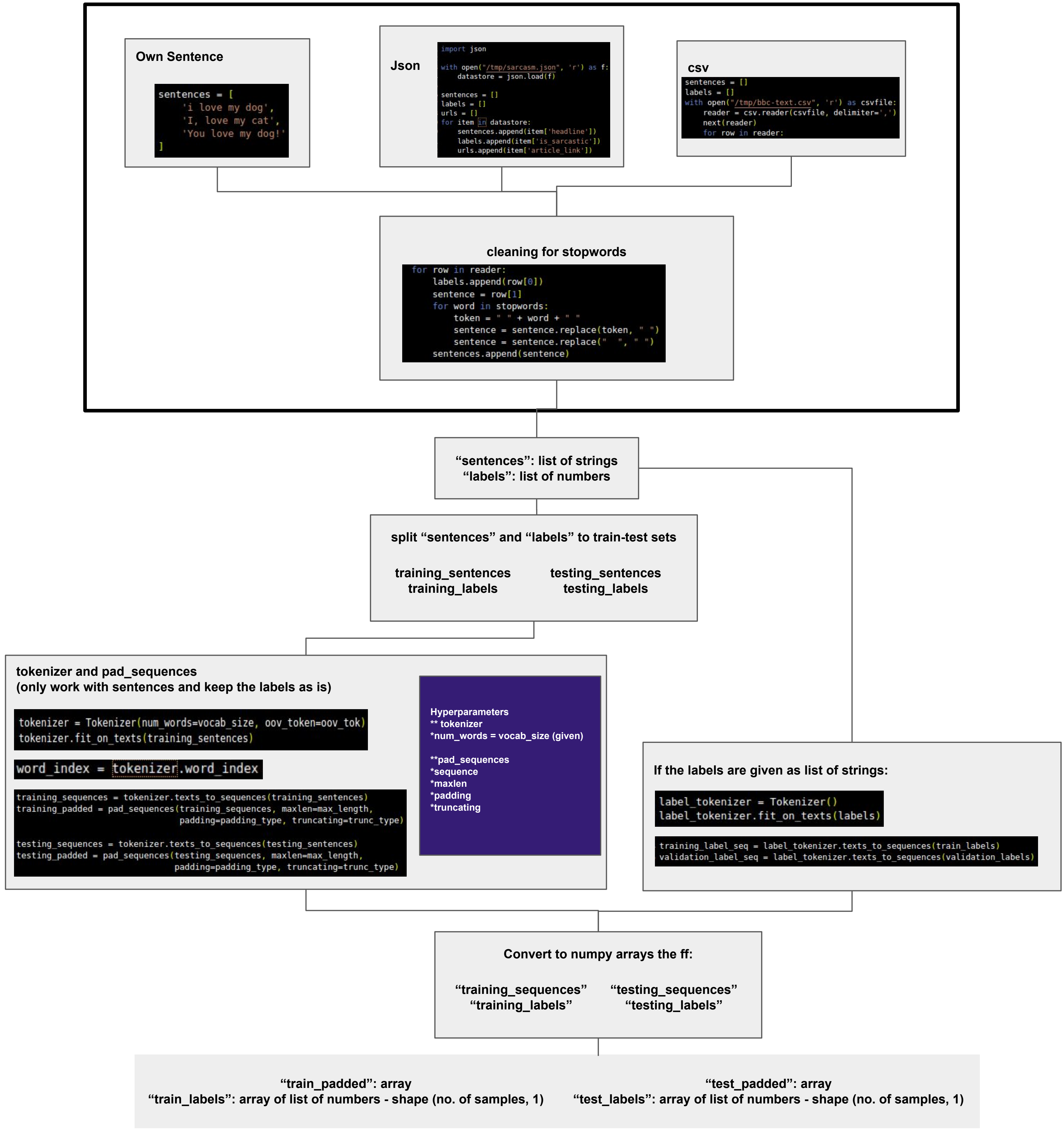
Word Embedding



Own Sentence

```
sentences = [
    'i love my dog',
    'I, love my cat',
    'You love my dog!'
]
```

Json

```
import json

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

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

csv

```
sentences = []
labels = []
with open("/tmp/bbc-text.csv", 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
```

cleaning for stopwords

```
for row in reader:
    labels.append(row[0])
    sentence = row[1]
    for word in stopwords:
        token = " " + word + " "
        sentence = sentence.replace(token, " ")
        sentence = sentence.replace("  ", " ")
    sentences.append(sentence)
```

"sentences": list of strings
"labels": list of numbers

split "sentences" and "labels" to train-test sets

training_sentences
training_labels

testing_sentences
testing_labels

tokenizer and pad_sequences
(only work with sentences and keep the labels as is)

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

```
word_index = tokenizer.word_index
```

```
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)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

Hyperparameters
** tokenizer
*num_words = vocab_size (given)

**pad_sequences
*sequence
*maxlen
*padding
*truncating

If the labels are given as list of strings:

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)
```

```
training_label_seq = label_tokenizer.texts_to_sequences(train_labels)
validation_label_seq = label_tokenizer.texts_to_sequences(validation_labels)
```

Convert to numpy arrays the ff:

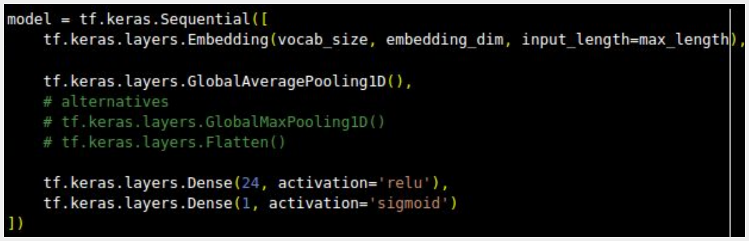
"training_sequences"
"training_labels"

"testing_sequences"
"testing_labels"

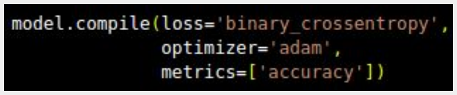
"train_padded": array
"train_labels": array of list of numbers - shape (no. of samples, 1)

"test_padded": array
"test_labels": array of list of numbers - shape (no. of samples, 1)

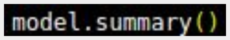
Build Model



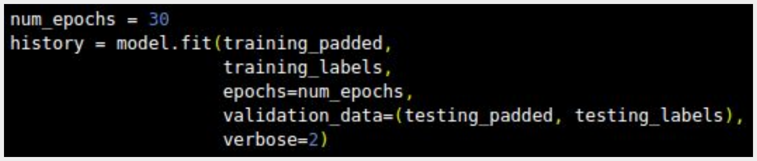
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),

    tf.keras.layers.GlobalAveragePooling1D(),
    # alternatives
    # tf.keras.layers.GlobalMaxPooling1D()
    # tf.keras.layers.Flatten()

    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```



```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

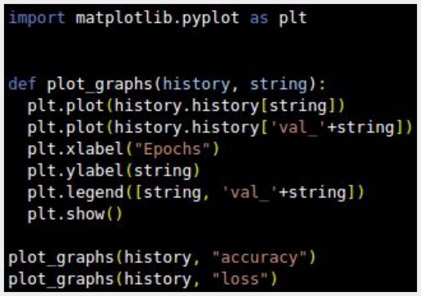


```
model.summary()
```



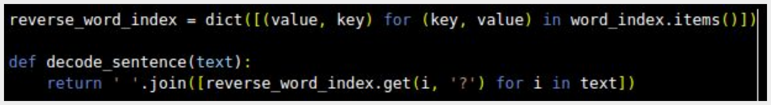
```
num_epochs = 30
history = model.fit(training_padded,
                    training_labels,
                    epochs=num_epochs,
                    validation_data=(testing_padded, testing_labels),
                    verbose=2)
```

Plot Loss and Accuracy

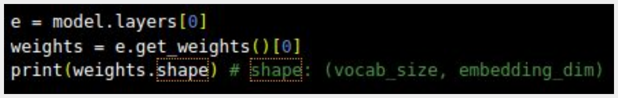


```
import matplotlib.pyplot as plt


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

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

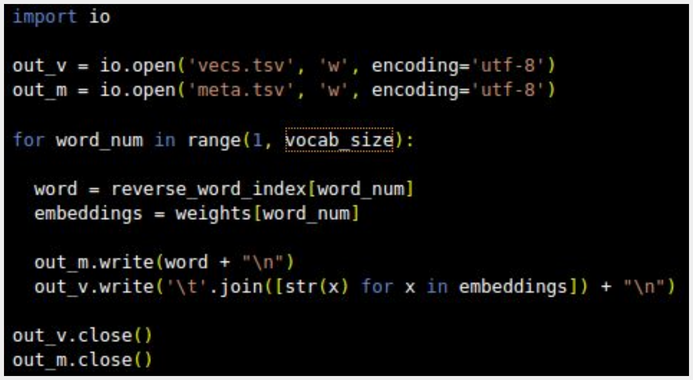
Creating tsv data



```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

def decode_sentence(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])
```

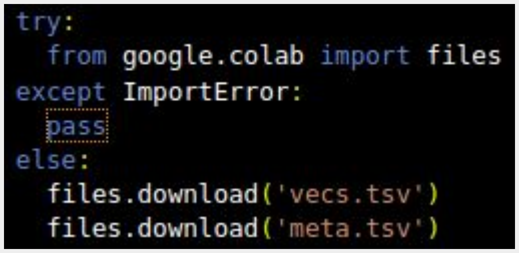


```
e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)
```



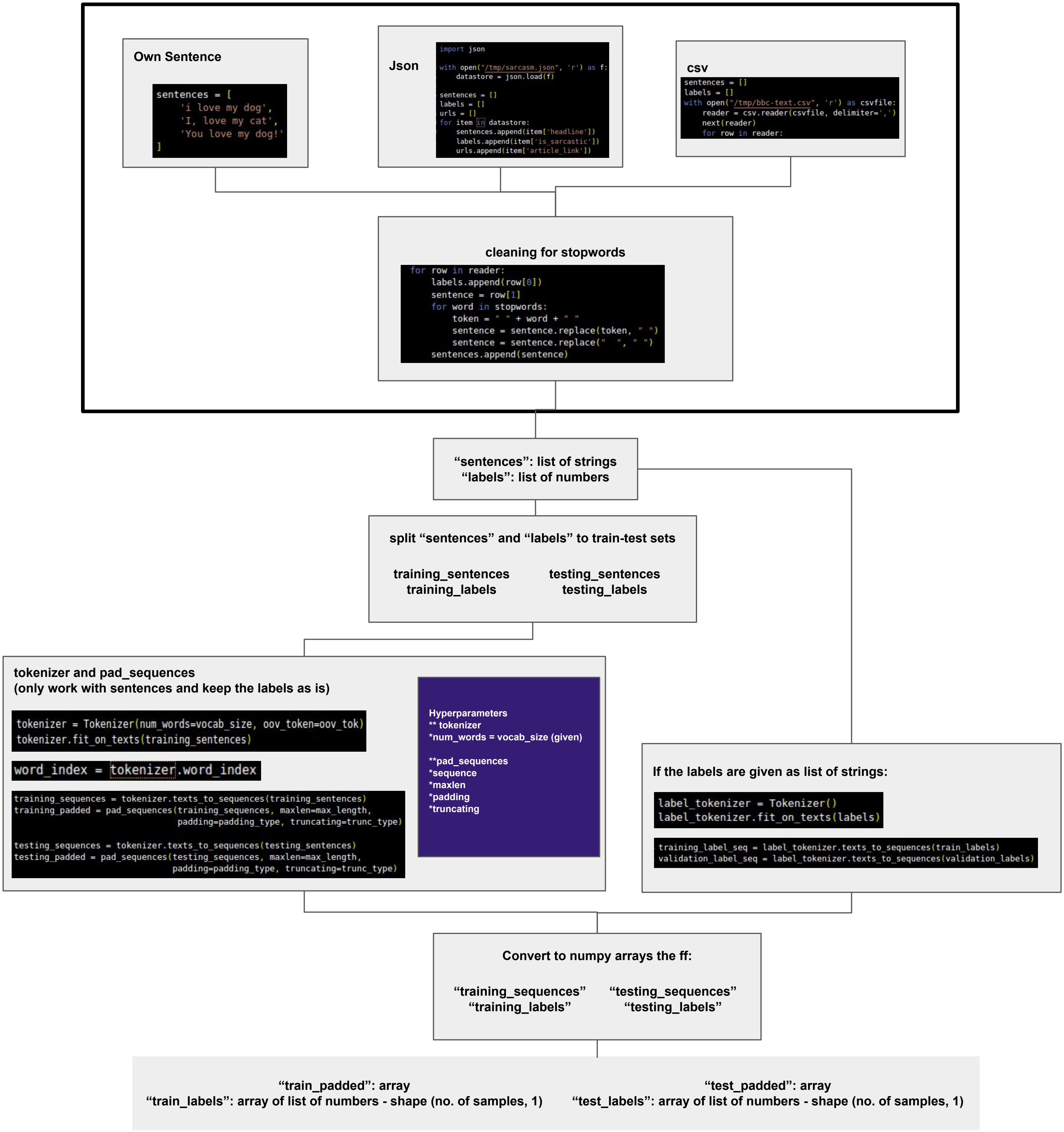
```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')

for word_num in range(1, vocab_size):

  word = reverse_word_index[word_num]
  embeddings = weights[word_num]

  out_m.write(word + "\n")
  out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")

out_v.close()
out_m.close()
```

*Download files from colab



```
try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download('vecs.tsv')
  files.download('meta.tsv')
```

Classification without Transfer Learning



Own Sentence

```
sentences = [
    'i love my dog',
    'I, love my cat',
    'You love my dog!'
]
```

Json

```
import json

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

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

csv

```
sentences = []
labels = []
with open("/tmp/bbc-text.csv", 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
```

cleaning for stopwords

```
for row in reader:
    labels.append(row[0])
    sentence = row[1]
    for word in stopwords:
        token = " " + word + " "
        sentence = sentence.replace(token, " ")
        sentence = sentence.replace("  ", " ")
    sentences.append(sentence)
```

"sentences": list of strings
"labels": list of numbers

split "sentences" and "labels" to train-test sets

training_sentences
training_labels

testing_sentences
testing_labels

tokenizer and pad_sequences
(only work with sentences and keep the labels as is)

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

```
word_index = tokenizer.word_index
```

```
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)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

Hyperparameters
** tokenizer
*num_words = vocab_size (given)

**pad_sequences
*sequence
*maxlen
*padding
*truncating

If the labels are given as list of strings:

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)
```

```
training_label_seq = label_tokenizer.texts_to_sequences(train_labels)
validation_label_seq = label_tokenizer.texts_to_sequences(validation_labels)
```

Convert to numpy arrays the ff:

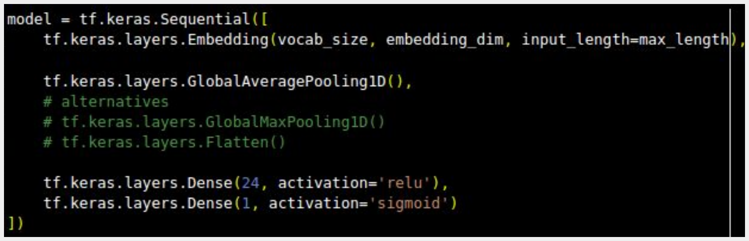
"training_sequences"
"training_labels"

"testing_sequences"
"testing_labels"

"train_padded": array
"train_labels": array of list of numbers - shape (no. of samples, 1)

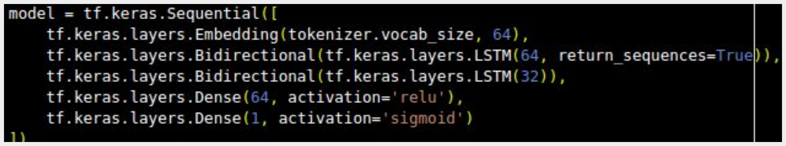
"test_padded": array
"test_labels": array of list of numbers - shape (no. of samples, 1)

Embedding Layer only and GlobalAveragePooling1D



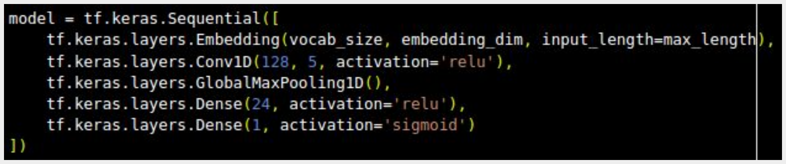
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),

    tf.keras.layers.GlobalAveragePooling1D(),
    # alternatives
    # tf.keras.layers.GlobalMaxPooling1D()
    # tf.keras.layers.Flatten()

    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Bidirectional and Stacked LSTMs



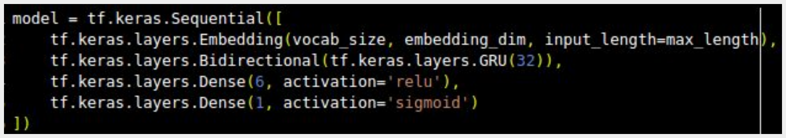
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Conv1D



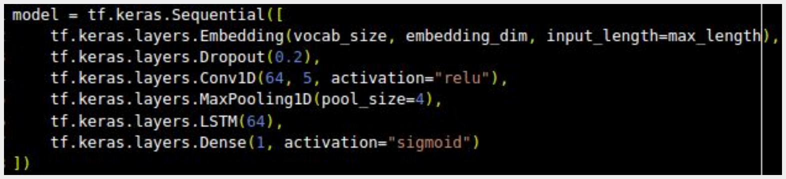
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalMaxPooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

GRU



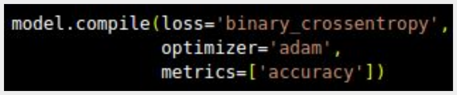
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32)),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Stacking LSTM and Conv1D



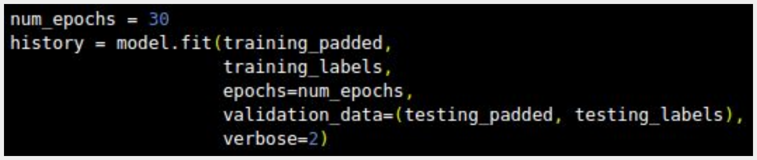
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(64, 5, activation="relu"),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

Model Compile, Summary, Fit



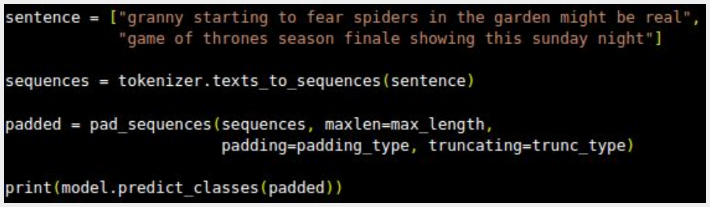
```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
model.summary()
```



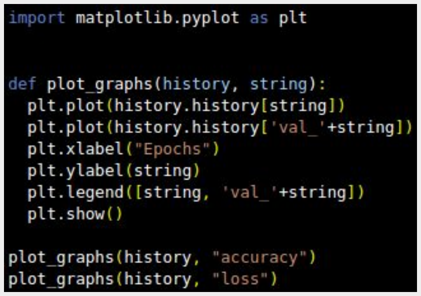
```
num_epochs = 30
history = model.fit(training_padded,
                    training_labels,
                    epochs=num_epochs,
                    validation_data=(testing_padded, testing_labels),
                    verbose=2)
```

Model Predict



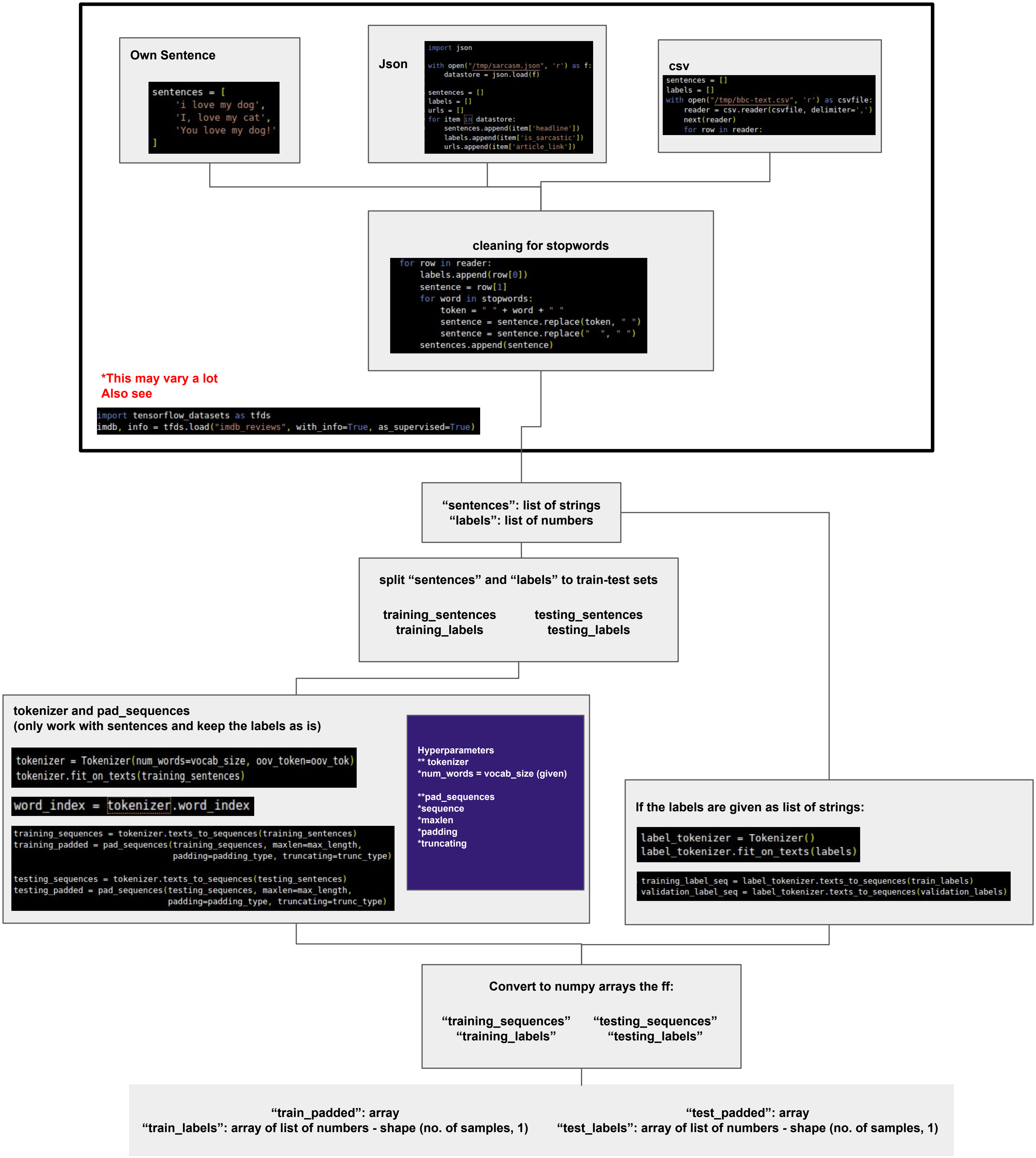
```
sentence = ["granny starting to fear spiders in the garden might be real",
            "game of thrones season finale showing this sunday night"]

sequences = tokenizer.texts_to_sequences(sentence)

padded = pad_sequences(sequences, maxlen=max_length,
                       padding=padding_type, truncating=trunc_type)

print(model.predict_classes(padded))
```

Plot Loss and Accuracy



```
import matplotlib.pyplot as plt


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

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

Classification with Transfer Learning - week 3



Own Sentence

```
sentences = [
    'i love my dog',
    'I, love my cat',
    'You love my dog!'
]
```

Json

```
import json

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

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

csv

```
sentences = []
labels = []
with open("/tmp/bbc-text.csv", 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
```

cleaning for stopwords

```
for row in reader:
    labels.append(row[0])
    sentence = row[1]
    for word in stopwords:
        token = " " + word + " "
        sentence = sentence.replace(token, " ")
        sentence = sentence.replace("  ", " ")
    sentences.append(sentence)
```

*This may vary a lot
Also see

```
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```

“sentences”: list of strings
“labels”: list of numbers

split “sentences” and “labels” to train-test sets

training_sentences
training_labels

testing_sentences
testing_labels

tokenizer and pad_sequences
(only work with sentences and keep the labels as is)

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

```
word_index = tokenizer.word_index
```

```
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)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

Hyperparameters
** tokenizer
*num_words = vocab_size (given)

**pad_sequences
*sequence
*maxlen
*padding
*truncating

If the labels are given as list of strings:

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)
```

```
training_label_seq = label_tokenizer.texts_to_sequences(train_labels)
validation_label_seq = label_tokenizer.texts_to_sequences(validation_labels)
```

Convert to numpy arrays the ff:

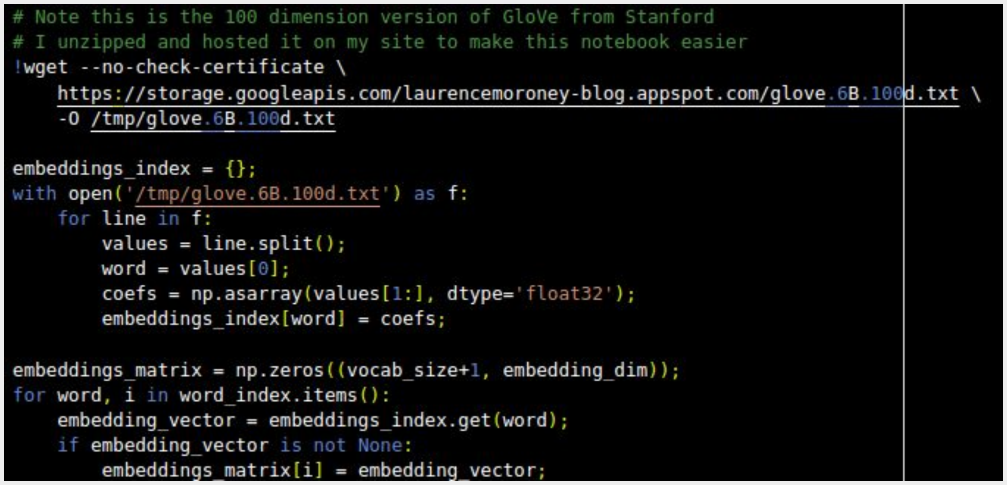
“training_sequences”
“training_labels”

“testing_sequences”
“testing_labels”

“train_padded”: array
“train_labels”: array of list of numbers - shape (no. of samples, 1)

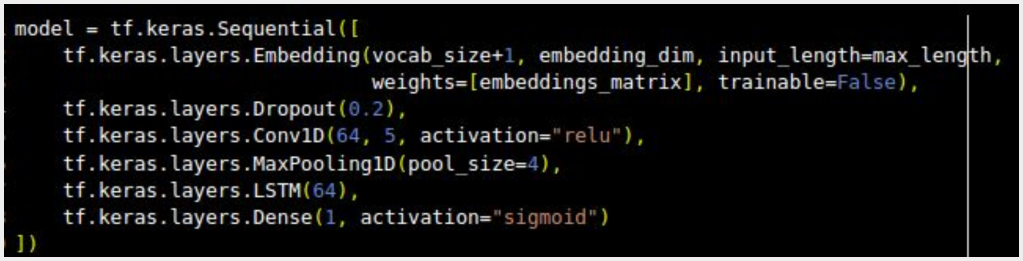
“test_padded”: array
“test_labels”: array of list of numbers - shape (no. of samples, 1)

Get Embeddings from GloVe



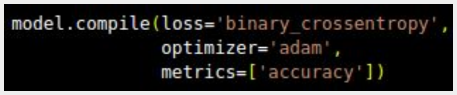
```
# Note this is the 100 dimension version of GloVe from Stanford
# I unzipped and hosted it on my site to make this notebook easier
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/glove.6B.100d.txt \
    -O /tmp/glove.6B.100d.txt

embeddings_index = {};
with open('/tmp/glove.6B.100d.txt') as f:
    for line in f:
        values = line.split();
        word = values[0];
        coefs = np.asarray(values[1:], dtype='float32');
        embeddings_index[word] = coefs;

embeddings_matrix = np.zeros((vocab_size+1, embedding_dim));
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word);
    if embedding_vector is not None:
        embeddings_matrix[i] = embedding_vector;
```

Model (See changes in Embedding layer)

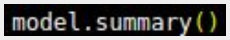


```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length,
                              weights=[embeddings_matrix], trainable=False),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(64, 5, activation="relu"),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

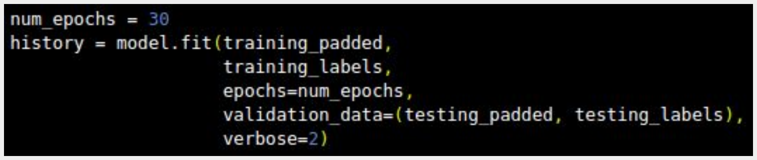
Model Compile, Summary, Fit



```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

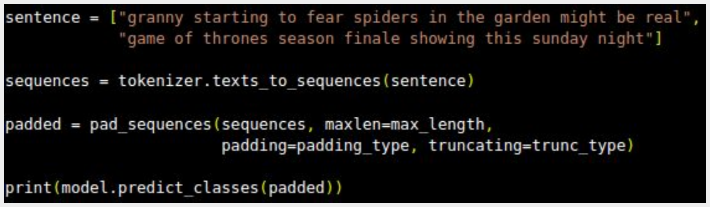


```
model.summary()
```



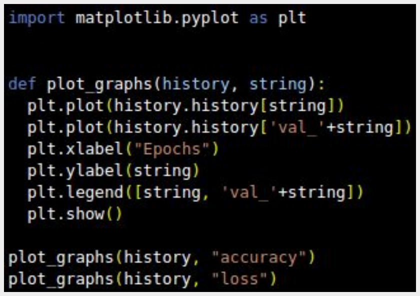
```
num_epochs = 30
history = model.fit(training_padded,
                    training_labels,
                    epochs=num_epochs,
                    validation_data=(testing_padded, testing_labels),
                    verbose=2)
```

Model Predict



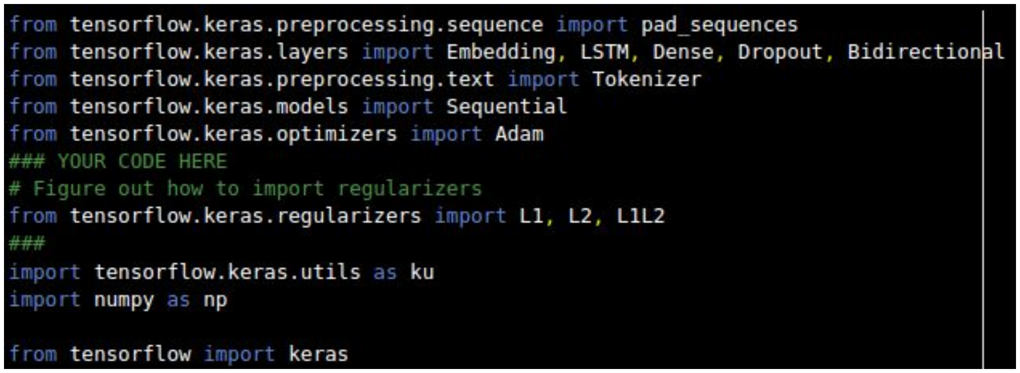
```
sentence = ["granny starting to fear spiders in the garden might be real",
            "game of thrones season finale showing this sunday night"]

sequences = tokenizer.texts_to_sequences(sentence)

padded = pad_sequences(sequences, maxlen=max_length,
                       padding=padding_type, truncating=trunc_type)

print(model.predict_classes(padded))
```

Plot Loss and Accuracy



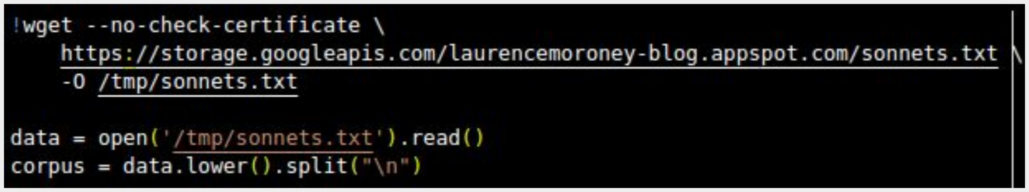
```
import matplotlib.pyplot as plt


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

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

Text(Song/Poetry) Generation



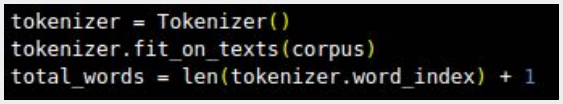
```python
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
### YOUR CODE HERE
# Figure out how to import regularizers
from tensorflow.keras.regularizers import L1, L2, L1L2
###
import tensorflow.keras.utils as ku
import numpy as np

from tensorflow import keras
```

Get the data



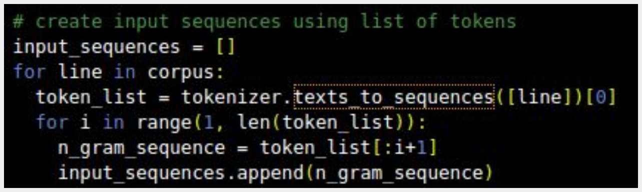
```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sonnets.txt \
    -O /tmp/sonnets.txt

data = open('/tmp/sonnets.txt').read()
corpus = data.lower().split("\n")
```

Tokenize the corpus



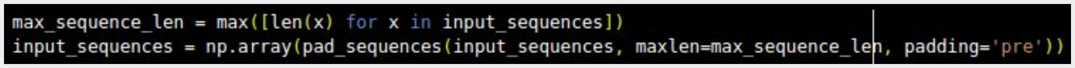
```python
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

N-gram and Input Sequences



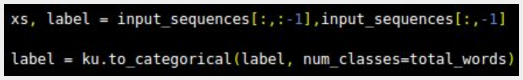
```python
# create input sequences using list of tokens
input_sequences = []
for line in corpus:
  token_list = tokenizer.texts_to_sequences([line])[0]
  for i in range(1, len(token_list)):
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)
```

Pad Sequences



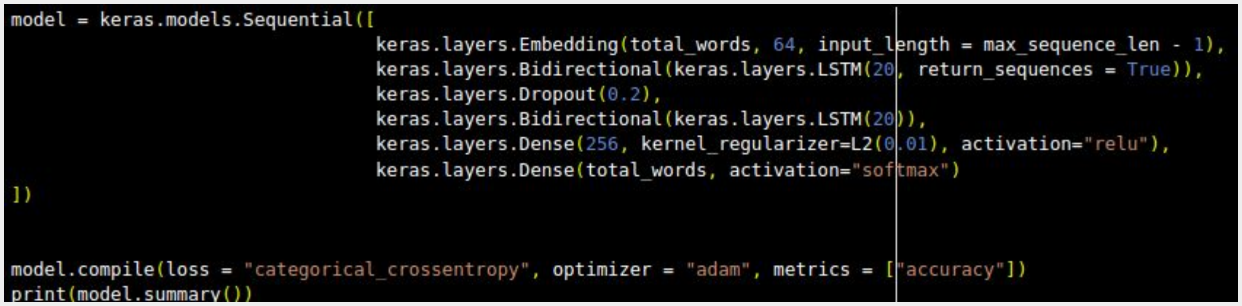
```python
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

Inputs and Labels

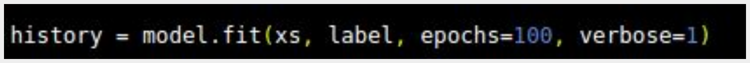


```python
xs, label = input_sequences[:,:-1],input_sequences[:,-1]

label = ku.to_categorical(label, num_classes=total_words)
```

Model

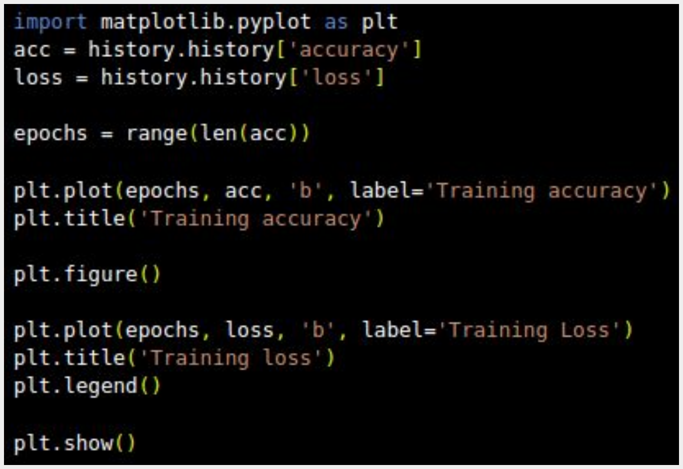


```
model = keras.models.Sequential([
                                 keras.layers.Embedding(total_words, 64, input_length = max_sequence_len - 1),
                                 keras.layers.Bidirectional(keras.layers.LSTM(20, return_sequences = True)),
                                 keras.layers.Dropout(0.2),
                                 keras.layers.Bidirectional(keras.layers.LSTM(20)),
                                 keras.layers.Dense(256, kernel_regularizer=L2(0.01), activation="relu"),
                                 keras.layers.Dense(total_words, activation="softmax")
])


model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
print(model.summary())
```



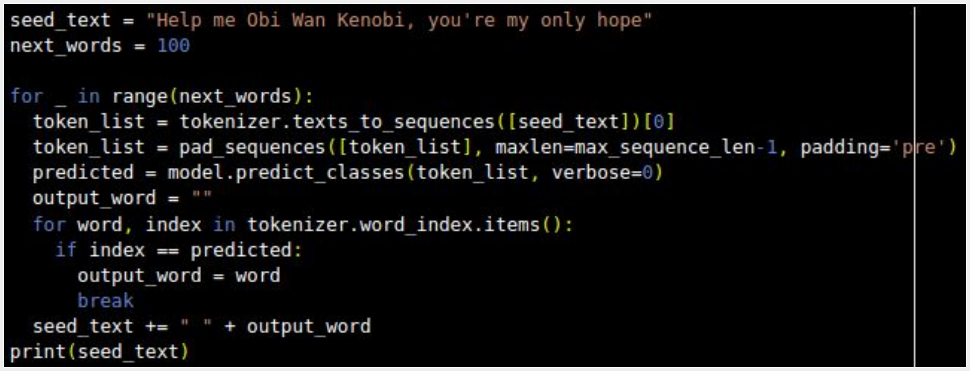
```
history = model.fit(xs, label, epochs=100, verbose=1)
```

Plot Accuracy and Loss



```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
loss = history.history['loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')

plt.figure()

plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()

plt.show()
```

Predict - Generate Sonnet/Songs/Poetry



```
seed_text = "Help me Obi Wan Kenobi, you're my only hope"
next_words = 100

for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
  predicted = model.predict_classes(token_list, verbose=0)
  output_word = ""
  for word, index in tokenizer.word_index.items():
    if index == predicted:
      output_word = word
      break
  seed_text += " " + output_word
print(seed_text)
```