DeepLearning.AI Tensorflow Developer

Natural Language Processing in TensorFlow: Week 2

Word Embedding

Words and similar words are clustered as vectors in multi-dimensional space. Movie review word embedding website: http://projector.tensorflow.org

Tensorflow data services provide dataset of IMDB movie reviews.

```
import tensorflow_datasets as tfds

imdb, info = tfds.load("imdb_reviews",with_info=True,as_supervised=True)

single_example = list(imdb['train'].take(1))[0]

print(single_example) // review and label tuple
```

Split train and test dataset reviews and labels from IMDB movie review. Create TextVectorization with maxtoken parameter set 10000 to discard low frequency words and then apply pre-padding with pad_sequences method on ragged sequences. Then zip sequences and labels of train and test dataset and apply caching, prefetching, shuffling and batching.

```
train_dataset, test_dataset = imdb['train'], imdb['test']

vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=VOCAB_SIZE)

train_reviews = train_dataset.map(lambda review, label: review)

train_labels = train_dataset.map(lambda review, label: label)

vectorize_layer.adapt(train_reviews)


def padding_func(sequences):

  sequences = sequences.ragged_batch(batch_size=sequences.cardinality())

  sequences = sequences.get_single_element()

  padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=MAX_LENGTH,
truncating=TRUNC_TYPE, padding=PADDING_TYPE)

 padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

 return padded_sequences


train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
train_dataset_vectorized = tf.data.Dataset.zip(train_sequences,train_labels)
train_dataset_final = (train_dataset_vectorized
                .cache()
                .shuffle(SHUFFLE_BUFFER_SIZE)
                .prefetch(PREFETCH_BUFFER_SIZE)
                .batch(BATCH_SIZE)
                )
```

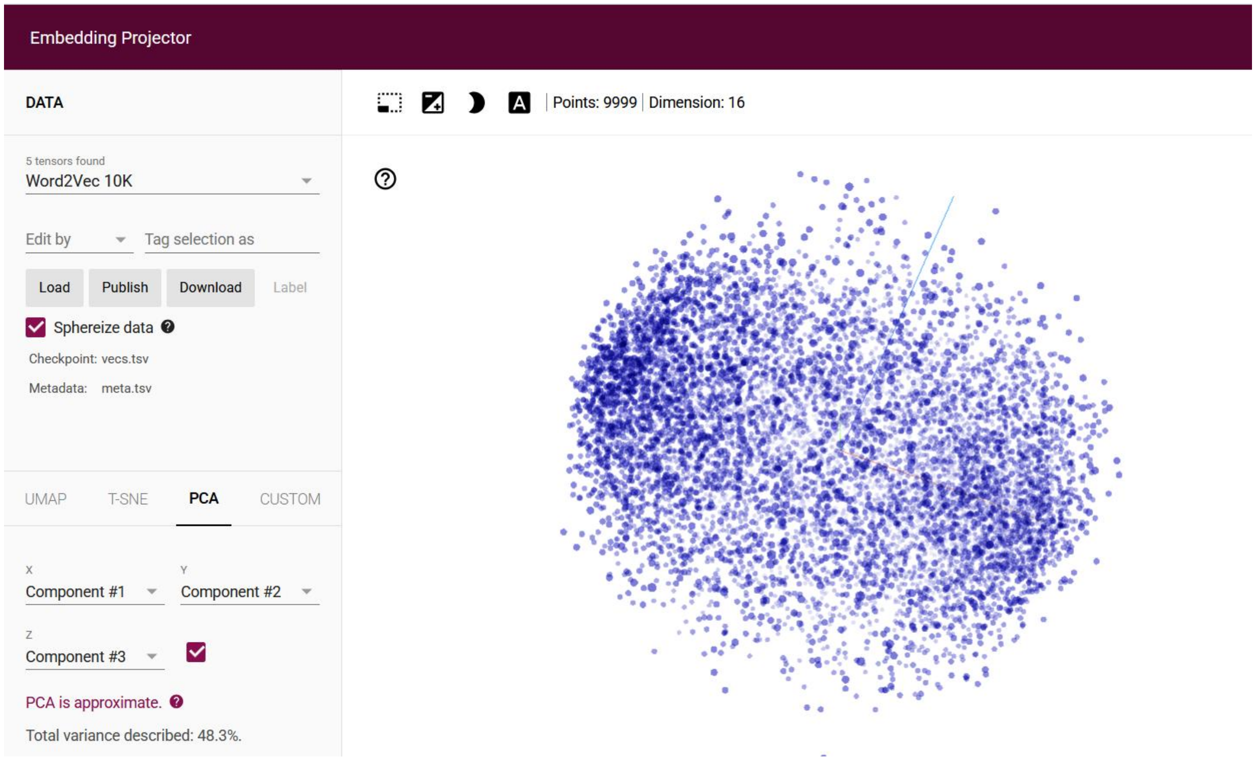
In word embedding, words are represented as vectors in higher dimension. Similar words with similar sentiments have similar embedding vectors. In sentiment classification model the input layer is the tensorflow Embedding layer, followed by Flatten and Dense layer. We can also use GlobalAveragePooling1D layer instead of Flatten layer which makes it train faster.

```
model = tf.keras.Sequential([
   tf.keras.Input(shape=(MAX_LENGTH,)),
   tf.keras.layers.Embedding(VOCAB_SIZE, EMBEDDING_DIM),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(6, activation='relu'),
   tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(train_dataset_final, epochs=NUM_EPOCHS, validation_data=test_dataset_final)
```

Visualize Embeddings of words based on index of vocabulary.

```
embedding_layer = model.layers[0]
embedding_weights = embedding_layer.get_weights()[0]
print(embedding_weights.shape)
```

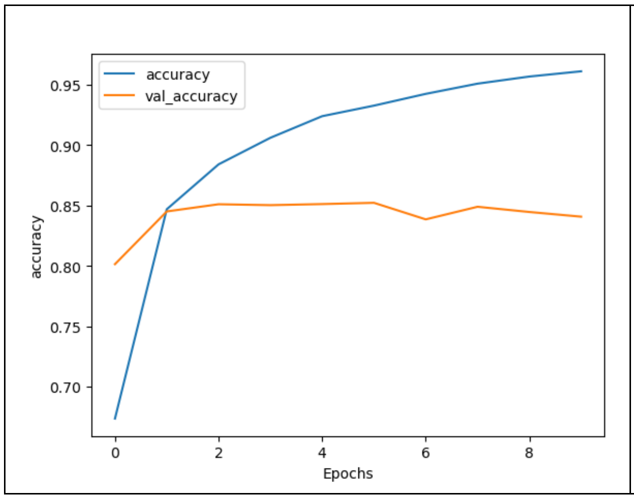
Store weights and words in vecs.tsv and meta.tsv file and load in tensorflow projector website.

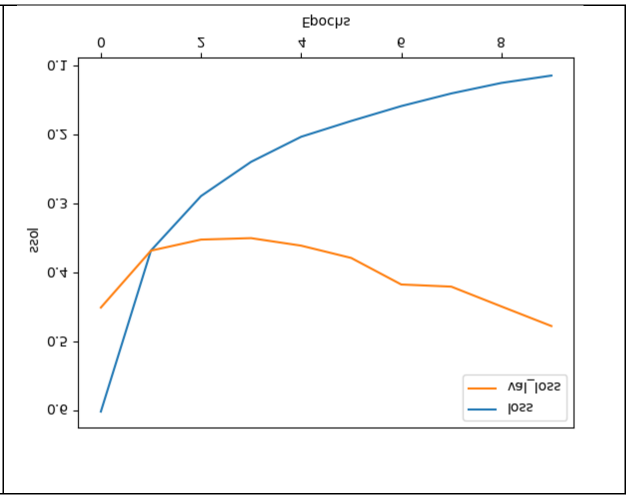


Create embedding for sarcasm dataset with 20K vocabulary size. Train model and plot the loss function for epochs. Incase of increased loss for validation set and achieve higher accuracy we need to change different hyper-parameters (token length, vocabulary size, embedding dim).

```
def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()


plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```





Sub-word Tokenization

Sub-word tokenization not only tokenize individual word but also tokenize part of the word to understand the meaning of the word sequences. Keras_nlp library is used to perform sub-word tokenization.

Generate sub-word vocabulary using compute_word_piece_vocabulary method of keras_nlp. Then create sub-word tokenizer using the generated vocabulary. Sub-word tokenization sequence has more token than the original string.

```
keras_nlp.tokenizers.compute_word_piece_vocabulary(
    train_reviews,
    vocabulary_size=8000,
    reserved_tokens=["[PAD]", "[UNK]"],
    vocabulary_output_file='imdb_vocab_subwords.txt')


subword_tokenizer =
keras_nlp.tokenizers.WordPieceTokenizer(vocabulary='./imdb_vocab_subwords.txt')


model = tf.keras.Sequential([
    tf.keras.Input(shape=(MAX_LENGTH,)),
    tf.keras.layers.Embedding(subword_tokenizer.vocabulary_size(), EMBEDDING_DIM),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```