Emoji Prediction using Transfer Learning

Satwik Srivastava
M21MA006
Department of Mathematics
IIT Jodhpur

Deepti Gupta
M21CS005
Department of Computer Science
IIT Jodhpur

Abstract—**In the current world where social media is being used extensively to express opinions and emotions we see the use of emojis being increased day by day. More and More complex emojis are required to express what you are saying to express it better in this connected world. Emoji prediction is implied in the IT industry as a feature which suggests emoji based on the text you are writing and all social media platforms imply some form of it in their product(s). However emoji prediction can also be used to predict an emotion for a given sentence or a tweet. This can be used in sentiment analysis. We can thus describe emoji prediction as an extension of the text prediction problem. In this report we aim to analyze a solution to this problem obtained using transfer learning.**

I. Introduction

Transfer Learning is a concept where we use our previously learned knowledge and apply it into another application domain (similar or different) to get results faster. Instead of designing a solution from scratch we choose a solution (architecture) which has worked before and change the final layer to provide us the required outputs. Here a previously developed model is thus a starting point for our new model.

II. Problem Statement

A. Text to Emoji Prediction

Given a dataset containing text to *emoji* mapping, your goal is to prepare a model which can predict learn from this dataset and predict the emoji mappings for a given test dataset.

B. Twitter Emoji Prediction

This is the extension of the above problem. In this we train our model on the twitter emoji dataset and find out the accuracy that we get for predicting emoji for tweets. These predictions can also be used for sentiment analysis of the tweets.

III. Concepts

To achieve our required model we used certain concepts of machine learning which make our process of learning, optimization and updation easier and more efficient. These concepts are described below.

A. Word Embeddings in Natural Language Processing

Word Embeddings or *Word Vectorization* is a method in Natural Language Processing (NLP) in which a word or a phrase (or phrases) are mapped to vector of real numbers which then is used for further learning tasks and model building. It is a dense word representation method and provides a much better way of representing a word as compared to a sparse representation like *one-hot encoding. One of the benefits of using dense and low-dimensional vectors is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors. … The main benefit of the dense representations is generalization power: if we believe some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities*[1].

B. Recurrent Neural Networs and LSTM

A Recurrent Neural Network (RNN) is Neural Network that uses a variable number of layers such that each layer takes input of a specefic position in a sequence (called a timestamp). The inputs are provided in a sequence to the *temporal layers*. RNN layers use the same set of parameters. This process is called *parameter sharing*

An unfurled RNN is reprented in *fig. 1*:

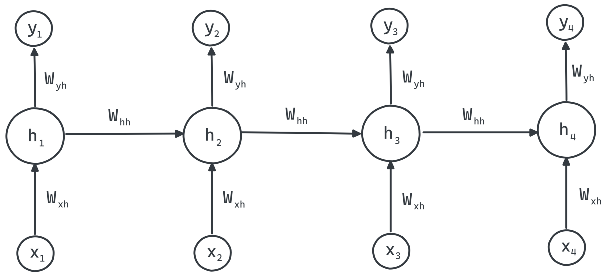

Fig. 1. An unfurled RNN

Long-Short-Term-Memory is an RNN architecture that is used to learn long term dependencies. They were introduced by Hochreiter Schmidhuber (1997) Like RNNs, LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.' The key to LSTMs is the use of Cell state. An LSTM layer has the ability to add or remove information through the use of gates. These gates help us keep in check the vanishing and exploding gradient problem in our neural network in check. The gates used in a typical LSTM are: *forget gate, input gate, output gate, update gate*.

IV. Methodology

We have used two different datasets in this experiment. The first one is the *emojize* dataset and the second one is the *Twitter Emoji Prediction Dataset* both of which are available on Kaggle.

A. Exploratory Data Analysis

We analyze the data to map emojis to the given sentences thus preparing the mapping for the text-to-emoji dataset. Similarly for the twitter dataset the mapping file was given and a dictionary (hash table) is used to quantify the mapping for our problem.

B. Embedding

We use GloVe Vector embedding[2] to vectorize (or create embeddings for) our data. The **glove6B50d** dataset contains embeddings for 6 Billion takens where each embedding has 50 dimensions.

The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics. For large corpora, this pass can be computationally expensive, but it is a one-time up-front cost. Subsequent training iterations are much faster because the number of non-zero matrix entries is typically much smaller than the total number of words in the corpus.[3]

GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. For example, consider the co-occurrence probabilities for target words ice and steam with various probe words from the vocabulary. Here are some actual probabilities from a 6 billion word corpus:[3]

The training objective of GloVe was to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well. For this reason, the resulting word vectors perform very well on word analogy tasks, such as those examined in the word2vec package.[3]

To assign embeddings we first initialize a hash-map that takes in words as keys and returns the embeddings as values. Then we call a function that takes in our data as input and for each word in a sentence creates a list of nd-arrays (here n=50) for each sentence/tweet.

We can also use different embeddings like *wordtovec* by google or change the dimensions of the current GloVe Datset.

V. Results

The architectures used are represented using a `model genome string` in which each layer is separated using a `";"`. The resulting model Architectures and the accuracy achieved running for given epochs is represented in the table below.

- 1: "LSTM 64; Dropout 0.3; LSTM 32; Dropout 0.2; Dense 10 relu; Dense 5 softmax;"
- 2: "LSTM 128; Dropout 0.3; LSTM 64; Dropout 0.2; Dense 32 relu; Dense 20 relu; Dense 5 relu;"
- 3: "LSTM 128; Dropout 0.3; LSTM 64; Dropout 0.2; Dense 32 relu; Dense 20 relu; Dense 5 softmax;"
- 4: "LSTM 256; Dropout 0.3; LSTM 128; Dropout 0.3; Dense 128 relu; Dense 64 relu; Dense 32 relu; Dense 20 softmax;"

Model Genome	Epochs	Train Accuracy(%)	Validation Accuracy(%)
1	10	41.53	21.43
2	500	91.53	92.68
3	500	-NAN-	-NAN-
4	10	21.49	21.83

References

[1] Page 92, Neural Network Methods in Natural Language Processing, 2017.

[2] Jeffrey Pennington, et al. 2014. GloVe: Global Vectors for Word Representation.

[3] Pennington, J. (2014). GloVe: Global Vectors for Word Representation. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/