In [ ]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import os
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from tqdm import tqdm
import math
from sklearn.model_selection import train_test_split
```

Setup

In [ ]:
```
train_df = pd.read_csv("../input/train.csv")
train_df, val_df = train_test_split(train_df, test_size=0.1)
```

In [ ]:
```
# embdedding setup
# Source https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
embeddings_index = {}
f = open('../input/embeddings/glove.840B.300d/glove.840B.300d.txt')
for line in tqdm(f):
    values = line.split(" ")
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

In [ ]:
```
# Convert values to embeddings
def text_to_array(text):
    empyt_emb = np.zeros(300)
    text = text[:-1].split()[:30]
    embeds = [embeddings_index.get(x, empyt_emb) for x in text]
    embeds+= [empyt_emb] * (30 - len(embeds))
    return np.array(embeds)

# train_vects = [text_to_array(X_text) for X_text in tqdm(train_df["question_text"])]
val_vects = np.array([text_to_array(X_text) for X_text in tqdm(val_df["question_text"][:3000])])
val_y = np.array(val_df["target"][:3000])
```

In [ ]:
```
# Data providers
batch_size = 128

def batch_gen(train_df):
    n_batches = math.ceil(len(train_df) / batch_size)
    while True:
        train_df = train_df.sample(frac=1.)  # Shuffle the data.
        for i in range(n_batches):
            texts = train_df.iloc[i*batch_size:(i+1)*batch_size, 1]
            text_arr = np.array([text_to_array(text) for text in texts])
            yield text_arr, np.array(train_df["target"][i*batch_size:(i+1)*batch_size])
```

Training

In [ ]:
```
from keras.models import Sequential
from keras.layers import CuDNNLSTM, Dense, Bidirectional
```

In [ ]:
```
model = Sequential()
model.add(Bidirectional(CuDNNLSTM(64, return_sequences=True),
                        input_shape=(30, 300)))
model.add(Bidirectional(CuDNNLSTM(64)))
model.add(Dense(1, activation="sigmoid"))

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

In [ ]:
```
mg = batch_gen(train_df)
model.fit_generator(mg, epochs=20,
                    steps_per_epoch=1000,
                    validation_data=(val_vects, val_y),
                    verbose=True)
```

Inference

In [ ]:
```
# prediction part
batch_size = 256
def batch_gen(test_df):
    n_batches = math.ceil(len(test_df) / batch_size)
    for i in range(n_batches):
        texts = test_df.iloc[i*batch_size:(i+1)*batch_size, 1]
        text_arr = np.array([text_to_array(text) for text in texts])
        yield text_arr

test_df = pd.read_csv("../input/test.csv")

all_preds = []
for x in tqdm(batch_gen(test_df)):
    all_preds.extend(model.predict(x).flatten())
```

In [ ]:
```
y_te = (np.array(all_preds) > 0.5).astype(np.int)

submit_df = pd.DataFrame({"qid": test_df["qid"], "prediction": y_te})
submit_df.to_csv("submission.csv", index=False)
```