In [ ]:

```
import json
import tensorflow as tf
import numpy as np
import urllib
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

def solution_model():
    url = 'http://storage.googleapis.com/download.tensorflow.org/data/sarcasm.json'
    urllib.request.urlretrieve(url, 'sarcasm.json')

    vocab_size = 1000
    embedding_dim = 16
    max_length = 120
    trunc_type='post'
    padding_type='post'
    oov_tok = "<OOV>"
    training_size = 20000

    sentences = []
    labels = []

    with open("sarcasm.json", 'r') as f:
        datastore = json.load(f)

    for item in datastore:
        sentences.append(item['headline'])
        labels.append(item['is_sarcastic'])

    training_sentences = sentences[0:training_size]
    training_labels = labels[0:training_size]
    testing_sentences = sentences[training_size:]  # validation set
    testing_labels = labels[training_size:]  # validation set

    tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
    tokenizer.fit_on_texts(training_sentences)

    training_sequences = tokenizer.texts_to_sequences(training_sentences)
    training_padded = pad_sequences(training_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
    testing_sequences = tokenizer.texts_to_sequences(testing_sentences)  # validation set
    testing_padded = pad_sequences(testing_sequences, maxlen=max_length, padding=padding_type,
                                   truncating=trunc_type)  # validation set

    training_padded = np.array(training_padded)
    training_labels = np.array(training_labels)
    testing_padded = np.array(testing_padded)  # validation set
    testing_labels = np.array(testing_labels)  # validation set

    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),

        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(96, return_sequences=True)),
        tf.keras.layers.Dropout(0.3),
```

```
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
        tf.keras.layers.Dropout(0.2),

        tf.keras.layers.Dense(96, activation='relu'),

        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accura
cy'])

    num_epochs = 15

    model.fit(training_padded, training_labels, epochs=num_epochs, validation_da
ta=(testing_padded, testing_labels), verbose=1)

    return model
```

In [ ]:

```
if __name__ == '__main__':
    model = solution_model()
    model.save("mymodel.h5")
```

In [1]:

```
import json
import tensorflow as tf
import numpy as np
import urllib
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

!pip install -q -U keras-tuner
import kerastuner as kt
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds
import IPython
```

```
     |████████████████████████████████| 61kB 1.9MB/s eta 0:00:011
  Building wheel for keras-tuner (setup.py) ... done
  Building wheel for terminaltables (setup.py) ... done
```

In [2]:

```
class ClearTrainingOutput(tf.keras.callbacks.Callback):
  def on_train_end(*args, **kwargs):
    IPython.display.clear_output(wait = True)
```

In [3]:

```
    url = 'http://storage.googleapis.com/download.tensorflow.org/data/sarcasm.json'
    urllib.request.urlretrieve(url, 'sarcasm.json')

    # DO NOT CHANGE THIS CODE OR THE TESTS MAY NOT WORK
    vocab_size = 1000
    embedding_dim = 16
    max_length = 120
    trunc_type='post'
    padding_type='post'
    oov_tok = "<OOV>"
    training_size = 20000

    sentences = []
    labels = []
    # YOUR CODE HERE
    with open("sarcasm.json", 'r') as f:
        datastore = json.load(f)

    for item in datastore:
        sentences.append(item['headline'])
        labels.append(item['is_sarcastic'])

    training_sentences = sentences[0:training_size]
    training_labels = labels[0:training_size]
    testing_sentences = sentences[training_size:]  # validation set
    testing_labels = labels[training_size:]  # validation set

    tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
    tokenizer.fit_on_texts(training_sentences)

    training_sequences = tokenizer.texts_to_sequences(training_sentences)
    training_padded = pad_sequences(training_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
    testing_sequences = tokenizer.texts_to_sequences(testing_sentences)  # validation set
    testing_padded = pad_sequences(testing_sequences, maxlen=max_length, padding=padding_type,
                                   truncating=trunc_type)  # validation set

    training_padded = np.array(training_padded)
    training_labels = np.array(training_labels)
    testing_padded = np.array(testing_padded)  # validation set
    testing_labels = np.array(testing_labels)  # validation set
```

In [9]:

```
def model_builder(hp):
  model = tf.keras.Sequential()

  model.add(tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=ma
x_length))

  model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(hp.Int('LSTM_1',
                                        min_value=32,
                                        max_value=128,
                                        step=16), return_sequences=True)))
  model.add(tf.keras.layers.Dropout(hp.Choice('Dropout_1', values = [0.2, 0.3,
0.5])))
  model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(hp.Int('LSTM_2',
                                        min_value=32,
                                        max_value=128,
                                        step=16))))
  model.add(tf.keras.layers.Dropout(hp.Choice('Dropout_2', values = [0.2, 0.3,
0.5])))

  model.add(tf.keras.layers.Dense(hp.Int('Dense_1',
                                        min_value=32,
                                        max_value=256,
                                        step=32), activation='relu'))

  model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

  #from tf.keras.optimizers import Adam
  model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(l
earning_rate = hp.Choice('learning_rate', values = [1e-3, 1e-4, 1e-5, 1e-6])), m
etrics=['accuracy'])

  return model
```

In [10]:

```
tuner = kt.Hyperband(model_builder,
                     objective = 'val_accuracy',
                     max_epochs = 15,
                     factor = 3,
                     directory = 'my_dir',
                     project_name = 'intro_to_kt')
```

In [11]:

```
tuner.search_space_summary()
```

Search space summary

|-Default search space size: 6

LSTM_1 (Int)

|-default: None

|-max_value: 128

|-min_value: 32

|-sampling: None

|-step: 16

Dropout_1 (Choice)

|-default: 0.2

|-ordered: True

|-values: [0.2, 0.3, 0.5]

LSTM_2 (Int)

|-default: None

|-max_value: 128

|-min_value: 32

|-sampling: None

|-step: 16

Dropout_2 (Choice)

|-default: 0.2

|-ordered: True

|-values: [0.2, 0.3, 0.5]

Dense_1 (Int)

|-default: None

|-max_value: 256

|-min_value: 32

|-sampling: None

|-step: 32

learning_rate (Choice)

|-default: 0.001

|-ordered: True

|-values: [0.001, 0.0001, 1e-05, 1e-06]

In [12]:

```
tuner.search(training_padded, training_labels, epochs = 15, validation_data = (t
esting_padded, testing_labels), callbacks = [ClearTrainingOutput()])
```

Trial complete

Trial summary

|-Trial ID: 9970df4828443905d5a28a1ad5dfb2bf

|-Score: 0.6574750542640686

|-Best step: 0

Hyperparameters:

|-Dense_1: 128

|-Dropout_1: 0.5

|-Dropout_2: 0.5

|-LSTM_1: 128

|-LSTM_2: 96

|-learning_rate: 1e-05

|-tuner/bracket: 1

|-tuner/epochs: 5

|-tuner/initial_epoch: 0

|-tuner/round: 0

```
Epoch 1/5
 89/625 [===>..........................] - ETA: 12s - loss: 0.6932 -
accuracy: 0.4982Buffered data was truncated after reaching the outpu
t size limit.
```

In [13]:

```
tuner.results_summary()
```

Results summary

|-Results in my_dir/intro_to_kt

|-Showing 10 best trials

|-Objective(name='val_accuracy', direction='max')

Trial summary

|-Trial ID: 1fe55e72d4001803a874adec087b71cd

|-Score: 0.833954393863678

|-Best step: 0

Hyperparameters:

|-Dense_1: 128

|-Dropout_1: 0.3

|-Dropout_2: 0.2

|-LSTM_1: 96

|-LSTM_2: 32

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 5

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: a904600182853f8283fa4565c8c3377e

Trial summary

|-Trial ID: 833f3fcde6b71e95c26ea273cf6047d4

|-Score: 0.833507239818573

|-Best step: 0

Hyperparameters:

|-Dense_1: 192

|-Dropout_1: 0.3

|-Dropout_2: 0.3

|-LSTM_1: 128

|-LSTM_2: 48

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 5

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: edd9a31c1c522937a35d244fe8d4e605

Trial summary

|-Trial ID: 75912d7821863268e0b2380d92948165

|-Score: 0.8324638605117798

|-Best step: 0

Hyperparameters:

|-Dense_1: 128

|-Dropout_1: 0.3

|-Dropout_2: 0.2

|-LSTM_1: 96

|-LSTM_2: 32

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 15

|-tuner/initial_epoch: 5

|-tuner/round: 2

|-tuner/trial_id: 1fe55e72d4001803a874adec087b71cd

Trial summary

|-Trial ID: 1c3268a2c48a69c54f122a3a111f141b

|-Score: 0.8311223983764648

|-Best step: 0

Hyperparameters:

|-Dense_1: 96

|-Dropout_1: 0.3

|-Dropout_2: 0.2

|-LSTM_1: 96

|-LSTM_2: 128

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

Trial summary

|-Trial ID: 89bb7a2b21a4bf70fa449f62af0df1ed

|-Score: 0.8285884857177734

|-Best step: 0

Hyperparameters:

|-Dense_1: 96

|-Dropout_1: 0.3

|-Dropout_2: 0.2

|-LSTM_1: 96

|-LSTM_2: 128

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 5

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: 1c3268a2c48a69c54f122a3a111f141b

Trial summary

|-Trial ID: 5679cb0722465ce8ac2a69ce2c0bfe1e

|-Score: 0.8270979523658752

|-Best step: 0

Hyperparameters:

|-Dense_1: 192

|-Dropout_1: 0.3

|-Dropout_2: 0.3

|-LSTM_1: 128

|-LSTM_2: 48

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 15

|-tuner/initial_epoch: 5

|-tuner/round: 2

|-tuner/trial_id: 833f3fcde6b71e95c26ea273cf6047d4

Trial summary

|-Trial ID: a904600182853f8283fa4565c8c3377e

|-Score: 0.8266507387161255

|-Best step: 0

Hyperparameters:

|-Dense_1: 128

|-Dropout_1: 0.3

|-Dropout_2: 0.2

|-LSTM_1: 96

|-LSTM_2: 32

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

Trial summary

|-Trial ID: 3aa08c89ed659b6c25d096a24d6e2c6c

|-Score: 0.8263526558876038

|-Best step: 0

Hyperparameters:

|-Dense_1: 64

|-Dropout_1: 0.3

|-Dropout_2: 0.3

|-LSTM_1: 128

|-LSTM_2: 96

|-learning_rate: 0.0001

|-tuner/bracket: 1

|-tuner/epochs: 15

|-tuner/initial_epoch: 5

|-tuner/round: 1

|-tuner/trial_id: deb6e4eff20cf378e95496eb4dd08208

Trial summary

|-Trial ID: deb6e4eff20cf378e95496eb4dd08208

|-Score: 0.8242658972740173

|-Best step: 0

Hyperparameters:

|-Dense_1: 64

|-Dropout_1: 0.3

|-Dropout_2: 0.3

|-LSTM_1: 128

|-LSTM_2: 96

|-learning_rate: 0.0001

|-tuner/bracket: 1

|-tuner/epochs: 5

|-tuner/initial_epoch: 0

|-tuner/round: 0

Trial summary

|-Trial ID: 2a157a62cb61e33d4deea94168ad0e8f

|-Score: 0.8211357593536377

|-Best step: 0

Hyperparameters:

|-Dense_1: 64

|-Dropout_1: 0.5

|-Dropout_2: 0.2

|-LSTM_1: 80

|-LSTM_2: 96

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 5

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: f0677c2a2d95c04dff94e633750c3435

In [ ]:

```
models = tuner.get_best_models(num_models=2)
```

In [ ]:

```
models[0]
```

In [ ]:

```
best_hps = tuner.get_best_hyperparameters(num_trials = 1)[0]
print(best_hps)
```

In [ ]:

```
model = tuner.hypermodel.build(best_hps)
```

In [ ]:

```
model.fit(training_padded, training_labels, epochs = 20, validation_data = (test
ing_padded, testing_labels), callbacks = [ClearTrainingOutput()])
```