Vicente De Leon

Deep Learning Principles: Homework 3

UID: 2001014594

LSTM Recurrent Neural Networks for Classification

Dependencies

TensorFlow

```
In [ ]: #!pip install --upgrade tensorflow  Latest Tensorflow
```

```
In [ ]: !pip install -q -U tensorflow
```

```
In [ ]: import tensorflow as tf
        print(tf.__version__) # Tensorflow version
```

```
2.12.0
```

Using GPUs to run the 15 epochs faster. It was taking too long to run the model, reason why I decided to use the GPUs to run the model (Google Colab Pro).

```
In [ ]: gpus = tf.config.list_physical_devices('GPU')
        if gpus:
            print("GPU available.")
        else:
            print("GPU not available.")
```

```
GPU available.
```

TensorBoard

```
In [ ]: !pip install -q -U tensorboard==2.12.0
        #!pip install tensorboard==2.12.0
```

```
In [ ]: import datetime
        %load_ext tensorboard
```

```
In [ ]: from tensorflow.keras.callbacks import TensorBoard
```

Pandas, Numpy, Keras

```
In [ ]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder
        from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.preprocessing.sequence import pad_sequences
        from tensorflow.keras.models import Sequential
        from keras.utils.vis_utils import plot_model
        from keras.layers import LSTM
        from keras.layers import Embedding
        from keras.layers import Dropout
        from tensorflow.keras.layers import Dense
        from tensorflow.keras.optimizers import Adam
```

Exploring the IMDB Data

Data came from Kaggle "IMDB Dataset of 50k Movie Reviews" for LSTM Binary Classification purposes.

Kaggle: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews?resource=download

```
In [ ]: df = pd.read_csv('IMDB Dataset.csv')
        df.head()
```

Out[ ]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Summary statistics

```
In [ ]: df.describe()
```

Out[ ]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

Checking for missing values

```
In [ ]: df.isnull().sum()
```

```
Out[ ]: review       0
        sentiment    0
        dtype: int64
```

Preprocessing

`Label Encoding` is used to handle categorical variables (column "sentiment" -> positive vs negative). Using this technique will allow me to convert (encode) the positive and negative sentiment into class 1 and class 0. This is for binary classification purposes. I am also splitting the data using SKlearn `train_test_split()` by setting the test size to 0.2 and random state to 42. These are the numbers I used for almost all the train_test_split steps last semester. I' m also creating a validation set, so its loss can be displayed in TensorBoard along the training loss.

```
In [ ]: X = df['review']
        y = df['sentiment']

        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(y)
```

```
In [ ]: # This is how I used to split data for my APML CNN Model test size 20% and random state 42.
        # Also, a validation set was implemented.
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
In [ ]: X_train.shape
Out[ ]: (32000,)
```

```
In [ ]: X_test.shape
Out[ ]: (10000,)
```

I am going to be using TensorFlow `Tokenizer()` to pre-process the text corpus by tokenizing text into words.

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

By default, all punctuation is removed, turning the texts into space-separated sequences of words (words may include the ' character). These sequences are then split into lists of tokens. They will then be indexed or vectorized.

0 is a reserved index that won't be assigned to any word.

Reference: https://dev.to/balapriya/tokenization-and-sequencing-in-tensorflow-3p0n

Back when I was taking "Intro to NLP" for Data Science with professor Olga Scrivner, I was using the below codes to normalize and preprocess data (course material) to create a Streamlit Sentiment Analyzer using Spotify data. However, it seems TensorFlow Tokenizer() seems to be more powerful, plus it goes hand to hand with the tutorial I am currently using.

```
######################### functions clean text

stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    tokens = nltk.word_tokenize(doc)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

```
df = pd.read_csv(csv_file)
df[df.Review.str.strip() == ''].shape[0]
norm_corpus = normalize_corpus(df['Review'])
df['Clean Review'] = norm_corpus
df = df[['Review', 'Clean Review']]

df.replace(r'^(\s?)+$', np.nan, regex=True)
df.dropna().reset_index(drop=True)

df['Clean Review'] = norm_corpus
df = df[['Review', 'Clean Review']]
st.write(df.head())
```

`OOV` -> Out of Vocab will replace any unkown words with this special token.

limiting vocabulary size to 10000 in the dataset. Only the top num_words will be considered.

```
In [ ]: tokenizer = Tokenizer(num_words=10000,oov_token='<OOV>')
        tokenizer.fit_on_texts(X_train)
```

`word_index` -> Used to see how tokens have been created and the indices assigned to words.

```
In [ ]: word_index = tokenizer.word_index
        print(word_index)
```

The Tokenizer is then fit on the training data only (X_train) to learn the vocabulary. Later, the texts_to_sequences method is applied separately to the training and testing sets to convert the text into sequences. Finally, the sequences are padded to a consistent length using pad_sequences.

`tokenizer.texts_to_sequences()` will be used to convert the text into sequences. Padding will also be implemented to the sequence to same length using `pad_sequences`.

This is a crucial step of preprocessing because as we know, machines can't understand text data as we do. In order to make the machine understand what we want is to convert the data into integers (sequence of numbers). Also, padding = 'post' will help us preserve the original order of the words.

```
In [ ]: X_train_sequences = tokenizer.texts_to_sequences(X_train)
        X_valid_sequences = tokenizer.texts_to_sequences(X_valid)
        X_test_sequences = tokenizer.texts_to_sequences(X_test)

        max_length = 250  # , 150, 500
        X_train_padded = pad_sequences(X_train_sequences, maxlen = max_length, padding = 'post')
        X_val_padded = pad_sequences(X_valid_sequences, maxlen = max_length, padding = 'post')
        X_test_padded = pad_sequences(X_test_sequences, maxlen = max_length, padding = 'post')
```

LSTM Model

This model was taken from tutorial Google Colab tutorial "IMDB Movie review.ipynb". Extra ideas were taken from Applied Machine Learning course, Introduction to NLP, and internet sources which helped me complete the model regarding regularization techniques, TensorBoard, model structure etc. Parameters like max_length, epochs, dropout numbers were randomly chosen after many experiments. Why regularization techniques using L2 and dropout? My model was suffering from cases of very low accuracy scores, high training losses and low validation losses. My model was mostly overfitting and sometimes underfitting. After playing with the max_length, epochs, and integers within the dropout I managed to get the highest accuracy score and best validation/training loss behavior.

Validation and Training behavior can be seeing using the TensorBoard dashboard below. Other options like histograms, graphs etc. can be selected within the dashboard. Very useful tool to see real time experimentation. I got the idea from Applied Machine Learning course.

Colab tutotial: https://colab.research.google.com/github/dipanjannC/Deep-Learning/blob/master/IMDB_Movie_review.ipynb#scrollTo=L0y6pKlrHfHS

- The tutorial is following Keras IMDB movie dataset and I am following the Kaggle IMDB Movie 50k Reviews.
- LSTM Layer is known as the Long Short Term Memory layer (type of RNN model). LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better. This type of model is extremely useful in memorizing important information. The model will be able to find out the actual meaning in input string and will give the most accurate output class.

Training and TensorBoard

```
In [ ]: from tensorflow.keras.regularizers import l2
```

Extra Information:

- The `embedding layer` has weights that are learned. This layer is responsable for converting the sequences (input) into embedding vectors and learns the word embedding during training.
- `Sigmoid Activation Function` Will always returns a value between 0 and 1.

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

- `Binary Cross Entropy` Computes the cross-entropy loss between true labels and predicted labels.

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

```
In [ ]: vocab_size = len(tokenizer.word_index) + 1
        embedding_dim = 32
        input_length = max_length
        batch = 32 # Keras batch size default "APML"
        epochs = 15 # 10

        with tf.device('/GPU:0'):  # using GPU for faster run
            model = Sequential()
            model.add(Embedding(vocab_size, embedding_dim, input_length = max_length))
            model.add(Dropout(0.5)) #5, 5, 5, 5, 5, 3
            model.add(LSTM(100, kernel_regularizer=l2(0.01)))
            model.add(Dropout(0.1)) #1, 15, 45, 5, 4, 4
            model.add(Dense(1, activation = 'sigmoid'))
            model.compile(loss='binary_crossentropy', # Loss also used within my CNN model from last semester
                          optimizer = 'adam', # Adam(learning_rate=0.0001)
                          metrics=['accuracy'])

            log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
            tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

            with tf.device('/GPU:0'):
              history = model.fit(X_train_padded, y_train,
                                  validation_data=(X_val_padded, y_valid),
                                  epochs=epochs,
                                  batch_size=batch,
                                  callbacks=[tensorboard_callback])
```

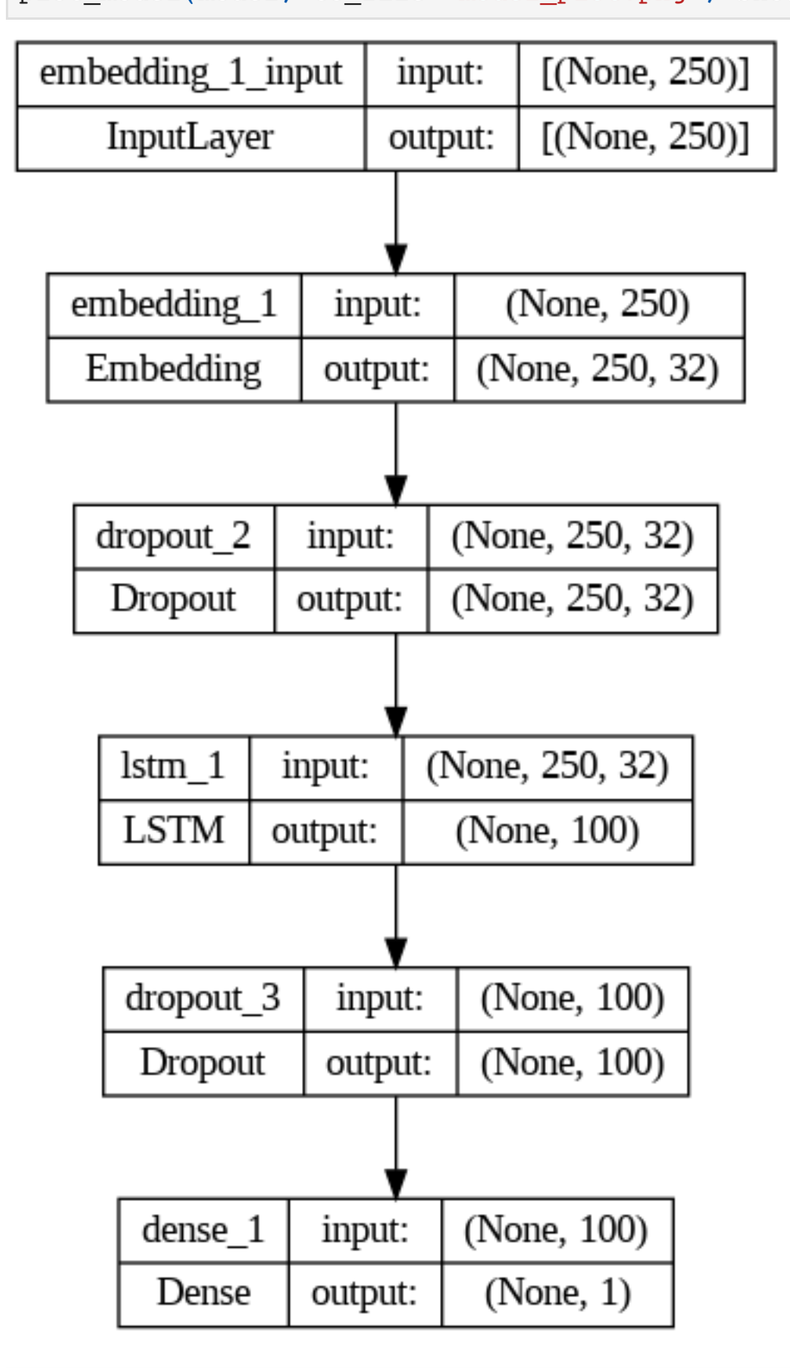
```
Epoch 1/15
1000/1000 [==============================] - 65s 63ms/step - loss: 0.7134 - accuracy: 0.5090 - val_loss: 0.6914 - val_accuracy: 0.5310
Epoch 2/15
1000/1000 [==============================] - 21s 21ms/step - loss: 0.6583 - accuracy: 0.6226 - val_loss: 0.6443 - val_accuracy: 0.6755
Epoch 3/15
1000/1000 [==============================] - 16s 16ms/step - loss: 0.6264 - accuracy: 0.6681 - val_loss: 0.5331 - val_accuracy: 0.7830
Epoch 4/15
1000/1000 [==============================] - 13s 13ms/step - loss: 0.6560 - accuracy: 0.6180 - val_loss: 0.6617 - val_accuracy: 0.6201
Epoch 5/15
1000/1000 [==============================] - 14s 14ms/step - loss: 0.6255 - accuracy: 0.6800 - val_loss: 0.6663 - val_accuracy: 0.5761
Epoch 6/15
1000/1000 [==============================] - 15s 15ms/step - loss: 0.5366 - accuracy: 0.7643 - val_loss: 0.6165 - val_accuracy: 0.7159
Epoch 7/15
1000/1000 [==============================] - 13s 13ms/step - loss: 0.5711 - accuracy: 0.7351 - val_loss: 0.5059 - val_accuracy: 0.8045
Epoch 8/15
1000/1000 [==============================] - 13s 13ms/step - loss: 0.6907 - accuracy: 0.5871 - val_loss: 0.6997 - val_accuracy: 0.5640
Epoch 9/15
1000/1000 [==============================] - 12s 12ms/step - loss: 0.6487 - accuracy: 0.6024 - val_loss: 0.6551 - val_accuracy: 0.5779
Epoch 10/15
1000/1000 [==============================] - 12s 12ms/step - loss: 0.6332 - accuracy: 0.6280 - val_loss: 0.4724 - val_accuracy: 0.8176
Epoch 11/15
1000/1000 [==============================] - 12s 12ms/step - loss: 0.5285 - accuracy: 0.7505 - val_loss: 0.4393 - val_accuracy: 0.8340
Epoch 12/15
1000/1000 [==============================] - 12s 12ms/step - loss: 0.6366 - accuracy: 0.6592 - val_loss: 0.4354 - val_accuracy: 0.8273
Epoch 13/15
1000/1000 [==============================] - 12s 12ms/step - loss: 0.3863 - accuracy: 0.8404 - val_loss: 0.3578 - val_accuracy: 0.8547
Epoch 14/15
1000/1000 [==============================] - 12s 12ms/step - loss: 0.3457 - accuracy: 0.8637 - val_loss: 0.3578 - val_accuracy: 0.8529
Epoch 15/15
1000/1000 [==============================] - 11s 11ms/step - loss: 0.3032 - accuracy: 0.8812 - val_loss: 0.3252 - val_accuracy: 0.8736
```

```
In [ ]: model.summary()
```

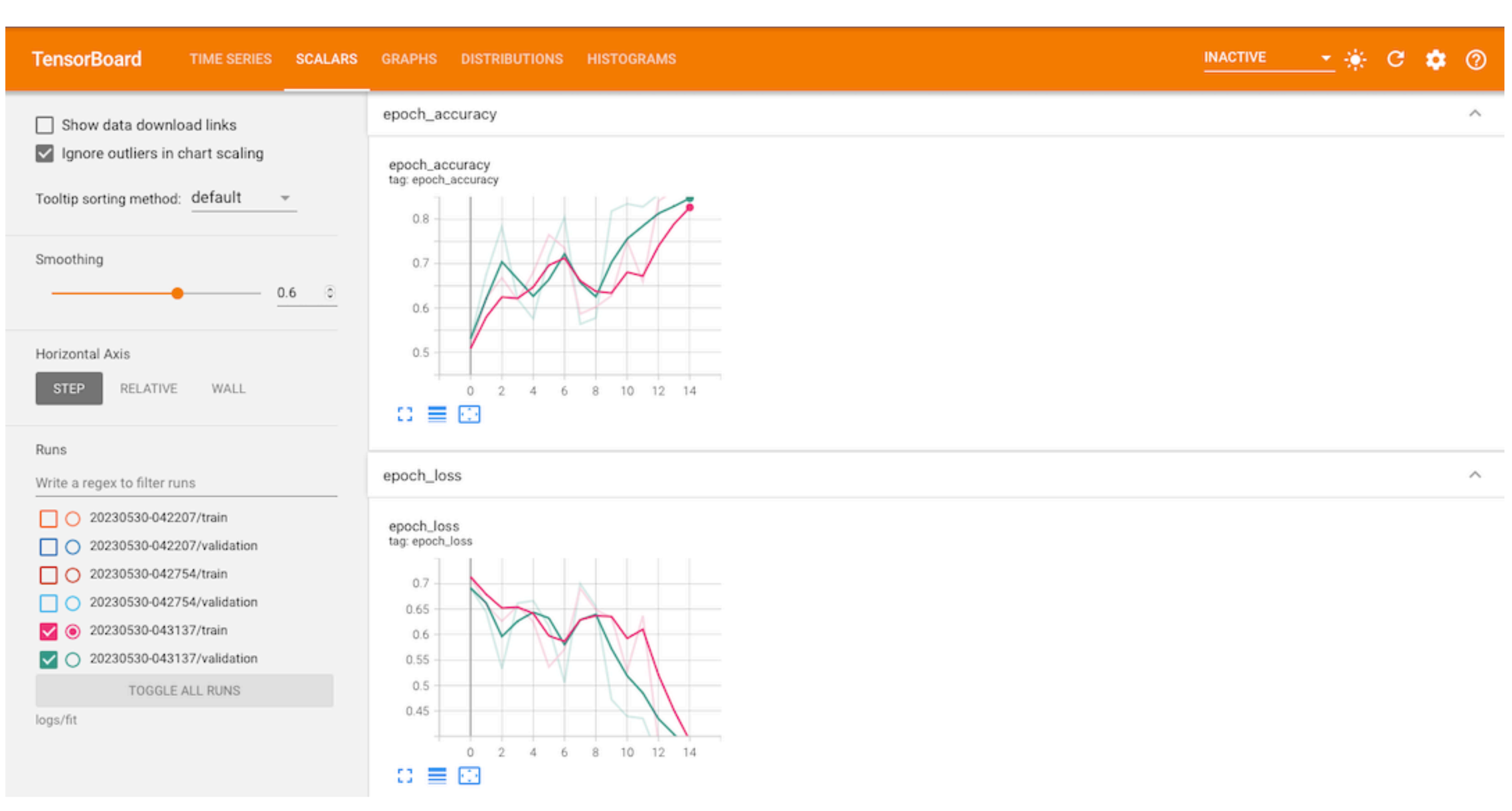
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 250, 32)           3241664

 dropout_2 (Dropout)         (None, 250, 32)           0

 lstm_1 (LSTM)               (None, 100)               53200

 dropout_3 (Dropout)         (None, 100)               0

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 3,294,965
Trainable params: 3,294,965
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Out[ ]:




```
In [ ]: %tensorboard --logdir logs/fit
```

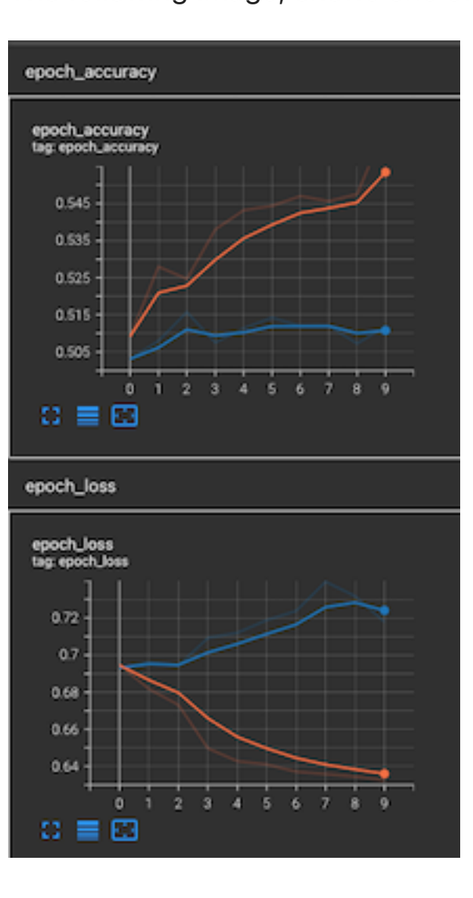



```
In [ ]: min_loss = min(history.history['loss']) # APML Homework
        min_val_loss = min(history.history['val_loss'])
        max_tr_acc = max(history.history['accuracy'])
        max_val_acc = max(history.history['val_accuracy'])

        print(f'The lowest training loss after 15 epochs is: {min_loss}')
        print(f'The lowest val loss after 15 epochs is: {min_val_loss}')
        print('\n')
        print(f'The hightest training accuracy after 15 epochs is: {max_tr_acc}')
        print(f'The hightest validation accuracy after 15 epochs is: {max_val_acc}')
```

```
The lowest training loss after 15 epochs is: 0.30322331190109253
The lowest val loss after 15 epochs is: 0.325209379196167


The hightest training accuracy after 15 epochs is: 0.8812187314033508
The hightest validation accuracy after 15 epochs is: 0.8736249804496765
```

As mentioned before, our highest train accuracy is of 88% after experimenting with different cases, integers, and techniques. Since this is my first time working with LSTM Model, I wanted it to try and keep it simple regarding accuracy score and epochs. Many tutorials found on internet has pretty good accuracy scores, however many models were extremely overfitting.

The following image, shows one of my first experiments where my model was sufffering from underfitting:



Evaluation and Testing

```
In [ ]: # Evaluation
        loss, test_accuracy = model.evaluate(X_test_padded, y_test)

        print(f'The loss for LSTM Model is: {loss}')
        print(f'The test Accuracy for LSTM Model is: {test_accuracy}')
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.3317 - accuracy: 0.8675
The loss for LSTM Model is: 0.33174842596054077
The test Accuracy for LSTM Model is: 0.8675000071525574
```

Let's print 5 random prediction to tets our model.

```
In [ ]: X_test_arr = np.array(X_test)

        # Testing
        predictions = model.predict(X_test_padded)

        final_pred = decoded_predictions = np.where(predictions > 0.5, 'positive', 'negative')

        # Random predictions to test the model
        print('Random Predictions:')
        random = np.random.choice(len(X_test), size=5, replace=False)
        for i in random:
            print(f'Review: {X_test_arr[i]}')
            print(f'Prediction: {final_pred[i]}')
            print()
```

```
313/313 [==============================] - 1s 5ms/step
Random Predictions:
Review: Like the previous commentator on this film, I too found myself in tears at times during this movie. Sometimes one wonders how a film of such awe-inspiring awfulness comes into existence. From the first moment when our protagonist wakes up in his New York apartment from a dream of subway trains intercut with galloping horses (what Irish emigrant hasn't had that one), its clear we are in trouble. And it doesn't get much better.<br /><br />Whisked back to 1950s Ireland, we enter a world where everybody speaks without intonation, and exclusively on the topic of the Irish Civil War. Schoolchildren go to school to learn about the Civil War. The drinkers in the pub divide themselves according to their Civil War allegiances. Remembrances are carried out for those who died in the Civil War. The town is divided between those who believe we should remember and those who want to forget...the Civil War. Every glance and conversation is dripping with meaning that traces back to the Civil War.<br /><br />The blurb on the videocover of Broken Harvest suggests that the film is a parable of the troubles in modern Ireland. The only parallel which strikes me is that in present day Dublin conversation is indeed dominated by one topic: house prices. If its intention is to offer some sort of insight into Ireland's obsession with its past, it fails miserably. It is striking how few Irish films have dealt with the Irish civil war and its legacies. However it will take a film of a great deal more subtlety and intelligence than this one to tell us anything about the lasting effects of such a traumatic event on the nation's psyche.<br /><br />For those American viewers who have suggested the film evokes the atmosphere of 1950s Ireland: it doesn't. 1950s Ireland was a horrible, poverty stricken pit of sexual repression and misery from which young people fled in their droves. However there was more than one topic of conversation.
Prediction: ['positive']

Review: I am a huge fan of Northern Exposure. Men In Trees is a complete knock-off of Northern Exposure. There's the city-folk from New York stuck in a remote backwoods town (Marin Frist / Joel Fleishman). She immediately doesn't hit it off with a local (Jack Slattery / Maggie O'Connell). The town only has one pilot who is not often available. Ther is only one radio show. And the entire show is about quirky people with sincere and odd relationships. Too many parallels. Just like Northern Exposure except one obvious demographic has been altered in each character. I'm bored. Give me something new to think about.<br /><br />On the upside, the writing is pretty good. But if this was a school project, it would certainly have earned the stamp of plagiarism. <br /><br />Perhaps people who love this show were sheltered from the dedicated creativity and hard labors of Joshua Brand and John Falsey.
Prediction: ['negative']

Review: Truly awful. Obviously an attempt to cash in on the Star Wars craze, but there's no excuse for this insipid piece of garbage. The storyline gets lost before the middle, the characters are forgettable and the kid is such a non-talent that all he does is pose for the camera. An alien planet they land on is actually the Naval stockyards in Long Beach, California. <br /><br />This movie actually made me physically ill watching it.<br /><br />Roger Corman has made some good, some bad movies in his career, but there's no excuse for this one. <br /><br />0/10 - IMDb won't let you score a zero.
Prediction: ['negative']

Review: What can I say? After having read Herbert's books and loving Lynch's movie version, I was extremely disappointed. I felt I was watching a reject version of Buck Rogers. The sets looked like left overs from a Star Wars TV special! I felt the acting was a bit amateurish by most. The costumes were garish and over done which gave it a '60s Flash Gordon, pulp feel. The worms! They're supposed to be Sand Worms and yet they appeared to "big stalagtites" with a mouth at the blunt end. The effects in general were pretty second rate. I won't even start about the disgraceful "Navigator" effect.<br /><br />This so-called "Frank Herbert's Dune" wasn't even faithfull to his books! It should have been called "Frank Herbert's Dune - For Dummies". Key plot elements were left out, names were changed and the entire "feel" of the story was "sanitised". I didn't even recognise the Harkonnens! In fact most of the characters appeared nothing like Herbert's descriptions had depicted them. I'm starting to get upset just remembering what a tragedy it was. I'm glad I couldn't stomach the second installment....
Prediction: ['negative']

Review: when i saw commercials for this i was thinking "NO WHAT HAS NICK AT NITE DONE!" because it was taking up "fresh prince" slots. well, i still love the fresh prince. but george lopez is a surprisingly good show. i love how not-stereotypical benny is. carmen is a pretty good character, its really funny to see how stupid and overemotional she can be sometimes. i feel bad for the guy who plays max, he looks much younger then he actually is! but max is a fun character, and acted well. and yeah, angie is a little stereotypical, but she has her funny moments. ha ha george does have a big head! nah but he can be really good too. funny show! it definitely should be on more often then home improvement.
Prediction: ['positive']
```

References

- Kaggle Data: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews?resource=download
- GPUs: https://www.tensorflow.org/guide/gpu
- Colab: https://colab.research.google.com/github/dipanjannC/Deep-Learning/blob/master/IMDB_Movie_review.ipynb#scrollTo=L0y6pKlrHfHS
- Label Encoding: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/
- Tokenizer: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
- Tokenizer, Sequences, Padding: https://dev.to/balapriya/tokenization-and-sequencing-in-tensorflow-3p0n
- Tokenizer: https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenization-text-data-prep.html
- num_words: https://stackoverflow.com/questions/53525994/how-to-find-num-words-or-vocabulary-size-of-keras-tokenizer-when-one-is-not-as
- OOV: https://www.kaggle.com/code/hamishdickson/using-keras-oov-tokens
- LSTM Layer: https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/
- Embedding: https://www.mathworks.com/help/textanalytics/ref/nnet.cnn.layer.wordembeddinglayer.html#
- Embedding: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
- Binary Cross Entropy Loss: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy
- Dropout: https://medium.datadriveninvestor.com/deep-learning-lstm-for-sentiment-analysis-in-tensorflow-with-keras-api-92e62cde7626
- TensorBoard with Keras Model.fit(): https://www.tensorflow.org/tensorboard/get_started
- L2 Penalty: https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L2
- L2 Penalty: https://keras.io/api/layers/regularizers/
- Binary Cross Entropy: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy
- Test score and test accuracy: https://stackoverflow.com/questions/43589842/test-score-vs-test-accuracy-when-evaluating-model-using-keras
- Other reference model: https://www.kaggle.com/code/rivals16/imdb-movie-review-nlp-lstm-with-dnn