Colin Hwang

Professor Sable

**ECE 467: Sentiment Analysis of Game Reviews**

**Introduction:**

For my project, I decided to use TensorFlow to design a neural network with the task of performing sentiment analysis for game reviews. With TensorFlow, I used the deep learning API Keras for tokenizing, padding, building my neural network, and training my architecture. I used a dataset from GameStop that consists of product reviews on GameStop’s websites. In the dataset, the user gives a review and a rating from 1 to 5. I ended up deciding to split the dataset into an 80:20 train/test split. The split is done automatically within the program each time it is run.

**Other Variables:**

Aside from the architecture, there were many things to consider when optimizing my neural network: learning rate, batch size, number of epochs, loss function, optimizer, test/train split, etc. There were too many variables and not enough time to evaluate how changing each, especially with respect to each other, would affect my system. Therefore, I decided to keep a most variables constant once they produced decent enough results.

For my optimizer, I used the Adam optimizer that had a built-in learning rate of 0.001, which I didn’t change. There was no real reasoning behind choosing Adam other than it preforms stochastic gradient descent. I also kept my loss function, categorical cross entropy, the same as well as the train test split for all trials. Therefore, aside from the architecture, I only experimented with the number of epochs and the batch size. Generally, more epochs and a lower batch size produced better results.

**Architectures and Results:**

Architecture 1:

For my first architecture, I used a fully connected RNN from Keras’ SimpleRNN class with 32 output nodes (this was arbitrarily decided without much thought). Since I was using one hot vectors as my input, I used an embedded layer from Keras’ embedded layer class. For my output layer, I used a dense layer, again from Keras’ denser layer class, with softmax activation and five output nodes for the five different possible ratings. For all my architectures, I used the same input and output layer.

Graphical user interface, text

Description automatically generated

Architecture 1 Results:

Chart, bar chart

Description automatically generatedText

Description automatically generated

My first attempt did not go so well. The data in general is heavily skewed towards ratings of 5, which likely caused the result we see in the bar graph. From the first to the second epoch, there was a jump in the accuracy. However, nothing meaningful occurred afterwards.

Architecture 2:

Since my first architecture did not go so well, I decided to try out LSTMs from Keras’ LSTM class. The output nodes per layer and the output/input layer were all kept the same. The first LSTM layer would return sequences while the second LSTM layer did not.

Graphical user interface, text

Description automatically generated

Architecture 2 Results:

Chart, bar chart

Description automatically generatedText

Description automatically generated

As seen by the bar graph, this model using LSTMs correctly predicted reviews from other categories, not just five ratings. However, it did not predict any reviews with the rating of 2. I was pleased the jump in accuracy, and I could get even better results with more epochs.

Architecture 3:

Another type of RNN offered by Keras was the GRU layer (gated recurrent units). We did fully cover this type of RNN in class so I am not too familiar with it. However, I found online that GRU’s tend to work better than LSTMs on smaller datasets. I was unsure whether my dataset (consisting of 4687 reviews) classified as a “smaller dataset”, but I was willing to try it out.

A screenshot of a computer

Description automatically generated with medium confidence

Architecture 3 Results:

Chart, bar chart

Description automatically generatedText

Description automatically generated

This architecture performed ever so slightly better, but this could have been for several reasons. The testing and training split was being recreated every time the program was run, therefore it this architecture could have coincidentally gotten a dataset with less 2’s, which is harder to predict. It appears that it did get less 2’s looking at the bar graphs. I deemed both models to produce equivalent results.

**Final Network:**

For my final network, I stuck with the second architecture since it uses an RNN I am familiar with and produced an accuracy that is clearly better than random results. I ran the model again with 20 epochs instead of 6, and the accuracy drastically increased from roughly 80% precent to 93.81%. I assume that I could get into the high 90’s for accuracy if I ran enough epochs. However, this result was good enough for me, and I decided to give my laptop a break.



Chart, bar chart

Description automatically generated