Research Assignment 3:

Artificial Intelligence and Deep Learning

Tyrone Brown

Abstract

Chatbots are a helpful application in customer service as it frees up humans to deal with specific and/or top priority issues. One way a chatbot can help is to automatically classify the customer's message/messages to determine the sentiment. In this research we explore the use of Dense Neural Networks and One-Dimensional Convolutional Neural Networks in building text classification models using IMDb film reviews for training and testing. As expected, CNNs performed the best with their ability to group terms together and detect patterns, however overfitting is an issue that needs to be monitored with the data available. Recurrent Neural Networks were not deeply explored in this research. The best model achieved an accuracy of 89.51% after 72 seconds of training time, however another CNN model achieved 89.37% in just 28 seconds.

Introduction

Convolutional Neural Networks are commonly used in image recognition as they are capable to mapping spatial features rather than only considering a list of individual pixels. With the same idea in mind, CNNs can also be used for text recognition. Similarly, the way that we speak can be considered spatial across a 1-dimensional plane. Taking the context of words into account rather than just individual words is where CNN improves on simpler bag-of-words techniques using on Dense Neural Networks.

We have been tasked with developing a chatbot to assist customer service representatives. The purpose of this research is to compare DNN text classification models, which treat each word (or embedding) as independent features, and CNN models which take neighboring words (context) into account while looking for patterns in text sequences.

Literature Review

A similar document classification study was performed, and results published to Towards Data Science (Holtes, 2019). The three primary models that were compared were CNN, Dense and Logistic Regression. Surprisingly, the results favored the use of the simpler Logistic method as it achieved accuracy close to that of a CNN, but exponentially faster (pre-processing and training). The researcher hypothesizes that this may be due to the long text nature of the documents that are being classified, as the sequence may matter less in distinguishing the overall meaning. Instead, the mere presence of certain words seemed to provide more predictive power.

Methodology

Methodology Overview

For this research, Dense Neural Net models and Convolutional Neural Net models were analyzed and compared with varying architectures for each. As we saw with image classification in prior research, a CNN provides the advantage to extracting features from spatial data, or in our case text sequence. On the other hand, a DNN will consider the presence (or count, importance, etc.) of each word independently.

Implementation and Programming

Figure 1 shows the libraries that were used in performing all experimentation steps. To load in the IMDb data, 300 was chosen as the maximum document length and the top 26 words were skipped using the Keras load_data() function. Since the training and test data are split in half, both sets were combined to create one full 38,501 review dataset (full set is 50,000, however Keras removes testing data that exceed maximum length). From here, these were split into training and test sets of 80% and 20%, respectively.

The final CNN model's implementation requires the use of TensorFlow/Keras to build and compile the model. The layers used within the model are Embedding(), Conv1D(), MaxPooling1D(), Dropout() and Flatten().

Data Preparation, Exploration and Visualization

The dataset used for training and testing the models used in this research is the IMDb movie review dataset that comes with and was obtained from TensorFlow's library. The data loads into a training set and testing set, consisting of 25,000 film reviews each. Each film consists of a label indicating whether the review's sentiment is negative (0) or positive (1). The subset that was used in this experimentation includes 19,346 negative reviews and 19,155 positive reviews.

Results

Key Observations

Sparse Vectors versus Word Embedding. Using fully connected Dense Neural Networks, document inputs as sparse vectors were considered again implementing word embedding as the initial layer. DNN models using sparse vectors representing term presence outperformed equivalent models using word embedding. The best sparse DNN achieved an accuracy of 88.98%, one of the better overall models in this research. No word embedded DNN model performed at the top end of the results, and in fact one model fell in the bottom 3 of 187 models.

DNN versus CNN. Implementing 1-dimensional convolutional neural networks yielded the top performing models of all analyzed. When considering like models in terms of dense hidden layers, epochs, batch sizes and dropout rate, the best 1D CNN model achieved an accuracy off 89.12%. This model uses two Conv/Max Pool layers with 20% dropout layers after each, and 1 ReLu hidden layer of size 16 between flattening and the Softmax output. The embedding layer at the top of the model creates embeddings of 100 dimensions.

Early Stopping. Severe overfitting was observed with all the DNN and CNN models observed, so the best CNN model was duplicated with fewer learning epochs. This led to a slight improvement to 89.38%. However, the model was trained in less than 1/3 of the time as the baseline 1D CNN model.

Word Embedding Dimensions. In addition to the implementation of Early Stopping above, increasing the number of dimensions of the word embeddings to 250 increased accuracy slightly to 89.40% while adding on training time.

Word Embedding / Dropout Increase. In addition to the implementation of Early Stopping above, increasing the number of dimensions of the word embeddings to 300 and an additional dropout layer in the fully connected portion of the model increased accuracy slightly to 89.51%. The dropout layer was added to further help with overfitting, however 3 epochs still seems optimal.

Experiment Results

190 models were evaluated as a part of this research. See Figure 2 for all the top 25 results.

DNN | Sparse Term Vectors. This experiment consisted of a DNN which takes 10,000 columns representing our vocabulary. Each document is represented by terms of the vocabulary that are present in the review. For both 1 hidden layer and 2 hidden layers, the best models consisted of layer sizes of 8 and a 20% dropout layer, resulting in accuracies of 88.87% and 88.98% respectively.

DNN | Word Embedding. Like the DNN models using sparse term vectors, each version of the DNN with word embeddings performed best when featuring hidden layers with 8 nodes and a 20% dropout layer. However, all of these models performed poorly compared to other architectures.

CNN. This experiment consisted of a variety of CNN models with different combinations of Conv/Max Pool Layer counts, number of filters, hidden layer counts, hidden layer sizes and dropout rates. Of all of these models, those with 2 Conv/Max Pool layers with 20% dropout layers were the top performers. The top overall (50 embedding dimensions, 64 filters, 2 Conv/Max Pool layers, 20% dropout) was used as the baseline for later CNN experiments. Evidence of early overfitting can be seen in Figure 3.

RNN. Simple Recurrent Neural Networks were also briefly considered and will be the focus of future research (in addition to LSTM). Compared to both our DNN and CNN models, the 3 RNN models built performed the worst of all 190 models (excluding 2 CNNs that seemed to be anomalies with 50% accuracy). RNN models ranged from 82.09% to 78.56% with RNN unit counts from 1-4.

Conclusions

After evaluating all models of the impacts of various structures/parameters, the recommendation on our chatbot application would be to implement a 1D CNN model that takes word embedding vectors of 50 dimensions. The network should utilize 2 Conv/Max Pooling layers, 64 filters, 1 hidden layer with 16 nodes between flattening and the output layer, and dropout layers of 20% dropout rate after each Conv/Max Pooling layer. Across all experiments, the dropout layers proved to help the model with accuracy, potentially due to the overfitting issue of all models. In addition to the dropout layers, stopping training after 3 epochs helps to reduce overfitting with our relatively small dataset. One further recommendation to improve on this CNN model would be to retrieve additional film reviews to build a larger corpus of documents.

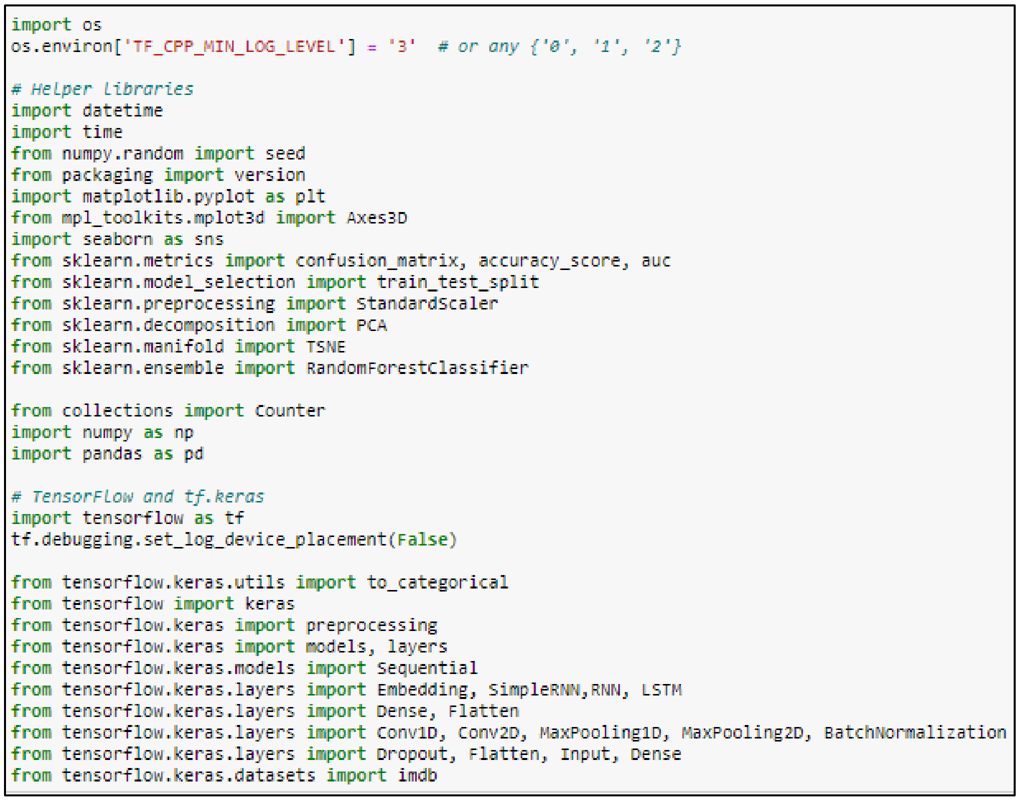
References

Holtes, G. (2019, March 01). Surprising Findings in Document Classification. Retrieved November 01, 2020, from https://towardsdatascience.com/surprising-findings-in-document-classification-7a79e30f1666

Appendix

Figure 1

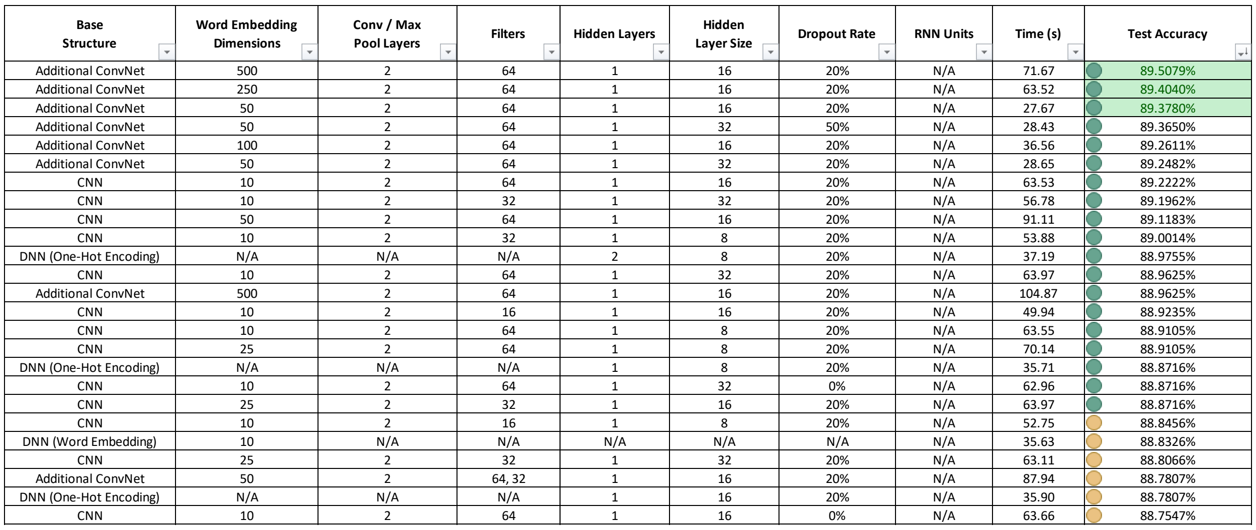
Python Libraries Used for Research



```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}

# Helper libraries
import datetime
import time
from numpy.random import seed
from packaging import version
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier

from collections import Counter
import numpy as np
import pandas as pd

# TensorFlow and tf.keras
import tensorflow as tf
tf.debugging.set_log_device_placement(False)

from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import preprocessing
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN,RNN, LSTM
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv1D, Conv2D, MaxPooling1D, MaxPooling2D, BatchNormalization
from tensorflow.keras.layers import Dropout, Flatten, Input, Dense
from tensorflow.keras.datasets import imdb
```

Figure 2

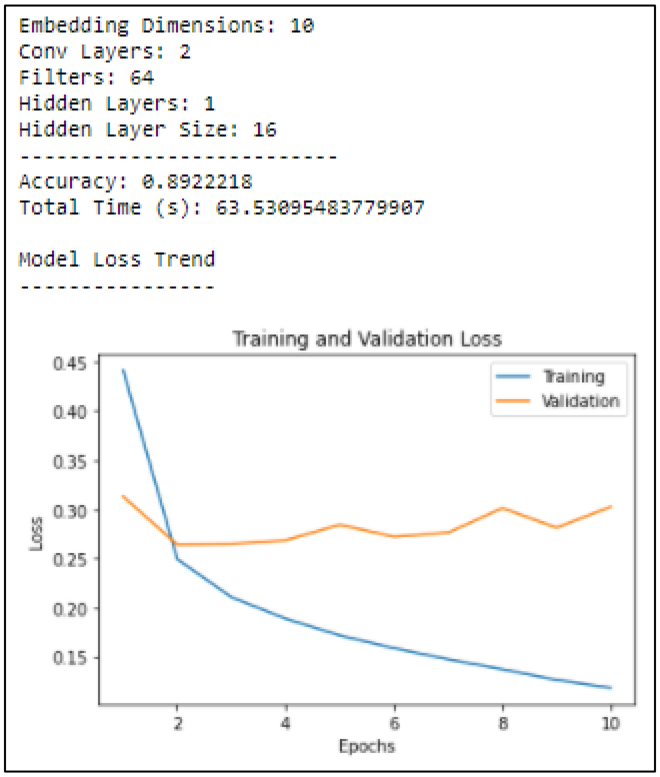
Top 25 Model Results



Base Structure	Word Embedding Dimensions	Conv / Max Pool Layers	Filters	Hidden Layers	Hidden Layer Size	Dropout Rate	RNN Units	Time (s)	Test Accuracy
Additional ConvNet	500	2	64	1	16	20%	N/A	71.67	🟢 89.5079%
Additional ConvNet	250	2	64	1	16	20%	N/A	63.52	🟢 89.4040%
Additional ConvNet	50	2	64	1	16	20%	N/A	27.67	🟢 89.3780%
Additional ConvNet	50	2	64	1	32	50%	N/A	28.43	🟢 89.3650%
Additional ConvNet	100	2	64	1	16	20%	N/A	36.56	🟢 89.2611%
Additional ConvNet	50	2	64	1	32	20%	N/A	28.65	🟢 89.2482%
CNN	10	2	64	1	16	20%	N/A	63.53	🟢 89.2222%
CNN	10	2	32	1	32	20%	N/A	56.78	🟢 89.1962%
CNN	50	2	64	1	16	20%	N/A	91.11	🟢 89.1183%
CNN	10	2	32	1	8	20%	N/A	53.88	🟢 89.0014%
DNN (One-Hot Encoding)	N/A	N/A	N/A	2	8	20%	N/A	37.19	🟢 88.9755%
CNN	10	2	64	1	32	20%	N/A	63.97	🟢 88.9625%
Additional ConvNet	500	2	64	1	16	20%	N/A	104.87	🟢 88.9625%
CNN	10	2	16	1	16	20%	N/A	49.94	🟢 88.9235%
CNN	10	2	64	1	8	20%	N/A	63.55	🟢 88.9105%
CNN	25	2	64	1	8	20%	N/A	70.14	🟢 88.9105%
DNN (One-Hot Encoding)	N/A	N/A	N/A	1	8	20%	N/A	35.71	🟢 88.8716%
CNN	10	2	64	1	32	0%	N/A	62.96	🟢 88.8716%
CNN	25	2	32	1	16	20%	N/A	63.97	🟢 88.8716%
CNN	10	2	16	1	8	20%	N/A	52.75	🟡 88.8456%
DNN (Word Embedding)	10	N/A	N/A	N/A	N/A	N/A	N/A	35.63	🟡 88.8326%
CNN	25	2	32	1	32	20%	N/A	63.11	🟡 88.8066%
Additional ConvNet	50	2	64, 32	1	16	20%	N/A	87.94	🟡 88.7807%
DNN (One-Hot Encoding)	N/A	N/A	N/A	1	16	20%	N/A	35.90	🟡 88.7807%
CNN	10	2	64	1	16	0%	N/A	63.66	🟡 88.7547%

Figure 3

Test Loss vs. Number of Epochs in Baseline Convolution Model



```
Embedding Dimensions: 10
Conv Layers: 2
Filters: 64
Hidden Layers: 1
Hidden Layer Size: 16
--------------------------
Accuracy: 0.8922218
Total Time (s): 63.53095483779907

Model Loss Trend
----------------
```