Deep Learning Final Project

Deep Learning for Fun:
Humor Classification on Yelp Reviews

Marwa Oussaifi
Maria Vasilenko

Abstract

Identifying humor in the everyday setting is deemed to be a subjective task. Yet, there are certain language patterns and structures that are widely recognized as funny. We attempt to use developments in Deep Learning field and apply to build deep learning models capable of classifying the text as funny or not. We analyze a set of 150,000 Yelp restaurant reviews, using "funny" upvotes from users to categorize reviews as humorous. We trained LSTM models, GRU, CNN models, and sequential combinations of LSTM and CNN models. A sequential combination of CNN and LSTM models gave us the highest accuracy of 79.1% on a test set. We attribute this result to the model's ability to combine the best features of the underlying algorithms: keep long term dependencies (LSTM) and create window-of-word features (CNNs).

Introduction and Problem Statement

Humor is by nature subjective; a lot depends on the interpretations, cultural background, and personal taste. Still, there are certain recognizable patterns that elicit laughs. While getting jokes from comedy, sitcoms, and punchlines, is pretty straightforward, a much more challenging task is to uncover humor in an everyday setting. Yelp reviews provide a good example of such a setting, since they represent the opinions of average people, not professional writers, on everyday matters. Yelp reviews are accompanied with the users' "funny" upvotes which to easily label the reviews as humorous or not. Thus, we identify our problem as a binary classification.

This project is inspired by the paper from our peers at Stanford: "Deep Learning Humor Classification on Yelp reviews".

Goal

The primary goal of our project is to recognize the humor in Yelp reviews, which give a good representation of everyday setting, using a variety of Deep Learning algorithms.

Data

Our data set is comprised of Yelp reviews, each coming with a text field for the actual review and "funny" upvote counts that provide the number of people who found the particular review

funny. We binarize this response variable by marking all reviews with more than 4 funny upvotes as a humorous, and others as not. The data set is available as JSON files. We balance the data by getting ~75,000 of the humorous and same amount of not humorous reviews, as per our definition. In total, we get a corpus of 150,000 reviews (to compare, R.Bais and M.Torres from Stanford took 120,000 reviews in total).

Algorithms

We tried several deep learning algorithms to approach the text classification problem. Specifically, we ran several modifications of LSTM, GRU and CNN models, as well as tried to make a combination of LSTM and CNN models.

We choose to try LSTM and GRU models so as to try and capture long-distance dependencies in the text. CNNs are, on the other hand, are helpful in the sense that they allow creating "windows" of words.

We took the models we learned in Yannet Interian's Deep Learning class as a base, as well as looked for implementation of text classification deep learning models in other sources:

1. LSTM models: https://github.com/yanneta/deep-learning-with-pytorch/blob/master/lesson7-lstm.ipynb
2. CNN: https://github.com/yanneta/ML-notebooks/blob/master/cnn-text.ipynb
3. Text Classification models with PyTorch: https://github.com/AnubhavGupta3377/Text-Classification-Models-Pytorch/blob/master/Model_RCNN/model.py
4. Text Classification models: https://github.com/prakashpandey9/Text-Classification-Pytorch/blob/master/models/LSTM_Attn.py

We performed the following modifications to the original code from class.

Text pre-processing step: we performed text cleaning and tokenization using Regex and SpaCy libraries. We decided to keep punctuation as we believe its use provides a way of expressing humor and thus may be useful in modeling.

We wrote our **own Dataset class** designed to get, encode and index reviews from the corpus of Yelp reviews.

For LSTMs, we used larger embedding and hidden layers sizes: 100 vs 50 in class. We also played around different learning rates and epochs sizes. We also reproduced the GRU model.

For CNN, we reproduced the model with 3 1D-convolutions (kernel sizes of 3,4 and 5), applying it to sentences with the max length of 400 words. We also added an embedding layer to train embeddings, rather than using pre-trained ones.

Our major model variations come from building sequential combinations of LSTM and CNN models.

Models Specifications and Results

Below we specify major model parameters and architecture and present the test accuracy scores.
Vanilla LSTM: we used plain LSTM with 0.5 dropout and embedding and hidden layer sizes of 100. The best model was trained using the learning rate of 0.005 (batch size = 1000) in 20 epochs.
LSTM with variable length: dropout = 0.5, hidden layer size = embedding layer size = 100. The best model is trained using the learning rate of 0.005 and batch size of 5000
CNN: 3 1D-convolutions layers with kernel sizes of 3,4 and 5, hidden size=300; the best model is trained on 500 batches, with the learning rate of 0.001.
CNN + pre-trained Glove embeddings: 3 1D-convolutions layers with kernel sizes of 3,4 and 5, hidden size=300 and max sentence length of 500. The best model is trained on 500 batches, with the learning rate of 0.001.
GRU: we used GRU with 0.5 dropout parameter and embedding and hidden layer sizes of 50. The best model was trained using the learning rate of 0.01 (batch size = 500) in 5 epochs.
GRU + CNN: combines the output of the CNN and the last hidden state of the GRU, employes embedding and hidden sizes of 50. The best model is trained using the learning rate of 0.01and batch size of 1000.
LSTM + CNN: combines the output of the CNN and the last hidden state of the LSTM, employes embedding and hidden sizes of 103. The best model is trained using the learning rate of 0.01 for 5 epochs then trained again for 5 more epochs with the learning rate of 0.001and batch size of 500.
Sequential LSTM_CNN: feeds sequentially the output of the LSTM as an input for the CNN convolutional layers, employes embedding of size 300 and hidden size of 100. The best model is trained using different learnings rates starting from 0.05 over different epochs.
Sequential CNN_LSTM: feeds sequentially the output of the CNN as an input for the LSTM layer, employes embedding of size 300 and hidden size of 100. The best model is trained using different learnings rates starting from 0.005 over different epochs.

Model	Best Test Accuracy	Parameters
Vanilla LSTM	0.772	LR = 0.005, batch size= 1000, embedding layer size = hidden layer size = 100
LSTM with variable length	0.772	LR 0.005, batch size= 500, embedding layer size = hidden layer size = 100
GRU	0.75	LR = 0.01 batch size= 500, embedding layer size = hidden layer size = 50
CNN	0.782	LR = 0.001, hidden layer size = 300, batch size = 500
CNN+GloVe	0.771	LR = 0.001, hidden layer size = 300, batch size = 500, after 1 epoch
LSTM+CNN	0.711	LR =0.01 for 5 epochs then 0.001 for 5 more epochs, batch size= 500, embedding layer size = hidden layer size = 103
GRU+CNN	0.724	LR = 0.01, batch size= 1000, embedding layer size = hidden layer size = 50
Sequential LSTM_CNN	**0.791**	Different learning rates over different number of epochs starting from 0.05 LR, batch size= 500, embedding size = 300, hidden layer size = 100
Sequential CNN_LSTM	0.79	Started with learning rate of 0.005 and kept decreasing it over epochs, batch size= 500, embedding size = 300, hidden layer size = 100
CharRNN	Still running :)	

Lessons Learned and Future Research

Our experiments with neural network algorithms show that LSTM and CNN models do a fairly good job in terms of identifying "funny" texts in an everyday setting. The best results (79.1% accuracy) come from the sequential CNN-LSTM model. We attribute this result to the model's ability to combine the best features of the underlying algorithms: keep long term dependencies (LSTM) and create window-of-word features (CNNs).

In the future, we want to work on improving the model architecture as well as tune parameters so that to reach better results with the combination models.

This research could be extended in the future to the problems of detecting different aspects of humor, including negative connotations such as sarcasm and toxic language. Also, it could be

used in a variety of problems where we might be interested in getting more subtle attributes about a user such as assessing his/her sense of humor or other linguistic characteristics. For example, such information could be used for building recommendation engines.

Responsibilities of team members

In order to achieve project objectives, we implemented the following steps:

- Exploratory data analysis (EDA)
- Dataset preparation
- Modeling

We both worked on all of the steps. To speed up the process, we split the modeling part. Maria worked on LSTM and CNN models, while Marwa worked on GRUs and different model combinations.

The code for our project could be found at the following link:
https://github.com/mashamasha/funny_reviews_deep_learning_project

References

1. R. Bais, M. Torres, "Deep Learning Humor Classification on Yelp reviews", 2018
2. L. Gultchin, G. Patterson, N. Baym, N. Swinger, A. Tauman Kalai, "Humor in Word Embeddings: Cockamamie Gobbledegook for Nincompoops", 2019
3. L. de Oliveira and A. L. Rodrigo, "Humor detection in Yelp reviews," 2015.