LSTM_Model_3

July 21, 2020

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

from tensorflow.keras.layers import (LSTM, Input, Embedding,
                                     Dense, Flatten, Concatenate,
                                     Dropout, Conv1D)
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
```

[4]:
```
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)
```

[5]:
```
word_counts = train['essay'].str.split().map(len)
np.percentile(word_counts, 97)
```

[5]:
```
245.0
```

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

[6]:
```
NUM_WORDS = 20_000
MAX_SEQUENCE_LENGTH = 250

tokenizer = Tokenizer(num_words=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
)
```

[8]:
```
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')
```

[9]:
```
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)
```

[10]:
```
num_words = len(tokenizer.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)
```

[11]:
```
from sklearn.preprocessing import OneHotEncoder

one_hot = OneHotEncoder(sparse=False, handle_unknown='ignore')

train_cats = train.select_dtypes(object).drop('essay', axis=1)
test_cats = test.select_dtypes(object).drop('essay', axis=1)

one_hot.fit(train_cats)

train_ohe = one_hot.transform(train_cats)
test_ohe = one_hot.transform(test_cats)
```

[12]:
```
train_nums = train.select_dtypes(exclude=object).drop('project_is_approved',␣
 ↪axis=1)
test_nums = test.select_dtypes(exclude=object).drop('project_is_approved',␣
 ↪axis=1)

train_non_txt = np.c_[train_ohe, train_nums]
test_non_txt = np.c_[test_ohe, test_nums]
```

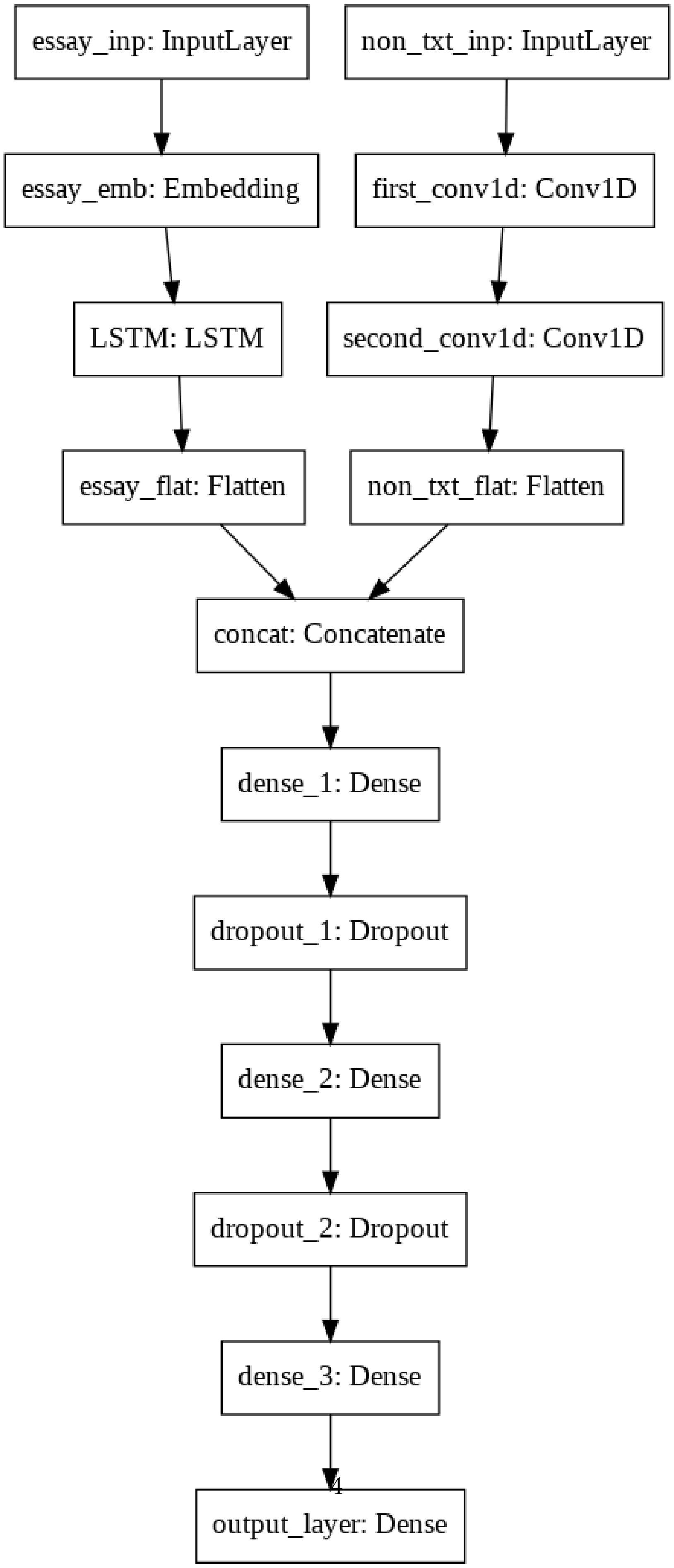
[13]:
```
train_non_txt = np.expand_dims(train_non_txt, 2)
test_non_txt = np.expand_dims(test_non_txt, 2)

input_dim = train_non_txt.shape[1:]
non_txt_inp = Input(shape=input_dim, name='non_txt_inp')
x = Conv1D(128, 3, activation='relu', name='first_conv1d')(non_txt_inp)
x = Conv1D(64, 5, activation='relu', name='second_conv1d')(x)
non_txt_flat = Flatten(name='non_txt_flat')(x)
```

[14]:
```
x = Concatenate(name='concat')([essay_flat, non_txt_flat])
x = Dense(64, activation='relu', name='dense_1')(x)
x = Dropout(0.33, name='dropout_1')(x)
x = Dense(64, activation='relu', name='dense_2')(x)
x = Dropout(0.33, name='dropout_2')(x)
x = Dense(32, activation='relu', name='dense_3')(x)
output_layer = Dense(1, activation='sigmoid', name='output_layer')(x)
```

[15]:
```
model = Model(inputs=[essay_inp, non_txt_inp], outputs=[output_layer])
model.compile(loss='binary_crossentropy', optimizer='adam')
plot_model(model)
```

[15]:


essay_inp: InputLayer
non_txt_inp: InputLayer
essay_emb: Embedding
first_conv1d: Conv1D
LSTM: LSTM
second_conv1d: Conv1D
essay_flat: Flatten
non_txt_flat: Flatten
concat: Concatenate
dense_1: Dense
dropout_1: Dropout
dense_2: Dense
dropout_2: Dropout
dense_3: Dense
output_layer: Dense

[16]:
```
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
```

[17]:
```
X_train = [train_essay_seq, train_non_txt]
y_train = train['project_is_approved']

X_test = [test_essay_seq, test_non_txt]
y_test = test['project_is_approved']
```

[18]:
```
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,
]
```

[19]:
```
!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 [==============================] - 155s 57ms/step - loss: 0.4069 -
val_loss: 0.3904 - auc: 0.7247 - lr: 0.0010
Epoch 2/100
2732/2732 [==============================] - 155s 57ms/step - loss: 0.3825 -
val_loss: 0.3773 - auc: 0.7399 - lr: 0.0010
```

```
Epoch 3/100
2732/2732 [==============================] - 155s 57ms/step - loss: 0.3753 -
val_loss: 0.3830 - auc: 0.7482 - lr: 0.0010
Epoch 4/100
2732/2732 [==============================] - 154s 56ms/step - loss: 0.3702 -
val_loss: 0.3694 - auc: 0.7539 - lr: 0.0010
Epoch 5/100
2732/2732 [==============================] - 153s 56ms/step - loss: 0.3651 -
val_loss: 0.3732 - auc: 0.7516 - lr: 0.0010
Epoch 6/100
2732/2732 [==============================] - 154s 56ms/step - loss: 0.3611 -
val_loss: 0.3679 - auc: 0.7573 - lr: 0.0010
Epoch 7/100
2732/2732 [==============================] - 153s 56ms/step - loss: 0.3580 -
val_loss: 0.3756 - auc: 0.7616 - lr: 0.0010
Epoch 8/100
2732/2732 [==============================] - 154s 56ms/step - loss: 0.3537 -
val_loss: 0.3662 - auc: 0.7623 - lr: 0.0010
Epoch 9/100
2732/2732 [==============================] - 153s 56ms/step - loss: 0.3501 -
val_loss: 0.3697 - auc: 0.7592 - lr: 0.0010
Epoch 10/100
2732/2732 [==============================] - 154s 56ms/step - loss: 0.3473 -
val_loss: 0.3757 - auc: 0.7580 - lr: 0.0010
Epoch 11/100
2732/2732 [==============================] - 153s 56ms/step - loss: 0.3433 -
val_loss: 0.3752 - auc: 0.7542 - lr: 0.0010
```