# LSTM_Model_1

July 21, 2020

[2]:
```
import numpy as np
import pandas as pd

from tensorflow.keras.layers import (LSTM, Input, Embedding,
                                     Dense, Flatten, Concatenate,
                                     Dropout)
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

[3]:
```
from sklearn.model_selection import train_test_split

data = pd.read_csv('/content/drive/My Drive/colab resources/AppliedAI/Donors␣
 ↪Choose/preprocessed_data.csv')

train, test = train_test_split(data, test_size=0.2, random_state=0)
```

[4]:
```
word_counts = train['essay'].str.split().map(len)
np.percentile(word_counts, 97)
```

[4]:
```
245.0
```

97% of all essays have a word count of less than 250.

[5]:
```
MAX_NUM_WORDS = 20_000
MAX_SEQUENCE_LENGTH = 250

tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)

tokenizer.fit_on_texts(train['essay'])

train_essay_seq = tokenizer.texts_to_sequences(train['essay'])
test_essay_seq = tokenizer.texts_to_sequences(test['essay'])

train_essay_seq = pad_sequences(
    train_essay_seq, maxlen=MAX_SEQUENCE_LENGTH, dtype='uint16', padding='pre',
    truncating='pre', value=0
```

```
)
test_essay_seq = pad_sequences(
    test_essay_seq, maxlen=MAX_SEQUENCE_LENGTH, dtype='uint16', padding='pre',
    truncating='pre', value=0
)
```

[7]:
```
EMBEDDING_DIM = 50

def create_glove_embeddings(glove_path):
  embeddings_dict = {}
  with open(glove_path) as glove:
    for line in glove:
      values = line.split()
      word = values[0]
      coefs = np.asarray(values[1:], dtype='float32')
      embeddings_dict[word] = coefs

  return embeddings_dict


embeddings_dict = create_glove_embeddings(f'glove.6B.{EMBEDDING_DIM}d.txt')
```

[8]:
```
def get_embedding_matrix(embeddings_dict, tokenizer):

  word_index = tokenizer.word_index
  num_words = len(word_index) + 1

  embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))

  for word, index in word_index.items():
    embedding_vector = embeddings_dict.get(word)
    if embedding_vector is not None:
      embedding_matrix[index] = embedding_vector

  return embedding_matrix

embedding_matrix = get_embedding_matrix(embeddings_dict, tokenizer)
```

[9]:
```
word_index = tokenizer.word_index
num_words = len(word_index) + 1

essay_inp = Input(shape=MAX_SEQUENCE_LENGTH, name='essay_inp')

x = Embedding(input_dim=num_words, # vocab size
          output_dim=EMBEDDING_DIM,
          weights=[embedding_matrix],
          trainable=False,
```

```
          input_length=MAX_SEQUENCE_LENGTH,
          name='essay_emb')(essay_inp)

x = LSTM(32, name='LSTM')(x)

essay_flat = Flatten(name='essay_flat')(x)
```

[10]:
```
tokenizer = Tokenizer()

tokenizer.fit_on_texts(train['school_state'])
train_school = tokenizer.texts_to_sequences(train['school_state'])
test_school = tokenizer.texts_to_sequences(test['school_state'])
train_school = np.array(train_school)
test_school = np.array(test_school)


tokenizer.fit_on_texts(train['project_grade_category'])
train_grade = tokenizer.texts_to_sequences(train['project_grade_category'])
test_grade = tokenizer.texts_to_sequences(test['project_grade_category'])
train_grade = pad_sequences(train_grade, dtype='uint8')
test_grade = pad_sequences(test_grade, dtype='uint8')

tokenizer.fit_on_texts(train['clean_categories'])
train_cat = tokenizer.texts_to_sequences(train['clean_categories'])
test_cat = tokenizer.texts_to_sequences(test['clean_categories'])
train_cat = pad_sequences(train_cat, dtype='uint8')
test_cat = pad_sequences(test_cat, dtype='uint8')

tokenizer.fit_on_texts(train['clean_subcategories'])
train_subcat = tokenizer.texts_to_sequences(train['clean_subcategories'])
test_subcat = tokenizer.texts_to_sequences(test['clean_subcategories'])
train_subcat = pad_sequences(train_subcat, dtype='uint8')
test_subcat = pad_sequences(test_subcat, dtype='uint8')

tokenizer.fit_on_texts(train['teacher_prefix'])
train_teach = tokenizer.texts_to_sequences(train['teacher_prefix'])
test_teach = tokenizer.texts_to_sequences(test['teacher_prefix'])
train_teach = np.array(train_teach)
test_teach = np.array(test_teach)
```

[11]:
```
school_inp = Input(shape=1, name='school_inp')
school_emb = Embedding(input_dim=train_school.max() + 1, # vocab len, reserve␣
 ↪one for zero
                       output_dim=3,
                       input_length=1,
                       name='school_emb')(school_inp)
school_flat = Flatten(name='school_flat')(school_emb)
```

```
grade_inp = Input(shape=3, name='grade_inp')
grade_emb = Embedding(input_dim=train_grade.max() + 1,
                      output_dim=10,
                      input_length=3,
                      name='grade_emb')(grade_inp)
grade_flat = Flatten(name='grade_flat')(grade_emb)

cat_inp = Input(shape=5, name='cat_inp')
cat_emb = Embedding(input_dim=train_cat.max() + 1,
                    output_dim=10,
                    input_length=5,
                    name='cat_emb')(cat_inp)
cat_flat = Flatten(name='cat_flat')(cat_emb)

subcat_inp = Input(shape=5, name='subcat_inp')
subcat_emb = Embedding(input_dim=train_subcat.max() + 1,
                       output_dim=10,
                       input_length=5,
                       name='subcat_emb')(subcat_inp)
subcat_flat = Flatten(name='subcat_flat')(subcat_emb)

teach_inp = Input(shape=1, name='teach_inp')
teach_emb = Embedding(input_dim=train_teach.max() + 1,
                      output_dim=3,
                      input_length=1,
                      name='teach_emb')(teach_inp)
teach_flat = Flatten(name='teach_flat')(teach_emb)
```

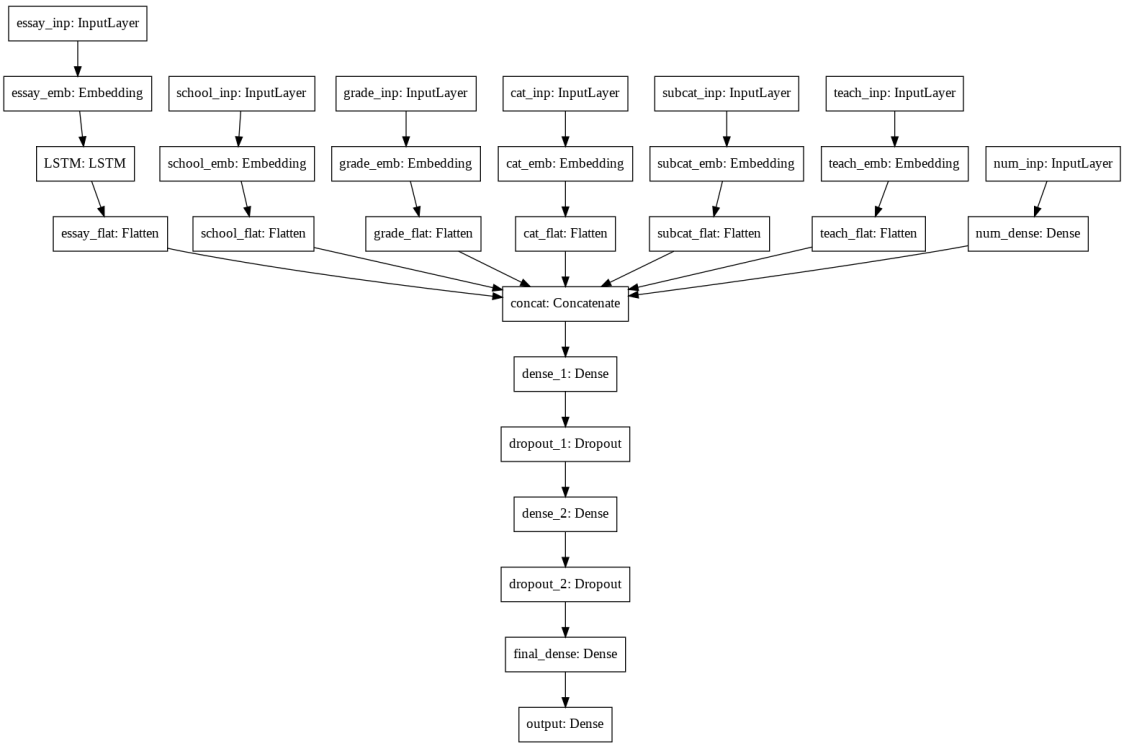
[23]:
```
train_num = pd.concat([train['teacher_number_of_previously_posted_projects'],␣
 ↪train['price']], axis=1)
test_num = pd.concat([test['teacher_number_of_previously_posted_projects'],␣
 ↪test['price']], axis=1)

num_inp = Input(shape=2, name='num_inp')
num_dense = Dense(8, activation='relu', name='num_dense')(num_inp)
```

[13]:
```
concat = Concatenate(name='concat')([essay_flat, school_flat, grade_flat,
                       cat_flat, subcat_flat, teach_flat, num_dense])

dense_1 = Dense(512, activation='relu', name='dense_1')(concat)
drop_1 = Dropout(0.33, name='dropout_1')(dense_1)

dense_2 = Dense(128, activation='relu', name='dense_2')(drop_1)
drop_2 = Dropout(0.33, name='dropout_2')(dense_2)

final_dense = Dense(32, activation='relu', name='final_dense')(drop_2)
```

```
output_layer = Dense(1, activation='sigmoid', name='output')(final_dense)
```

[14]:
```
model = Model(inputs=[essay_inp, school_inp, grade_inp,
                      cat_inp, subcat_inp, teach_inp, num_inp],
              outputs=output_layer)
plot_model(model)
```

[14]:




[28]:
```
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import (Callback, EarlyStopping,
                                        TensorBoard, ReduceLROnPlateau)

class ROCCallback(Callback):

  def __init__(self, validation_data):
    super(ROCCallback, self).__init__()
    self.validation_data = validation_data

  def on_epoch_end(self, epoch, logs={}):
    probs = self.model.predict(self.validation_data[0])
    y_true = self.validation_data[1]
    score = roc_auc_score(y_true, probs)
    logs['auc'] = score
```

[25]:
```
X_train = [train_essay_seq, train_school, train_grade,
           train_cat, train_subcat, train_teach, train_num]
y_train = train['project_is_approved']


X_test = [test_essay_seq, test_school, test_grade,
          test_cat, test_subcat, test_teach, test_num]
y_test = test['project_is_approved']
```

[29]:
```
roc_callback = ROCCallback((X_test, y_test))
early_stopping = EarlyStopping(patience=3)
tensorboard = TensorBoard()
reduce_lr = ReduceLROnPlateau()

callbacks = [
             roc_callback,
             early_stopping,
             tensorboard,
             reduce_lr,
]

model.compile(loss='binary_crossentropy', optimizer='adam')
```

[30]:
```
!rm -rf ./logs/*

history = model.fit(X_train, y_train, batch_size=32, epochs=100,
                    callbacks=callbacks, validation_data=(X_test, y_test))
```

```
Epoch 1/100
2732/2732 [==============================] - 133s 49ms/step - loss: 0.3751 -
val_loss: 0.3707 - auc: 0.7483 - lr: 0.0010
Epoch 2/100
2732/2732 [==============================] - 131s 48ms/step - loss: 0.3685 -
val_loss: 0.3707 - auc: 0.7503 - lr: 0.0010
Epoch 3/100
2732/2732 [==============================] - 131s 48ms/step - loss: 0.3650 -
val_loss: 0.3660 - auc: 0.7565 - lr: 0.0010
Epoch 4/100
2732/2732 [==============================] - 131s 48ms/step - loss: 0.3606 -
val_loss: 0.3772 - auc: 0.7598 - lr: 0.0010
Epoch 5/100
2732/2732 [==============================] - 131s 48ms/step - loss: 0.3579 -
val_loss: 0.3691 - auc: 0.7596 - lr: 0.0010
Epoch 6/100
2732/2732 [==============================] - 131s 48ms/step - loss: 0.3544 -
val_loss: 0.3690 - auc: 0.7597 - lr: 0.0010
```