Predictive Text Modeling (SeekGen)

Team members: Anirudh Mallya and Alan Grinberg

The aim was to deploy a predictive text tri-gram Neural Language Model (*SeekGen*) that can look at sequence and understand semantics to predict words that a user may type next. This paper will cover the various approaches to the problem, from iMessage text data extraction and preprocessing to calibration of the different Machine Learning and Deep Learning techniques that were used for sequence prediction. The stretch goal is to be able to update these Neural Language Models to output user-oriented predictions over time. This idea is driven by the fact that people's structure of texting changes over time and so does language. Hence, the time taken for a model to periodically update and the space occupied by the model on the operating platform are crucial factors that drive the selection between different Deep Learning models. The model is deployed onto Heroku using a Flask application that acts as the User Interface between the model and user, allowing them to pick from possible word predictions with a key press. It also acts as a source of obtaining new data for updates in the future.

1 Approach

All parts of our approach are original. The problem was approached with the thought that the fourth word in a three-word sequence is very predictable in the English language. For example, "how are you" is most likely followed by words such as "doing" and "going". Not only does it give us a probability likelihood of that word, but also, it's part of speech form and suffix, in this case, a verb with the '-ing' form.

1.1 Data Selection

The major challenge to solving this problem was to pick a dataset that is trained on a vocabulary that is of appropriate size, since one-hot encoding each word (word vector) results in a training example that's the size of the vocabulary. The resulting batch tensor can grow very quickly. Hence, the data used must cover as many general words in the English dictionary as possible.

iMessage chat data is used to learn the language sequence and pos tags for a specific individual. This data was obtained from the personal database file stored by Apple on the MacBook Pro and extracted using SQLite 3.

1.2 Preprocessing

Words were preprocessed from iMessage text data. This was done with tools such as regular expression matching, word tokenization, case matching, frequency-based filtering, lemmatization, contractions, and dictionary validation. Table 1.1 shows the step by step filtering process for an example sentence.

Sentence: A a flying bird's :P	
Preprocessing task	**Result**
Regex'[^a-zA-Z\'\'\?]' filtering	A a flying bird's P
Lower case conversion	a a flying bird's p
English dictionary filtering	a a flying bird's
Contractions	a a flying bird is
Frequency filtering	a flying bird is
Lemmatization	a fly bird is

Table 1.2.1: Sampling of preprocessing methods used

Upon preprocessing, data is ready to be implemented for sequence prediction and Part-Of-Speech (POS) tagging using Machine Learning and Deep Learning classification algorithms.

1.3 Sequence Prediction

Since data here is very complex and class imbalance could be an issue, we felt the need to explore more predictive features to predict from a sequence, rather than using *only* words in the sequence as a feature. Hence, we explored Deep Learning methods using Multilayer Perceptron models (MLP), Convolutional Neural Networks (CNNs) and variations of Recurrent Neural Networks (RNNs) to find these additional hidden patterns to learn from.

Example- Input: “how are you”; output choices:

[“guys”, “deal”, “working”, “a”, “supposed”]

1.4 Part-Of-Speech (POS) tagging

We decided to stick with classical Machine Learning models such as Logistic Regression and Support Vector Classifiers (SVC) for POS-tagging since the number of unique tokens in the data is vastly lesser than in the sequence prediction task. The smaller set of unique tokens makes the data simple to learn from, hence requiring a rather simpler decision boundary.

POS tagging is used in SeekGen to filter out possibly incorrect predictions from the sequence prediction model by ensuring that we meet rules of English grammar. Therefore, it ensures language structure.

From the example in the previous subsection, “how are you” is encoded as ['WRB', 'VBP', 'PRP'] that signifies wh-adverb followed by verb-present tense and a pronoun. The model predicts ['JJ', 'NN', 'RB'] as top choices for the predicted words pos-tag. Therefore, given the sequence, an adjective, noun or adverb is highly likely to be the pos-tag of the predicted word. This helps narrow down output choices to

[“guys, “deal”, “working”].

1.5 Model Saving and Optimizations

Since it’s impractical to train models every time the application is restarted, SeekGen models were saved using TensorFlow and the pickle library. They were then synced onto Google Cloud with appropriate security restrictions for scalability.

The python scripts were optimized to scale better and process faster using NumPy, Pandas and TensorFlow GPU 2.0.

1.6 Deploying SeekGen onto the Web

Based on the practical usage of the word prediction model, we went forward with an application that could be run locally, as well as deployed on the web in the form of a Flask application.

Upon launch, saved Artificial Intelligent models from SeekGen are loaded from the pickled format. The application can be broken down into two parts, the backend serving up the model, pulling a corpus from a database file, and a front-end that loads up a form that accepts the user’s text input and sends it back to the model.

Once processed, the most likely possible predictions are displayed so that a user can make the selection simply from their keyboard with a keypress.

2 Experiments

2.1 Datasets

Data used was obtained from iMessage data from the writers’ personal computers. This covers the general structure of English sentences and chat texts required for the problem.

2.2 Detail and results

All Artificial Intelligent models were run using TensorFlow 2.0.

Sequence model:

The chosen sequence prediction model is a bi-directional Recurrent Neural Network with 512 LSTM cells. The validation accuracy in this scenario is exceedingly tight since the target word doesn’t contain possible paths that could lead to the same meaning of a sentence; hence we use validation accuracy only for detecting overfitting. Data consists of 82,402 training examples and 9,156 validation examples, where the input is a batch of concatenated sequence vector embeddings, and the target is a one-hot encoded vector label of vocabulary size.

Model: Bi-directional RNN LSTM (library: TensorFlow)

Data was split into 90% training and 10% test during training.

Parameters	Meaning
Bidirectional LSTM = 512	512 memory cells for an RNN layer
Dense Layer = 1401 (Vocabulary size)	Output layer cells
Batch size = 32	Batch size processing
Epochs = 30	Number of iterations
hidden activation = relu	Hidden layer activations
Output activation = softmax	Output layer activation, corresponding to predictions.

Table 2.2.1: Bidirectional RNN LSTM calibration

Model Selection:

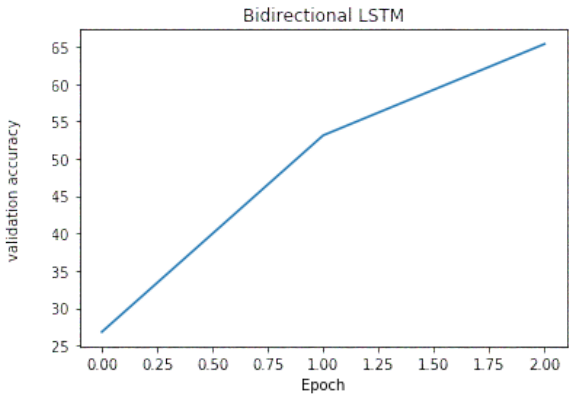

Figure 2.2.1: Bidirectional LSTM shows near linear/ very stable growth in the accuracy rate

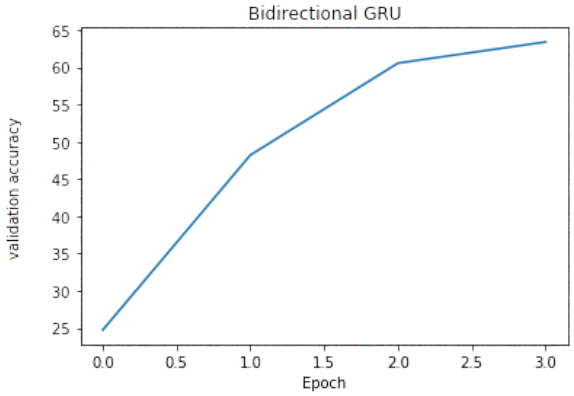

Figure 2.2.2: Bidirectional GRU showing less stable than optimum growth but similar end result to bidirectional LSTM

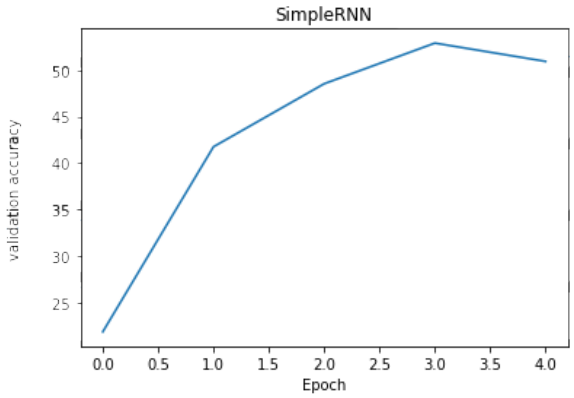

Figure 2.2.3: Simple RNN shows how the simple neural network maintains steady growth but tails off quickly, ending up lower than all other models

The different model architectures were trained up to the point where overfitting began, as evidenced by decreasing accuracy rates. The top performers leveraging 512 units or memory cells provided the best results due to their ability to work well with long distance dependencies. The models were trained until epochs shown in graphs above. As we can see the bidirectional models perform the best relative to the simpler ones such as a simple RNN by capturing more information.

POS-tagging model:

The pos-tagging model uses predictions from a Support Vector Classifier to predict the pos-tag of the next word in the sequence. Data consists of 82,402 training examples and 9,156 validation examples of pos-tagged sequence inputs and their corresponding pos-tagged label output.

Model: Support Vector Classifier

Parameters	Meaning
gamma = 'auto'	Parameter for non-linear hyperplanes
Probability = True	Outputs probability of classes
class_weight = 'balanced'	Deals with class imbalance
Epochs = 200	200 iterations maximum. Early stopping is True.

Table 2.2.3: SVC model calibration

Overall web structure:

Application	Usage
Flask routing/ jinja2	Routing requests sent by the user/application; dynamically changing content based on model output
Scripts: JS/jQuery	Scripts to allow for interaction and sending data to model running in the background
SeekGen	Model files and database predictions are run from, along with word embeddings

Table 2.2.4: Web application structure

Application Preview:

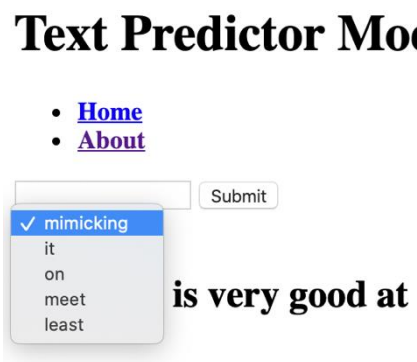

Figure 2.2.2: Preview of the live application

2.3 Evaluation

Since there can be multiple possible right predictions to a sequence, the gold test dataset consists of 85 manually created sentences and evaluated using BLEU score.

BLEU scores range between 0 and 1, the closer to 1, the higher the precision. This metric was used for evaluation between the golden dataset and model predictions. In addition, upon testing on the sentence dataset, a precision of 0.63 was obtained.

This was expected considering the usage of common speech as well as language related to the input data from existing chats. It was presumed that predictions would align well with conversational language as well as the style of users participating in the chats.

3 Future Work & Conclusions

Upon evaluation of the current application, we have identified a couple areas where the user friendliness and flexibility of the application can be improved. One element will be the deployment of the application to Heroku so that it can be accessible globally. The other change would involve adjustment of the Flask app to create a set of accessible APIs that would be used to expand on current functionality/ leverage it beyond the current possible outputs.

Beyond visual and API additions, it is important to note that while the models created and run in this project were fairly effective in generating predictions for chat based conversations, the extendibility of the application would require further work on the learning side as well as the addition of learning over time. We must note that compared to existing word prediction technology, there is always room to improve, but in this particular case, we expect to see accuracy below 100% – leaving room for prediction of most often used words or simply providing reasonable options for the user.

As always with such methods of model creation, we may consider other possible optimizations besides what has already been implemented. Expansion of SeekGen to mobile operating system specific applications would be a useful implementation of the predictions, perhaps in the form of custom suggestions in the predictive tray in a mobile keyboard. Another consideration would be to allow model updates (potentially using cloud-based storage/ a local user dictionary) based on a user's custom data and text input in various applications, allowing user-oriented predictions.

4 References

Brown Corpus, Brown University, 1961.

Komatsu, H., et al. "Corpus-Based Predictive Text Input." *Proceedings of the 2005 International Conference on Active Media Technology, 2005. (AMT 2005).*, 2005, doi:10.1109/amt.2005.1505271.

Pennington, Jeffrey, et al. "GloVe: Global Vectors for Word Representation." GloVe: Global Vectors for Word Representation, 2014, nlp.stanford.edu/projects/glove/.