Team 8: ChatBot

1st Brooke Bound
dept. of Computer Science (off Aff.)
MTSU (of Aff.)
Murfreesboro TN, USA
brb5u@mtmail.mtsu.edu

2nd David Brugger
dept. of Computer Science (of Aff.)
MTSU (of Aff.)
Murfreesboro TN, USA
dab7r@mtmail.mtsu.edu

3rd Jake Hagenow
dept. of Computer Science (of Aff.)
MTSU (of Aff.)
Murfreesboro TN, USA
jmh2ej@mtmail.mtsu.edu

4th Lorenzo McDaniel
dept. of Computer Science (of Aff.)
MTSU (of Aff.)
Murfreesboro TN, USA
lsm3x@mtmail.mtsu.edu

5th Paula Pamplona
dept. of Computer Science (of Aff.)
MTSU (of Aff.)
Murfreesboro TN, USA
pmp3g@mtmail.mtsu.edu

***Abstract*—When we set out to create the chatbot, our goal was to successfully train a model that would accurately be able to answer predetermined questions users would give it. We were able to train our model to 100 percent accuracy. Our chat bot was able to respond to the predetermined text patterns we trained it on but did not generalize the text that may deviate from the training sets. Upon testing our model, we found that typing exactly the question pattern we trained for the model would produce the desired result. We wanted to dig deeper and check if our model could generalize text, it had not been trained on. After performing our quantitative and qualitative tests on the performance of our model we found the 100 percent accuracy our model was producing was misleading. We found 95 percent on average, the chatbot's accuracy fell between 39 percent - 49 percent. Using a loop, confusion matrix, and Cross validation technique known as repeated sub-sampling, we were able to determine which categories our chatbot struggled with and which categories it performed well in.**

***Index Terms*—Chatbot, Natural Learning, Natural Language Processing**

I. Introduction

Our motivation for this project was to create a useful application of a trained neural network to respond to simple questions and give guidance to customers in a real world situation. Our chatbot can answer questions that relate to a business it is trained on. It could be implemented in an online store website, and could also be implemented in some sort of text message system. Our key aim when creating this project was to come up with a useful tool that could be customized for lots of different situations.

II. Background

The background for this project was based on previous work we had looked up. Such Chatbots as Heek, Amtrack, and Hazel gave us inspiration for things we would like to do. In our chat bot we use Tokenization to be able to recognize small pieces of text. Tokenization is a way of seperating a piece of text into smaller units called tokens. We also use a process called word stemming which is a process where words are reduced to a root. We use this in order to dumb the text down so that the computer can easily understand it. Basically we just clean up the words so that the computer can recognize them better.

III. Methods

For our chatbot, we used tensorflow, JSON, numpy, keras, IPython.display, and matplotlib.pyplot in order to build a functioning chatbot. Tensorflow and keras were used to build the bot, JSON was used to read our training data, numpy was used to create arrays from our data, IPython.display was used to display results, and matplotlib.pyplot was used to plot our results.

Firstly, we used a .JSON file to store our training data. We filled this file with phrases that are likely to be asked by the user, along with various responses to these phrases. After importing this file, we extracted the data into words and labels as well as patterns and their associated tags. We then stemmed the words in order to slim down and simplify the data for our bot. We used LancasterStemmer from nltk.stem.lancaster in order to perform the stemming.

After stemming, we looped through our words (both input and output) and represented each sentence with numbers (creating a bag of one-hot-encoded words). Finally, we turned these lists into numpy arrays.

Next, we built a multilayer network with ReLU hidden layers. We created an output layer of size 6 with a softmax activation function. We compiled this model with the Adam optimizer and all normal Adam optimizer hyperparameters.

After compiling, we trained our model with a batch size of 1 and went through 100 epochs. The program will then run the chat function, which will continue to run until the user types 'quit.' [1].

IV. Results

Figure 2 This is the resulting summary of the model. We have several hidden layers all funneling to the output layer that has the softmax activation function.

Figure 3 These are a few selected epochs from training. As is clearly shown, loss is going down as accuracy is

increasing. We found 100 epochs to be a good amount of training for this bot, as we continually got 100

Figure 4 Our loss continually dropped as we went further in the training. Though loss is not 0, it is still low enough for the bot to fully function. (5.009 after 100 epochs).

Figure 1 We wanted to see how well our chatbot can respond to patterns we may not have seen. We used a Cross Validation statistic technique known as repeated random sub sampling in order to help answer the previous question. Repeated random sub sampling "creates multiple random splits of the dataset into training and validation data. For each such split, the model is fit to the training data, and predictive accuracy is assessed using the validation data. The results are then averaged over the splits." To accomplish the task, we set up a loop that would grab a random set of data from our intents file. We would not train the model on the random data sample that we grab from the training file, instead we would use this data to evaluate our model when training is complete. We decided to run the loop for 30 iterations, within each iteration of the we would take a random subset of data, evaluate this data on model and record the accuracy. This would allow us to gather enough data to view performance of our model and generate a bell curve. We constructed a bell curve to see where the distribution of our accuracy values lay in the chart. On average our model was getting around 46 percent accuracy. Once we were able to obtain the standard error, we did our calculation for the 95 percent confidence level.

Figure 5 We found that our models upper limit performance was around 46 percent and the lower was 39 percent. Thus, we are 95 percent confident that on average our model will perform between 39 percent - 46 percent for accuracy.

We utilized a confusion matrix to test the efficacy of our net. *Figure 6* In total, the chatbot was taught twenty different categories, but there was another class (-1) added to the possible predictions for when it could not come up with an answer. As seen in the confusion matrix, misclassifications were rare with the most common error being the net not coming up with an answer. There were a few classes that performed with around 70

[2].

Fig. 1. Example of Model Conversation.

V. Discussion

The Quantitative and Qualitative statistics we gathered from our model gives us a baseline starting position for improving our model. With these statistics we can gauge the true performance of our model. We can compare any future versions of our model with this version to see if our model is performing better or getting worse. It is clear now that just viewing the accuracy and loss percentages of our model could be misleading in determining our chatbots performance. Our model was achieving favorable accuracy and loss values, we were able to train the model to 100 [3].

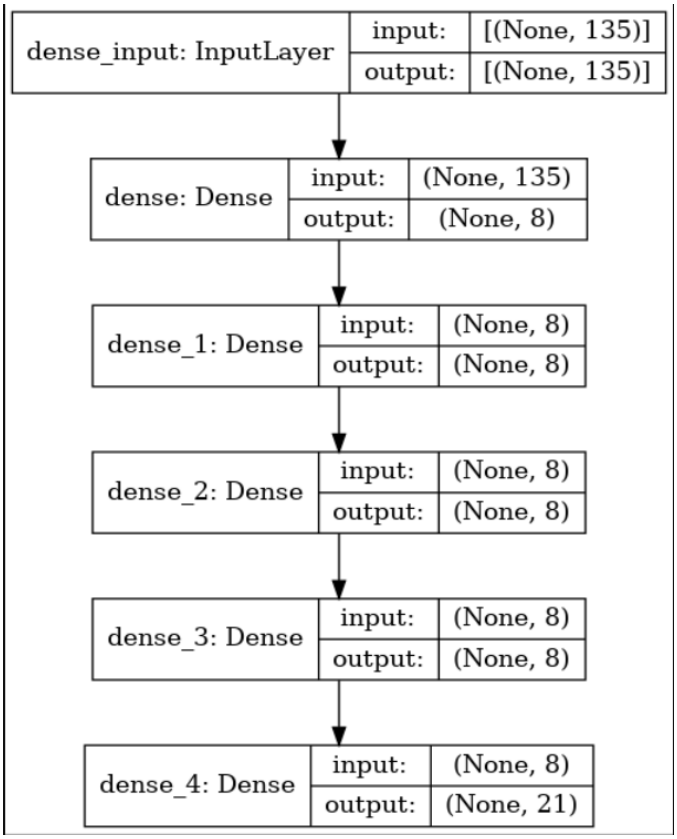

Fig. 2. Model Summary.

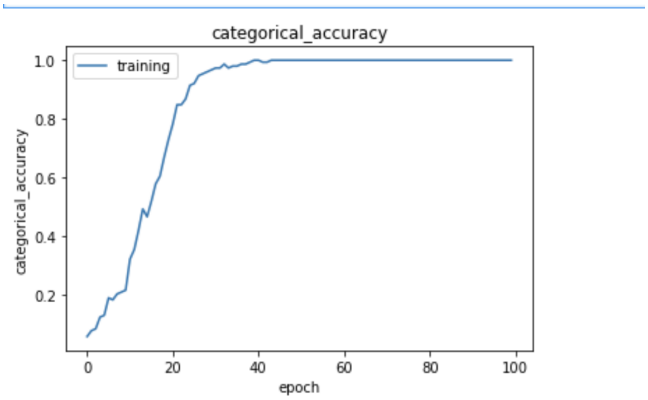

Fig. 3. Example of Model Accuracy.

References

[1] Wikipedia, "Cross-validation (statistics)." [Online]. Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics)
[2] E. Aghammadzada, "Chatbots: intent recognition set."
[3] T. with Tim, "Python chatbot tutorial." [Online]. Available: https://www.techwithtim.net/tutorials/ai-chatbot/part-1/

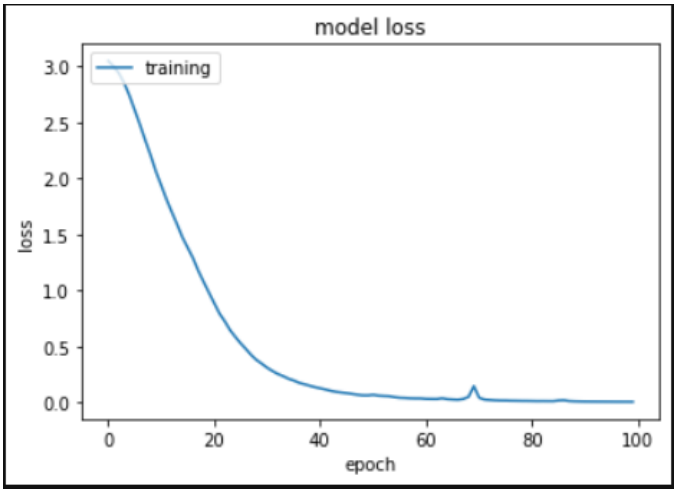

Fig. 4. Example of Model Loss.

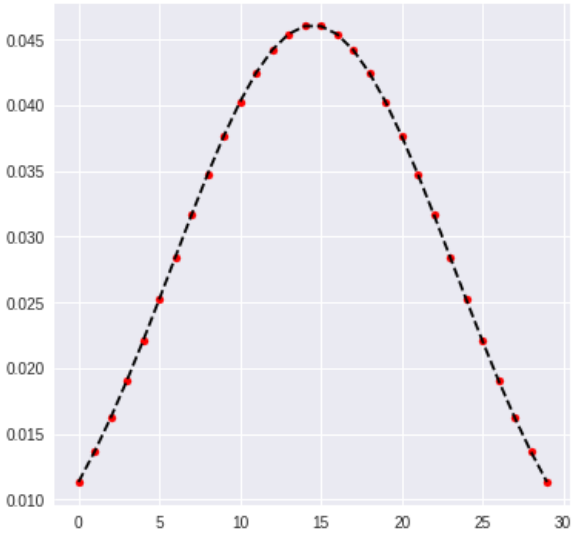

Fig. 5. Example of Mean Distribution.

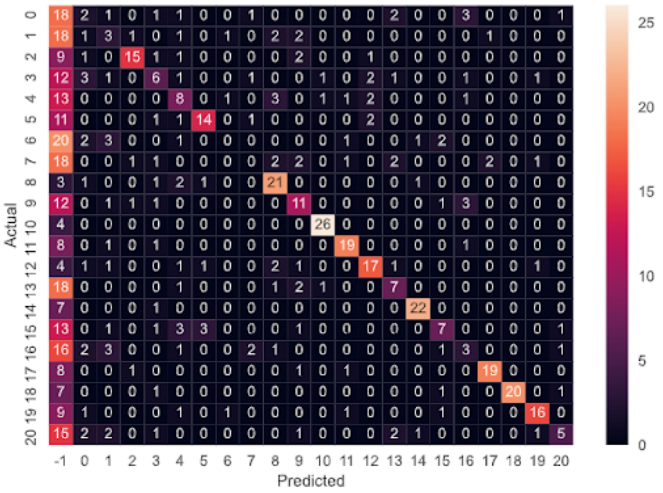

Fig. 6. Confusion Matrix.