LEARNING OF SEMANTICALLY EQUIVALENT QUESTIONS

```
# importing libraries
import keras
import nltk
import tensorflow as tf
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

```
# import dataset from drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /content/gdrive
```

IMPORTING DATASET FROM KAGGLE (optional)

The dataset can be downloaded from this website: https://www.kaggle.com/c/quora-question-pairs/data

The below cells shows how the dataset can be extracted to the notebook from kaggle. To import the dataset from ka kaggle.json from Kaggle -- MyAccount -- Create New API Token - auto downloads as "kaggle.json

```
from google.colab import files

files.upload() #this will prompt you to upload the kaggle.json to the notebook
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 66 bytes, last modified: 7/25/2019 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"hshreedhar","key":"e96183fe90a76e704596438aec7c338
```

```
!pip install -q kaggle
!mkdir -p ~/.kaggle
```



```
kaggle.json
```

```
import os
os.chdir('/content/gdrive/My Drive/Kaggle/kaggle-questionpair/input')
```

```
!kaggle competitions download -c quora-question-pairs --force
```

```
Downloading sample_submission.csv.zip to /content/gdrive/My Drive/Kaggle/kaggle-q
 20% 1.00M/4.95M [00:00<00:01, 2.20MB/s]
100% 4.95M/4.95M [00:00<00:00, 10.1MB/s]
Downloading train.csv.zip to /content/gdrive/My Drive/Kaggle/kaggle-questionpair/
 80% 17.0M/21.2M [00:01<00:04, 1.05MB/s]
100% 21.2M/21.2M [00:01<00:00, 14.2MB/s]
Downloading test.csv.zip to /content/gdrive/My Drive/Kaggle/kaggle-questionpair/i
 93% 105M/112M [00:08<00:00, 10.1MB/s]
100% 112M/112M [00:08<00:00, 13.7MB/s]
```

PREPARING THE DATASET

```
# Below is a sample dataset
# Dataset for training and validation
df = pd.read_csv('../input/train.csv') # read the CSV file
df.head() # to vieww the first 5 contents in a file
```

	id	qid1	qid2	question1	
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guid
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian (
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increa
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]2
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would surv

We are setting the size of the training dataset in the block below. NUM_OF_SAMPLES can be changed as desired.

NUM_OF_SAMPLES is the number of data points from each class. The total number of datapoints is 2*NUM_OF_SAI

```
# to get equal number of duplicate and non duplicate question pairs in the dataset
NUM_OF_SAMPLES = 10000 # Using smaller sample for training set, for each subset.

df_dup = df[df['is_duplicate'] == 1]
df_dup = df_dup[0:NUM_OF_SAMPLES]

df_diff = df[df['is_duplicate'] == 0]
df_diff = df_diff[0:NUM_OF_SAMPLES]

len(df_dup)
```


Limiting the size of the data set for an unbiased training

in the dataset, 60% of the question-pairs do not have a duplicate pair and 40% have a duplicate pair. We are creating qustion-pair with and without duplicates for an unbiased training. Off this sub-dataset that is created, we'll use 80% f

```
# here, we are spitting the dataset in training and validation, with 80% of the question-pai

num_of_train = int(round(0.8*NUM_OF_SAMPLES))        # Number of samples for training
num_of_valid = int(NUM_OF_SAMPLES - num_of_train)    # Number of samples for validation


# Choosing NUM_OF_SAMPLES random question pair for both duplicate and non-duplicate labels
dup_indices = np.random.choice(df_dup.index,size=num_of_train)
df_dup_train = df_dup.loc[dup_indices]

rem_indices = set(df_dup.index).difference(set(dup_indices)) # Remaining indices
dup_valid_indices = np.random.choice(list(rem_indices),num_of_valid)
df_dup_valid = df_dup.loc[dup_valid_indices]

diff_indices = np.random.choice(df_diff.index,size=num_of_train)
df_diff_train = df_diff.loc[diff_indices]

rem_indices = set(df_diff.index).difference(set(diff_indices)) # Remaining indices
diff_valid_indices = np.random.choice(list(rem_indices),num_of_valid)
df_diff_valid = df_diff.loc[diff_valid_indices]



# Selecting Training and Validation data points
df_train = pd.concat([df_dup_train,df_diff_train])
df_valid = pd.concat([df_dup_valid, df_diff_valid])

df_train_label = df_dup_train['is_duplicate']
df_train_label = df_train_label.append(df_diff_train['is_duplicate'])

df_valid_label = df_dup_valid['is_duplicate']
df_valid_label = df_valid_label.append(df_diff_valid['is_duplicate'])

df_train = df_train[['question1','question2']]
df_valid = df_valid[['question1', 'question2']]
```

Here, we have our dataset ready for training "df_train" and "df_valid" for validation. Duplicate values are the lables

```
# this block is to fetch the total number of duplicate questomn pair in the training set
print('Total number of question pain in the training set: {}'.format(len(df_train)))
```

```
Total number of question pain in the training set: 16000
```

```
print(df_train.shape)
print(df_valid.shape)
```

```
(16000, 2)
(4000, 2)
```

Automatic document saving has been pending for 26 minutes. Reloading may fix the problem. Save and reload the page.

```
# cleaning up of the dataset
import re
import string

re_punc = re.compile('[%s]' % re.escape(string.punctuation))
re_print = re.compile('[^%s]' % re.escape(string.printable))
```

```
def pre_process_line (line):
  # tokenize on white space
  try:
    line = line.split()
    # convert to lowercase
    line = [word.lower() for word in line]
    # remove punctuation from each token
    line = [re_punc.sub('', w) for w in line]
    # remove non-printable chars form each token
    line = [re_print.sub('', w) for w in line]
    # remove tokens with numbers in them
    line = [word for word in line if word.isalpha()]
    return ' '.join(line)
  except:
    return ' '

def pre_process_questions(ques):
  df_processed = ques.copy(deep=True)
  for i in range(ques.shape[0]):
    line = pre_process_line(ques.iloc[i])
    df_processed.iloc[i] = line
  return df_processed


df_train_q1 = df_train['question1'].apply(pre_process_line)
df_train_q2 = df_train['question2'].apply(pre_process_line)
df_valid_q1 = df_valid['question1'].apply(pre_process_line)
df_valid_q2 = df_valid['question2'].apply(pre_process_line)
```

Encoding the questions

The word embedding (from q1 and q2) are split to training and validation datasets. Further, the words within the que

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences


all_qs = df_train_q1.tolist() + df_train_q2.tolist() + df_valid_q1.tolist() + df_valid_q2.tc

tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_qs)
vocab_size = len(tokenizer.word_index) + 1
encoded_qs = tokenizer.texts_to_sequences(all_qs)

MAX_QUES_LEN = 300
padded_qs = pad_sequences(encoded_qs,maxlen=MAX_QUES_LEN,padding='post')
```

```
# Divide the questions

encoded_train_q1 = padded_qs[0:(2*num_of_train)]
encoded_train_q2 = padded_qs[(2*num_of_train):(4*num_of_train)]
encoded_valid_q1 = padded_qs[(4*num_of_train):(4*num_of_train+2*num_of_valid)]
```

Automatic document saving has been pending for 26 minutes. Reloading may fix the problem. Save and reload the page.

```
encoded_valid_q1.shape
```

```
(4000, 300)
```

Initializing Embedding layer using GloVe

```
import gensim
from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec

glove_file = datapath('/content/gdrive/My Drive/NLP- Deep Learning/glove.6B.300d.txt')
word2vec_glove_file = get_tmpfile("glove.6B.300d.word2vec.txt")
glove2word2vec(glove_file, word2vec_glove_file)
WORD2VEC = KeyedVectors.load_word2vec_format(word2vec_glove_file)
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:398: UserWarn
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
```

```
EMBEDDING_DIM = 300 # embedding dimentions are set to 300 as we are training on a 300 diment

def embeddings_init(shape,dtype=None):
    embedding_matrix = np.zeros((vocab_size,EMBEDDING_DIM))
    not_present = []
    for word, idx in tokenizer.word_index.items():
      try:
        vec = WORD2VEC[word]
        if vec is not None:
          embedding_matrix[idx] = vec
      except:
        not_present.append(word)
        continue
    print(len(not_present))
    return embedding_matrix
```

TRAIN THE MODEL

```
import keras
from keras import backend as K
from keras.layers import Input, Dense, Conv1D, MaxPooling1D, Flatten, Embedding
from keras import models
from keras.optimizers import SGD
```

```
def get_siamese_model(input_shape):
  """
    Model architecture
  """

  q1_input = Input(input_shape)
```

Automatic document saving has been pending for 26 minutes. Reloading may fix the problem. Save and reload the page.

```
  model.add(Embedding(vocab_size,EMBEDDING_DIM,input_length=MAX_QUES_LEN,
                      embeddings_initializer=embeddings_init))

  # 1st is a conv layer with max-pooling with 300 neurons with kernel size as 15
  model.add(Conv1D(300, 15,activation='tanh')) # Conv1D
  model.add(MaxPooling1D(pool_size=2))
```

```
    # next will ne deep network of 3 fully-connected layer with 50 neuron each

    model.add(Dense(50, activation='tanh'))
    model.add(Dense(50, activation='tanh'))
    model.add(Dense(50, activation ='tanh'))

    # Generate the encodings (feature vectors) for the two images
    model_q1 = model(q1_input)
    model_q2 = model(q2_input)

    # concatenating the outputs from the 2 model-layers of the siamese network
    concat = keras.layers.Concatenate()([model_q1, model_q2])
    flatcat = Flatten()(concat)

    # Add a dense layer with sigmoid to generate the similarity score
    prediction = Dense(1,activation='tanh')(flatcat)

    # Connect the inputs with the outputs
    siamese_net = models.Model(inputs=[q1_input, q2_input],outputs=prediction)

    return siamese_net,model


# creating the network
siamese_net, seqmodel = get_siamese_model((MAX_QUES_LEN,))
siamese_net.summary()
seqmodel.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0805 03:39:46.018667 140377308571520 deprecation_wrapper.py:119] From /usr/local

W0805 03:39:46.067380 140377308571520 deprecation_wrapper.py:119] From /usr/local

W0805 03:39:46.261919 140377308571520 deprecation_wrapper.py:119] From /usr/local

W0805 03:39:46.294217 140377308571520 deprecation_wrapper.py:119] From /usr/local

2119
________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
input_1 (InputLayer)            (None, 300)          0
________________________________________________________________________________
input_2 (InputLayer)            (None, 300)          0
________________________________________________________________________________
sequential_1 (Sequential)       (None, 143, 50)      6479450     input_1[0][0]
                                                                 input_2[0][0]
________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 143, 100)     0           sequential_1[1][
                                                                 sequential_1[2][
________________________________________________________________________________
flatten_1 (Flatten)             (None, 14300)        0           concatenate_1[0]
________________________________________________________________________________
dense_4 (Dense)                 (None, 1)            14301       flatten_1[0][0]
================================================================================
Total params: 6,493,751
Trainable params: 6,493,751
Non-trainable params: 0
________________________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 300, 300)          5109000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 286, 300)          1350300
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 143, 300)          0
_________________________________________________________________
dense_1 (Dense)              (None, 143, 50)           15050
_________________________________________________________________
dense_2 (Dense)              (None, 143, 50)           2550
_________________________________________________________________
dense_3 (Dense)              (None, 143, 50)           2550
=================================================================
Total params: 6,479,450
Trainable params: 6,479,450
```


FIT THE DATA

```
# Define optimiser, learning rate
optimizer = SGD(lr = 0.01)
siamese_net.compile(loss="mean_squared_error",optimizer=optimizer,metrics=['accuracy'])
history = siamese_net.fit([encoded_train_q1,encoded_train_q2], df_train_label, validation_da
```

```
W0805 03:39:50.902232 140377308571520 deprecation_wrapper.py:119] From /usr/local

W0805 03:39:51.193379 140377308571520 deprecation_wrapper.py:119] From /usr/local

W0805 03:39:51.259613 140377308571520 deprecation_wrapper.py:119] From /usr/local

Train on 16000 samples, validate on 4000 samples
Epoch 1/20
 - 22s - loss: 0.2373 - acc: 0.5990 - val_loss: 0.2351 - val_acc: 0.5992
Epoch 2/20
 - 15s - loss: 0.2222 - acc: 0.6422 - val_loss: 0.2305 - val_acc: 0.6170
Epoch 3/20
 - 15s - loss: 0.2113 - acc: 0.6691 - val_loss: 0.2293 - val_acc: 0.6138
Epoch 4/20
 - 16s - loss: 0.1998 - acc: 0.6974 - val_loss: 0.2265 - val_acc: 0.6338
Epoch 5/20
 - 16s - loss: 0.1871 - acc: 0.7258 - val_loss: 0.2260 - val_acc: 0.6392
Epoch 6/20
 - 16s - loss: 0.1748 - acc: 0.7508 - val_loss: 0.2304 - val_acc: 0.6395
Epoch 7/20
 - 16s - loss: 0.1601 - acc: 0.7794 - val_loss: 0.2323 - val_acc: 0.6415
Epoch 8/20
 - 16s - loss: 0.1432 - acc: 0.8158 - val_loss: 0.2405 - val_acc: 0.6325
Epoch 9/20
 - 16s - loss: 0.1276 - acc: 0.8435 - val_loss: 0.2480 - val_acc: 0.6272
Epoch 10/20
 - 16s - loss: 0.1109 - acc: 0.8740 - val_loss: 0.2521 - val_acc: 0.6225
Epoch 11/20
 - 16s - loss: 0.0949 - acc: 0.8994 - val_loss: 0.2644 - val_acc: 0.6175
Epoch 12/20
 - 16s - loss: 0.0819 - acc: 0.9193 - val_loss: 0.2650 - val_acc: 0.6148
Epoch 13/20
 - 16s - loss: 0.0695 - acc: 0.9374 - val_loss: 0.2666 - val_acc: 0.5980
Epoch 14/20
 - 16s - loss: 0.0574 - acc: 0.9546 - val_loss: 0.2720 - val_acc: 0.6038
Epoch 15/20
 - 16s - loss: 0.0480 - acc: 0.9647 - val_loss: 0.2879 - val_acc: 0.5958
Epoch 16/20
 - 16s - loss: 0.0402 - acc: 0.9734 - val_loss: 0.2891 - val_acc: 0.6012
Epoch 17/20
 - 16s - loss: 0.0329 - acc: 0.9811 - val_loss: 0.3126 - val_acc: 0.5948
Epoch 18/20
 - 16s - loss: 0.0277 - acc: 0.9857 - val_loss: 0.2973 - val_acc: 0.5975
Epoch 19/20
```

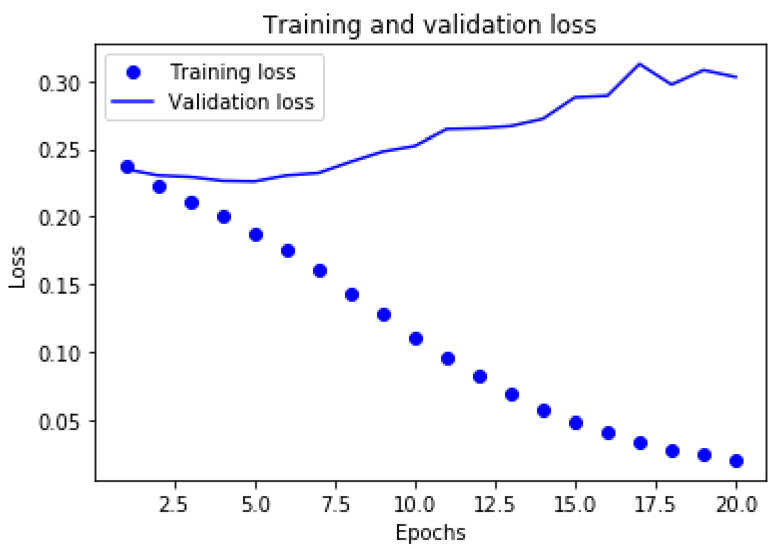


```
print('Number of training samples = {}'.format(num_of_train))
print('Number of validation samples = {}'.format(num_of_valid))
print('Vocabulary size = {}'.format(vocab_size))
```

```
Number of training samples = 8000
Number of validation samples = 2000
Vocabulary size = 17030
```

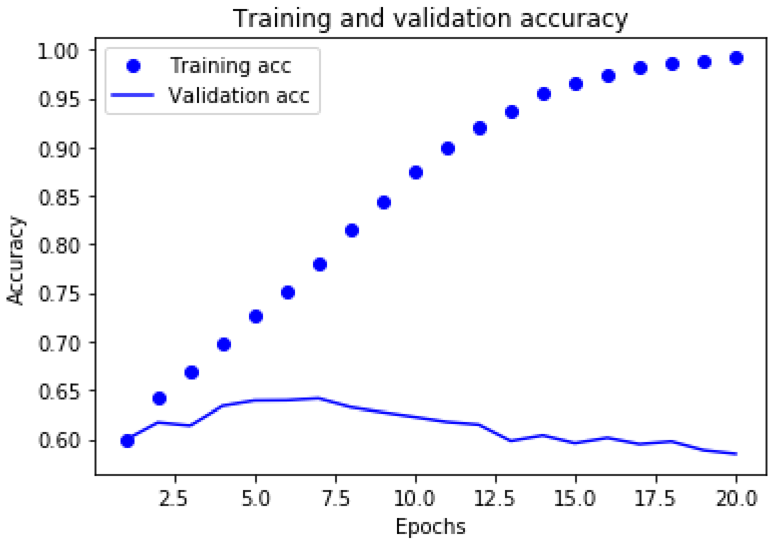
PLOT GRAPHS

```
# Plotting training and validation loss (from lecture 2 notes)
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Plotting training and validation accuracy (from lecture 2 notes)
acc = history.history['acc']
val_acc = history.history['val_acc']
epochs = range(1,len(acc)+1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

Automatic document saving has been pending for 26 minutes. Reloading may fix the problem. Save and reload the page.


Training and validation accuracy
Training acc
Validation acc
Accuracy
Epochs
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0