Emoji2Tweet: BrainStation Capstone

Introduction & Background

Can emojis capture the idea/concept behind a tweet? I attempted to answer this question by creating a model that uses a set of input emojis (💗,🎄,💑) to generate a tweet ("I 💗 spending 🎄with my 💑"). I did not find other projects that tackled this problem, instead I found inspiration from image captioning.

In the problem of image captioning, you have a feature vector that is created from the image you want to caption. This feature vector is merged with the output of a recurrent neural network (RNN). The method used to merge and where the merging is done has been explored in the literature. *Tanti et. al* showed that merging the feature vector with the output of the RNN through addition to provide the best results[1]. In this project I followed a guide on Image Captioning, which applies *Tanti et. al*'s method, and adapted it to my problem[2].

Rather than using an image as the feature vector, I sought to use Natural Language Processing to create a feature vector out of the emojis. In a previous project I explored applying a word2vec model on twitter data[3]. I followed a similar process to create the emoji vectors in this project.

Data Wrangling

Collection and Preprocessing

The data was collected using the tweepy python module to interface with Twitter's Streaming API. I started with 683,362 tweets obtained from the United States and Great Britain, of which 142,733 had at least one emoji. The tweets were stripped of metadata and tokenized using a modified python port of the ARK Tweet NLP tokenizer (my fork). This tokenizer combined crafted regexes to identify the tokens. I modified the python script to include a regex for emojis.

Exploratory Data Analysis

Tweets are on average 10 tokens in length (median: 8), and 95% of the tweets have 23 tokens or less. One of the internal vectors of the model is dependent on the maximum length of the tweet. Thus in order to reduce the size of the model, I limit tweets to be 30 tokens or less. Out of the ~142k tweets with at least one emoji, ~20k did not contain words only emojis. These tweets were removed since they are not representative of the type of tweet we want to generate. Out of the resulting ~122k tweets, ~28k tweets had repeated emojis.

I explored the number of emojis in a tweet, which follows a power law, with an average of 2.17 emojis per tweet (median: 1). 95% of the tweets have 5 emojis or less. I addressed repeated emojis by limiting the number of emojis in a tweet to 10. Additionally I prepared two sets of emoji inputs. The first is the set of emojis extracted from the tweet verbatim. That is, repeated emojis are preserved. The second set avoids duplicates by only keeping unique emojis.

The data set was split, saving 30% of the ~140k tweets for testing. Due to time and resource constraints I reduced the training set to 5000 tweets, as well as the validation set and testing set to 1000 tweets.

---

[1] M. Tanti, A. Gatt, K. P. Camilleri. "Where to put the Image in an Image Caption Generator". https://arxiv.org/pdf/1703.09137.pdf

[2] J. Brownlee. "How to Develop a Deep Learning Photo Caption Generator from Scratch". https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

[3] D. Bravo. "Wordmoji2vec: Embeddings of Words and Emojis". https://github.com/dfbravo/wordmoji2vec/blob/master/report.pdf

Model

The model takes as input a set of emojis (💗,🎄,👫), which are converted into the feature vector. The feature vector is a numeric representation of the emojis, which are assumed to represent the concept/idea behind a tweet. Additionally, the portion of the tweet generated thus far is used as input. The model predicts the token that is most likely to follow in the tweet. This process is repeated until the full tweet is generated. The model architecture is shown in Figure 1.

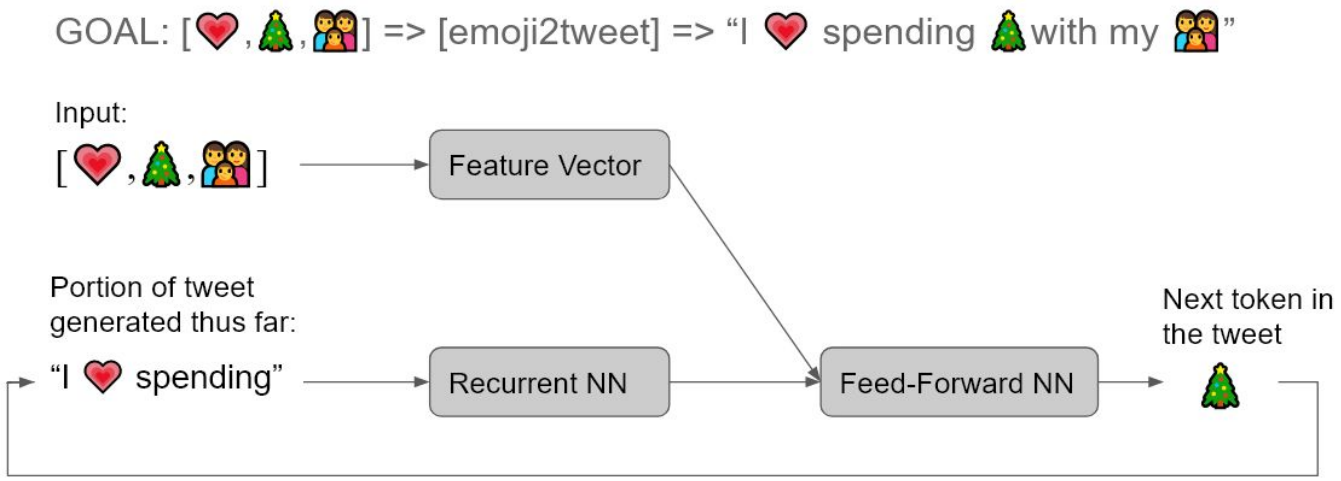

Figure 1: Diagram of a single token creation.The goal is to generate a full tweet starting from a set of input emojis. The model takes as input a set of emojis and the portion of the tweet the model has generated thus far. The next token is predicted and this process is repeated to create a full tweet token by token, where each token is a word or emoji.

Feature Vector

The feature vector is created by converting each emoji into its vector representation from the word2vec model, and summing these vectors together. The word2vec model was trained on the entire training data set, resulting in 300 dimensional vectors. Each feature vector is fed through a dropout layer set at a rate of 50% and finally through a dense layer with a size of 256 units.

Recurrent Neural Network

The RNN component was composed of an embedding layer with 256 units to reduce the dimensionality of the input from a vocabulary size of 10k. The result was fed through a dropout layer at a rate of 50%, and finally through a Long Short-Term Memory layer using 256 units.

Merging and Predicting the Next Token

The feature vector and result of the RNN are merged through two different methods, addition and concatenation along axis=1. The result is fed through a 256 unit dense layer, and ultimately through a softmax function to predict the next token.

Evaluation

To evaluate my model I decided to use the same method used in image captioning: the bilingual evaluation understudy (BLEU) score. This method evaluates a generated caption (or tweet) against reference captions in the dataset by comparing their n-grams regardless of word order. That is, the n-grams of the generated tweet are compared to the n-grams of the reference tweets. The BLEU scores are averaged over the data set to produce a final score for the model.

I compared 2 variations of the model based on the merging method, addition and concatenation along axis=1, and two sets of emoji inputs, a verbatim list of emojis extracted from the tweet, and a list of unique emojis extracted from the tweets. The model with the highest BLEU scores for the validation set used concatenation for merging, and the verbatim list of emojis.

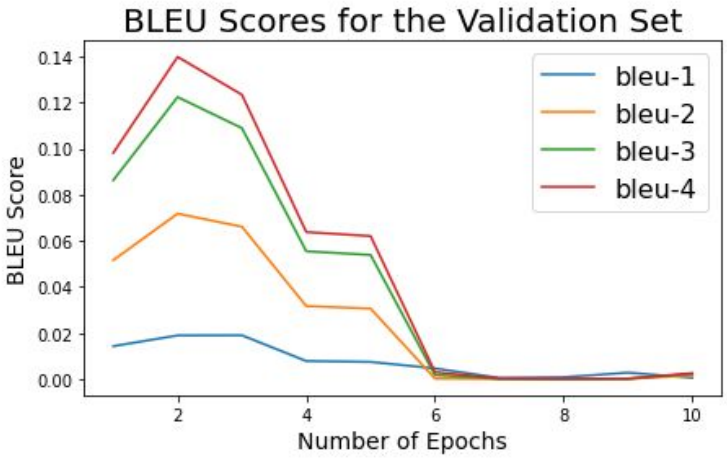

Table 1: BLEU Scores for the Testing Set using the optimal model

	score
bleu-1	0.018915
bleu-2	0.071796
bleu-3	0.122410
bleu-4	0.139877

Figure 2: The BLEU scores calculated for the validation after every epoch. The scores are highest at epoch 2, so we will use this specific model to evaluate the testing set.

The BLEU scores were calculated at each epoch to determine for how long the model should be trained.In this case, we use the model saved at epoch 2. A plot of scores can be seen in Figure 2. The BLEU scores calculated for the evaluation set are found in Table 1.

Conclusion & Extensions

So can emojis capture the idea/concept behind a tweet? I was able to create a model that generates tweets based on an input set of emojis, but the results leave more to be desired. With more data and resources I could improve the results, but more work is required to determine a proper method of evaluating the generated tweets.

A flaw in adapting the BLEU score as the evaluation method is the definition of a reference tweet. In image captioning, the reference captions are the ground truth captions for an image. What is a ground truth tweet for an input set of emojis? Emojis are used in many contexts and simply looking up tweets that use the same emojis will not guarantee these reference tweets are representative of the ground truth (concept/idea behind the tweet). For this reason I used only the original tweet as the reference tweet. That is, I try to recreate the tweets from the testing set and use the BLEU score to evaluate the recreation.To address this issue I would like to explore using a clustering algorithm using the similarity of tweets. That is, I could use similarly clustered tweets as reference tweets. Additionally, exploring other methods of evaluation is recommended, as the BLEU score might not be the best fit for this problem.

In the future I would like to deploy on a platform with more resources, refactor my code to facilitate a grid search-like hyperparameter optimization of emoji2tweet's and word2vec's parameters, and to perform cross-validation by automating the process training process. Additionally, I would like to explore augmenting the feature vector to include words and not just emojis. The motivation is that important keywords will add more context improving results. Determining which keywords are important is an interesting question. What's the least amount of emojis/words required to generate a sensible tweet? A model that properly generates tweets from emojis could empower users to say more with less. Why type out a whole tweet when some emojis can be used to generate it?