# # CODE

# First, lets import all the necessary packages

import os

os.environ["CUDA\_VISIBLE\_DEVICES"] = "1" #USE GPU

from time import time

import pandas as pd

import numpy as np

from gensim.models import KeyedVectors

import re

from nltk.corpus import stopwords

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import seaborn as sns

import itertools

import datetime

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Model

from keras.layers import Input, Embedding, LSTM, Merge

import keras.backend as K

from keras.optimizers import Adadelta

from keras.callbacks import ModelCheckpoint

# Global variables

# File paths

TRAIN\_CSV = '.../home/Medium/train.csv'

TEST\_CSV = '.../home/Medium/test.csv'

EMBEDDING\_FILE = '.../home/Medium/GoogleNews-vectors-negative300.bin.gz'

#EMBEDDING\_FILE = '.../home/Medium/wiki-news-300d-1M.vec'

MODEL\_SAVING\_DIR = '.../home/ecohen/HDD/HDD4/Models/stackoverflow/'

# Create embedding matrix

# Load training and test set

train\_df = pd.read\_csv(TRAIN\_CSV, quoting=3, error\_bad\_lines=False)

test\_df = pd.read\_csv(TEST\_CSV)

stops = set(stopwords.words('english'))

def text\_to\_word\_list(text):

''' Pre process and convert texts to a list of words '''

text = str(text)

text = text.lower()

# Clean the text

text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)

text = re.sub(r"what's", "what is ", text)

text = re.sub(r"\'s", " ", text)

text = re.sub(r"\'ve", " have ", text)

text = re.sub(r"can't", "cannot ", text)

text = re.sub(r"n't", " not ", text)

text = re.sub(r"i'm", "i am ", text)

text = re.sub(r"\'re", " are ", text)

text = re.sub(r"\'d", " would ", text)

text = re.sub(r"\'ll", " will ", text)

text = re.sub(r",", " ", text)

text = re.sub(r"\.", " ", text)

text = re.sub(r"!", " ! ", text)

text = re.sub(r"\/", " ", text)

text = re.sub(r"\^", " ^ ", text)

text = re.sub(r"\+", " + ", text)

text = re.sub(r"\-", " - ", text)

text = re.sub(r"\=", " = ", text)

text = re.sub(r"'", " ", text)

text = re.sub(r"(\d+)(k)", r"\g<1>000", text)

text = re.sub(r":", " : ", text)

text = re.sub(r" e g ", " eg ", text)

text = re.sub(r" b g ", " bg ", text)

text = re.sub(r" u s ", " american ", text)

text = re.sub(r"\0s", "0", text)

text = re.sub(r" 9 11 ", "911", text)

text = re.sub(r"e - mail", "email", text)

text = re.sub(r"j k", "jk", text)

text = re.sub(r"\s{2,}", " ", text)

text = text.split()

return text

# Prepare embedding

vocabulary = dict()

inverse\_vocabulary = ['<unk>'] # '<unk>' will never be used, it is only a placeholder for the [0, 0, ....0] embedding

word2vec = KeyedVectors.load\_word2vec\_format(EMBEDDING\_FILE, binary=True)

#embedding\_dict = KeyedVectors.load\_word2vec\_format(EMBEDDING\_FILE, binary=False)

#embedding\_dict.svae\_word2vec\_format(EMBEDDING\_FILE + ".bin", binary=True)

#word2vec = KeyedVectors.load\_word2vec\_format(EMBEDDING\_FILE + ".bin", binary=True)

questions\_cols = ['question1', 'question2']

# Iterate over the questions only of both training and test datasets

for dataset in [train\_df, test\_df]:

for index, row in dataset.iterrows():

# Iterate through the text of both questions of the row

for question in questions\_cols:

q2n = [] # q2n -> question numbers representation

for word in text\_to\_word\_list(row[question]):

# Check for unwanted words

if word in stops and word not in word2vec.vocab:

continue

if word not in vocabulary:

vocabulary[word] = len(inverse\_vocabulary)

q2n.append(len(inverse\_vocabulary))

inverse\_vocabulary.append(word)

else:

q2n.append(vocabulary[word])

# Replace questions as word to question as number representation

dataset.set\_value(index, question, q2n)

embedding\_dim = 300

embeddings = 1 \* np.random.randn(len(vocabulary) + 1, embedding\_dim) # This will be the embedding matrix

embeddings[0] = 0 # So that the padding will be ignored

# Build the embedding matrix

for word, index in vocabulary.items():

if word in word2vec.vocab:

embeddings[index] = word2vec.word\_vec(word)

del word2vec

# Prepare training and validation data

max\_seq\_length = max(train\_df.question1.map(lambda x: len(x)).max(),

train\_df.question2.map(lambda x: len(x)).max(),

test\_df.question1.map(lambda x: len(x)).max(),

test\_df.question2.map(lambda x: len(x)).max())

# Split to train validation

validation\_size = 40000

training\_size = len(train\_df) - validation\_size

X = train\_df[questions\_cols]

Y = train\_df['is\_duplicate']

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, Y, test\_size=validation\_size)

# Split to dicts

X\_train = {'left': X\_train.question1, 'right': X\_train.question2}

X\_validation = {'left': X\_validation.question1, 'right': X\_validation.question2}

X\_test = {'left': test\_df.question1, 'right': test\_df.question2}

# Convert labels to their numpy representations

Y\_train = Y\_train.values

Y\_validation = Y\_validation.values

# Zero padding

for dataset, side in itertools.product([X\_train, X\_validation], ['left', 'right']):

dataset[side] = pad\_sequences(dataset[side], maxlen=max\_seq\_length)

# Make sure everything is ok

assert X\_train['left'].shape == X\_train['right'].shape

assert len(X\_train['left']) == len(Y\_train)

# Build the model

# Model variables

n\_hidden = 50

gradient\_clipping\_norm = 1.25

batch\_size = 64

n\_epoch = 25

def exponent\_neg\_manhattan\_distance(left, right):

''' Helper function for the similarity estimate of the LSTMs outputs'''

return K.exp(-K.sum(K.abs(left-right), axis=1, keepdims=True))

# The visible layer

left\_input = Input(shape=(max\_seq\_length,), dtype='int32')

right\_input = Input(shape=(max\_seq\_length,), dtype='int32')

embedding\_layer = Embedding(len(embeddings), embedding\_dim, weights=[embeddings], input\_length=max\_seq\_length, trainable=False)

# Embedded version of the inputs

encoded\_left = embedding\_layer(left\_input)

encoded\_right = embedding\_layer(right\_input)

# Since this is a siamese network, both sides share the same LSTM

shared\_lstm = LSTM(n\_hidden)

left\_output = shared\_lstm(encoded\_left)

right\_output = shared\_lstm(encoded\_right)

# Calculates the distance as defined by the MaLSTM model

malstm\_distance = Merge(mode=lambda x: exponent\_neg\_manhattan\_distance(x[0], x[1]), output\_shape=lambda x: (x[0][0], 1))([left\_output, right\_output])

# Pack it all up into a model

malstm = Model([left\_input, right\_input], [malstm\_distance])

# Adadelta optimizer, with gradient clipping by norm

optimizer = Adadelta(clipnorm=gradient\_clipping\_norm)

malstm.compile(loss='mean\_squared\_error', optimizer=optimizer, metrics=['accuracy'])

# 这个checkpoint就是记录weights bias什么的 .h5 是你要记录的名字 val\_loss 是monitor

model\_checkpoint = ModelCheckpoint('weights.h5', monitor='val\_loss', save\_best\_only=True)

# 这个记录csv

csv\_logger = CSVLogger('log.csv', append=True, separator=',')

# Start training

training\_start\_time = time()

malstm\_trained = malstm.fit([X\_train['left'], X\_train['right']], Y\_train, batch\_size=batch\_size, nb\_epoch=n\_epoch,

validation\_data=([X\_validation['left'], X\_validation['right']], Y\_validation),

callbacks=[model\_checkpoint, csv\_logger])

print("Training time finished.\n{} epochs in {}".format(n\_epoch, datetime.timedelta(seconds=time()-training\_start\_time)))

# Plotting the results

# Plot accuracy

plt.plot(malstm\_trained.history['acc'])

plt.plot(malstm\_trained.history['val\_acc'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

# Plot loss

plt.plot(malstm\_trained.history['loss'])

plt.plot(malstm\_trained.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()