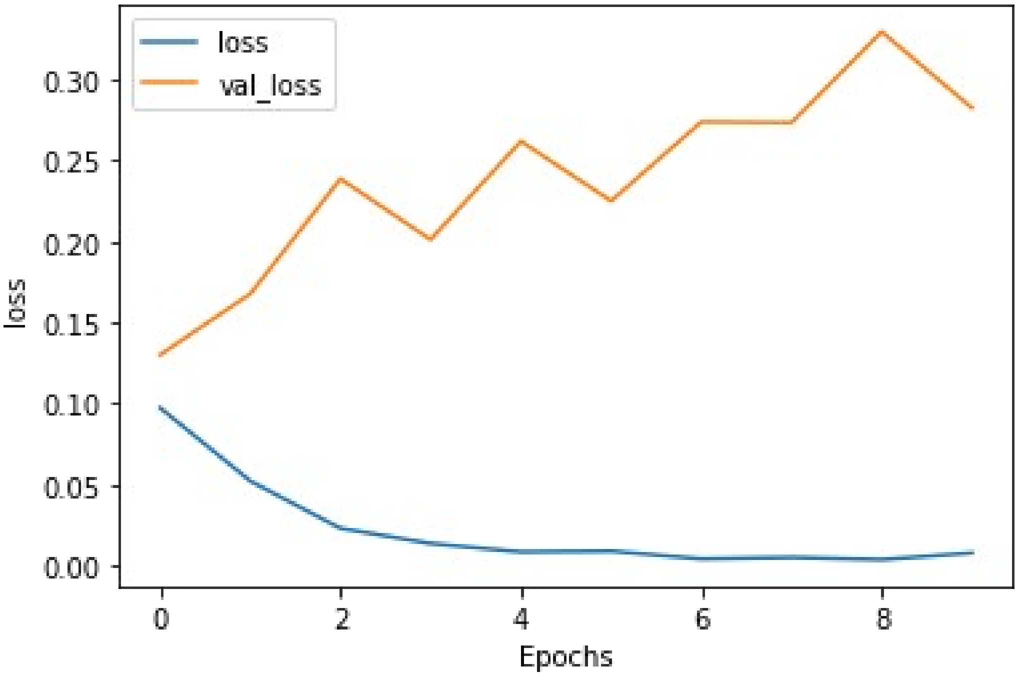
```
Epoch 8/10
306/306 [==============================] - 268s 873ms/step - loss:
0.0048 - binary_accuracy: 0.9985 - val_loss: 0.2735 -
val_binary_accuracy: 0.9604
Epoch 9/10
306/306 [==============================] - 268s 873ms/step - loss:
0.0034 - binary_accuracy: 0.9990 - val_loss: 0.3295 -
val_binary_accuracy: 0.9542
Epoch 10/10
306/306 [==============================] - 267s 868ms/step - loss:
0.0076 - binary_accuracy: 0.9977 - val_loss: 0.2826 -
val_binary_accuracy: 0.9583
```

Task 11: Evaluate the BERT Text Classification Model

```
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
  plt.show()

plot_graphs(history, 'binary_accuracy')
```



```
plot_graphs(history, 'loss')
```



```
model.evaluate(valid_data, verbose=1)
```

```
30/30 [==============================] - 9s 299ms/step - loss: 0.2826
- binary_accuracy: 0.9583
```

```
[0.2825920879840851, 0.9583333134651184]
```

```
sample_example = [" ",\
                  " ",\
                  " ",\
                  " ",\
                  " ",\
                  " "]
test_data = tf.data.Dataset.from_tensor_slices((sample_example,
[0]*len(sample_example)))
test_data = (test_data.map(to_feature_map).batch(1))
preds = model.predict(test_data)
#['Insincere' if pred >=0.5 else 'Sincere' for pred in preds]
```

```
6/6 [==============================] - 0s 11ms/step
```

```
preds
```

```
array([[0.00142263],
       [0.00142263],
       [0.00142263],
       [0.00142263],
       [0.00142263],
       [0.00142263]], dtype=float32)
```