Notebook Objective:

Objective of the notebook is to look at the different pretrained embeddings provided in the dataset and to see how they are useful in the model building process.

First let us import the necessary modules and read the input data.

In [ ]:
```
import os
import time
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
from sklearn import metrics

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, CuDNNGRU, Conv1D
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model
from keras import initializers, regularizers, constraints, optimizers, layers
```

In [ ]:
```
train_df = pd.read_csv("../input/train.csv")
test_df = pd.read_csv("../input/test.csv")
print("Train shape : ",train_df.shape)
print("Test shape : ",test_df.shape)
```

Next steps are as follows:

- Split the training dataset into train and val sample. Cross validation is a time consuming process and so let us do simple train val split.
- Fill up the missing values in the text column with 'na'
- Tokenize the text column and convert them to vector sequences
- Pad the sequence as needed - if the number of words in the text is greater than 'max_len' truncate them to 'max_len' or if the number of words in the text is lesser than 'max_len' add zeros for remaining values.

In [ ]:
```
## split to train and val
train_df, val_df = train_test_split(train_df, test_size=0.1, random_state=2018)

## some config values
embed_size = 300 # how big is each word vector
max_features = 50000 # how many unique words to use (i.e num rows in embedding vector)
maxlen = 100 # max number of words in a question to use

## Fill up the missing values
train_X = train_df["question_text"].fillna("_na_").values
val_X = val_df["question_text"].fillna("_na_").values
test_X = test_df["question_text"].fillna("_na_").values

## Tokenize the sentences
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(train_X))
train_X = tokenizer.texts_to_sequences(train_X)
val_X = tokenizer.texts_to_sequences(val_X)
test_X = tokenizer.texts_to_sequences(test_X)

## Pad the sentences
train_X = pad_sequences(train_X, maxlen=maxlen)
val_X = pad_sequences(val_X, maxlen=maxlen)
test_X = pad_sequences(test_X, maxlen=maxlen)

## Get the target values
train_y = train_df['target'].values
val_y = val_df['target'].values
```

Without Pretrained Embeddings:

Now that we are done with all the necessary preprocessing steps, we can first train a Bidirectional GRU model. We will not use any pre-trained word embeddings for this model and the embeddings will be learnt from scratch. Please check out the model summary for the details of the layers used.

In [ ]:
```
inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size)(inp)
x = Bidirectional(CuDNNGRU(64, return_sequences=True))(x)
x = GlobalMaxPool1D()(x)
x = Dense(16, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Train the model using train sample and monitor the metric on the valid sample. This is just a sample model running for 2 epochs. Changing the epochs, batch_size and model parameters might give us a better model.

In [ ]:
```
## Train the model
model.fit(train_X, train_y, batch_size=512, epochs=2, validation_data=(val_X, val_y))
```

Now let us get the validation sample predictions and also get the best threshold for F1 score.

In [ ]:
```
pred_noemb_val_y = model.predict([val_X], batch_size=1024, verbose=1)
for thresh in np.arange(0.1, 0.501, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(val_y, (pred_noemb_val_y>thresh).astype(int))))
```

Now let us get the test set predictions as well and save them

In [ ]:
```
pred_noemb_test_y = model.predict([test_X], batch_size=1024, verbose=1)
```

Now that our model building is done, it might be a good idea to clean up some memory before we go to the next step.

In [ ]:
```
del model, inp, x
import gc; gc.collect()
time.sleep(10)
```

So we got some baseline GRU model without pre-trained embeddings. Now let us use the provided embeddings and rebuild the model again to see the performance.

In [ ]:
```
!ls ../input/embeddings/
```

We have four different types of embeddings.

- GoogleNews-vectors-negative300 - https://code.google.com/archive/p/word2vec/
- glove.840B.300d - https://nlp.stanford.edu/projects/glove/
- paragram_300_sl999 - https://cogcomp.org/page/resource_view/106
- wiki-news-300d-1M - https://fasttext.cc/docs/en/english-vectors.html

A very good explanation for different types of embeddings are given in this kernel. Please refer the same for more details..

Glove Embeddings:

In this section, let us use the Glove embeddings and rebuild the GRU model.

In [ ]:
```
EMBEDDING_FILE = '../input/embeddings/glove.840B.300d/glove.840B.300d.txt'
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.split(" ")) for o in open(EMBEDDING_FILE))

all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
embed_size = all_embs.shape[1]

word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector

inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
x = Bidirectional(CuDNNGRU(64, return_sequences=True))(x)
x = GlobalMaxPool1D()(x)
x = Dense(16, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

In [ ]:
```
model.fit(train_X, train_y, batch_size=512, epochs=2, validation_data=(val_X, val_y))
```

In [ ]:
```
pred_glove_val_y = model.predict([val_X], batch_size=1024, verbose=1)
for thresh in np.arange(0.1, 0.501, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(val_y, (pred_glove_val_y>thresh).astype(int))))
```

Results seem to be better than the model without pretrained embeddings.

In [ ]:
```
pred_glove_test_y = model.predict([test_X], batch_size=1024, verbose=1)
```

In [ ]:
```
del word_index, embeddings_index, all_embs, embedding_matrix, model, inp, x
import gc; gc.collect()
time.sleep(10)
```

Wiki News FastText Embeddings:

Now let us use the FastText embeddings trained on Wiki News corpus in place of Glove embeddings and rebuild the model.

In [ ]:
```
EMBEDDING_FILE = '../input/embeddings/wiki-news-300d-1M/wiki-news-300d-1M.vec'
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.split(" ")) for o in open(EMBEDDING_FILE) if len(o)>100)

all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
embed_size = all_embs.shape[1]

word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector

inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
x = Bidirectional(CuDNNGRU(64, return_sequences=True))(x)
x = GlobalMaxPool1D()(x)
x = Dense(16, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [ ]:
```
model.fit(train_X, train_y, batch_size=512, epochs=2, validation_data=(val_X, val_y))
```

In [ ]:
```
pred_fasttext_val_y = model.predict([val_X], batch_size=1024, verbose=1)
for thresh in np.arange(0.1, 0.501, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(val_y, (pred_fasttext_val_y>thresh).astype(int))))
```

In [ ]:
```
pred_fasttext_test_y = model.predict([test_X], batch_size=1024, verbose=1)
```

In [ ]:
```
del word_index, embeddings_index, all_embs, embedding_matrix, model, inp, x
import gc; gc.collect()
time.sleep(10)
```

Paragram Embeddings:

In this section, we can use the paragram embeddings and build the model and make predictions.

In [ ]:
```
EMBEDDING_FILE = '../input/embeddings/paragram_300_sl999/paragram_300_sl999.txt'
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.split(" ")) for o in open(EMBEDDING_FILE, encoding="utf8", errors='ignore') if len(o)>100)

all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
embed_size = all_embs.shape[1]

word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector

inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
x = Bidirectional(CuDNNGRU(64, return_sequences=True))(x)
x = GlobalMaxPool1D()(x)
x = Dense(16, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [ ]:
```
model.fit(train_X, train_y, batch_size=512, epochs=2, validation_data=(val_X, val_y))
```

In [ ]:
```
pred_paragram_val_y = model.predict([val_X], batch_size=1024, verbose=1)
for thresh in np.arange(0.1, 0.501, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(val_y, (pred_paragram_val_y>thresh).astype(int))))
```

In [ ]:
```
pred_paragram_test_y = model.predict([test_X], batch_size=1024, verbose=1)
```

In [ ]:
```
del word_index, embeddings_index, all_embs, embedding_matrix, model, inp, x
import gc; gc.collect()
time.sleep(10)
```

Observations:

- Overall pretrained embeddings seem to give better results compared to non-pretrained model.
- The performance of the different pretrained embeddings are almost similar.

Final Blend:

Though the results of the models with different pre-trained embeddings are similar, there is a good chance that they might capture different type of information from the data. So let us do a blend of these three models by averaging their predictions.

In [ ]:
```
pred_val_y = 0.33*pred_glove_val_y + 0.33*pred_fasttext_val_y + 0.34*pred_paragram_val_y
for thresh in np.arange(0.1, 0.501, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(val_y, (pred_val_y>thresh).astype(int))))
```

The result seems to better than individual pre-trained models and so we let us create a submission file using this model blend.

In [ ]:
```
pred_test_y = 0.33*pred_glove_test_y + 0.33*pred_fasttext_test_y + 0.34*pred_paragram_test_y
pred_test_y = (pred_test_y>0.33).astype(int)
out_df = pd.DataFrame({"qid":test_df["qid"].values})
out_df['prediction'] = pred_test_y
out_df.to_csv("submission.csv", index=False)
```

References:

Thanks to the below kernels which helped me with this one.

1. https://www.kaggle.com/jhoward/improved-lstm-baseline-glove-dropout
2. https://www.kaggle.com/sbongo/do-pretrained-embeddings-give-you-the-extra-edge