

Insight into a Network's Decision Making Process

Improving Sarcasm Detection

Usman Kaisser Moghal

FAST University
i191245@nu.edu.pk

Abstract

The latest trend in sarcasm detection involves using different types of networks. Irrespective of the result they show, their inner workings remain a mystery, and hence are not subject to scrutiny for improvement. This work showcases a unique approach into the thresholds of understanding a network's decision making process. The weights (Hyper-Parameters) of a sarcasm detector are analyzed over several epochs during training over a twitter dataset. The weights are then fed into a Deep Neural Network (DNN) which will output the weights with greater relevance with the task at hand (sarcasm detection). These weights can then be fine-tuned and used by the sarcasm detector for faster convergence and more accurate results.

Categories and Subject Descriptors Natural Language Processing [Understanding Decision Making]: Sarcasm Detection

General Terms Understanding Decision Making, Sarcasm Detection

Keywords Sarcasm Detection, DNN, LSTM, Twitter, Convergence, Hyper-Parameter Tuning

1. Problem statement

A neural network is typically used as a black box, which takes some input and after tweaking the network a bit, produces favourable results. The exact inner workings of the network remain a mystery. Moreover, most researchers ignore how the network makes its decisions, so long as the decisions are within the thresholds of the accepted accuracy.

In this paper, an attempt is made to grasp this hidden meaning, and hence influence the network's results.

2. Introduction

2.1 Motivation

In the domain of Natural Language Processing (NLP), Recurrent neural networks (RNN's) such as LSTM's are considered as a standard for problems such as understanding and generating text data [1].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CONF 'yy, Month d-d, 20yy, City, ST, Country.
Copyright © 20yy ACM 978-1-xxxx-xxxx-n/yy/mm...\$15.00.
<http://dx.doi.org/10.1145/nnnnnnnn.nnnnnnn>

Despite their wide spread use, the exact way they function is still uncharted territory. They may be understanding something from the data they are provided, and hence making intelligent decisions, however how this works exactly, remains to be understood. This information is crucial, as it would not only allow researchers to further customize and optimize their networks, but also open up new avenues of research.

There are certain areas of active work, where this insight into the thought process of neural networks, would be immensely beneficial. One of these areas is sarcasm detection.

Sarcasm is a unique linguistic tool, often used to display a specific notion to the audience, one which hints at the originators intents and thinking patterns. Some notions can only be expressed via sarcasm. One unique aspect of sarcasm, is the juxtaposition of the concepts relayed, in that it often contains apparently conflicting or misleading information. This property makes it difficult to detect, even by humans. This is why it's so important that when computer systems are involved, such linguistic features are accurately detected and analyzed.

For instance, it's a documented fact that social media had a major role to play during the 2016 United States presidential election [2] [3] [4]. Various waves generated on social media influenced the election in unprecedented ways. Some believe that the results would have been very different, had there been more regulation of social media.

Once such faucet would be the rampant use of sarcasm in such instances. Since it tends to make a more powerful point, it's often used by others. However, as seen in [5], it's often misrepresented or misunderstood by others, due to lack of transparency. Having a system that can accurately detect sarcasm, and the actual intended meaning, would allow a more accurate view of the situation.

Currently existing sarcasm detectors are to a certain degree, successful. However, they lack adequate data sets to mature fully [6] [7] [8]. Hence, an alternative approach needs to be taken to improve results.

2.2 Ideal Solution

After having been influenced by the work of several researchers, this paper targets an approach that has been yet left unexplored. The idea is to understand the weights (hyper-parameters) of the network. As the network processes more data, the weights are changed to reflect the output. These changes are actually the decisions being made by the network. To understand the decision making process, we need to analyze the weights, and exactly how and why they change. Their behavior will hint at the thinking process of the network.

This is not as simple a matter as it might first appear. These weights are not individuals, but rather part of a collective, that when in tandem, produce the results we see. Meaning, we can't just

consider the base value of the individual weights. There are a slew of other aspects, including but not limited to:

- the inter-dependencies of the weights
- the relation between individual weights and the input values
- the relation between the network loss and the weights

If the full detailed analysis can be performed on these weights, a target network (sarcasm detector) can be highly optimized, especially in this case where datasets for training are scarce. Due to the complex connections of these weights, this would need to be done using another neural network.

So a network needs to be trained to take these weights of the target network, and produce indicators as to their relevance to the task at hand along with behaviors of these weights.

2.3 Proposed Tactic

Due to various limitations, this paper will only attempt to take a step towards the ideal solution. This first step involves getting the weights of a network, and passing them to a rudimentary Deep Neural Network (DNN). The DNN will take these weights along with the accuracy shown by the original network, and after learning the relationship between weights and accuracy, predict the accuracy of new weights.

This is not the same as the ideal solution. However, this will provide a proof of concept, that given weights of a working network, a DNN can learn the behavior of the weights as they change, and thus understand the decision making process.

3. Related work

Many researchers have already realized the importance of understanding the inner workings of a Neural Networks decision making process, and have published commendable papers on related topics. Others have started working on applications involving this concept.

As seen in [9], a technique called Layer-wise Relevance Propagation (LRP) is used to explain the inner workings of a feedforward network, and then used to back propagate the relevance through multiplicative interactions, thus improving the overall network.

In [10] the researchers focus more on how the neural network work in text classification, essentially pointing out that the layers function as ngram detectors, finally suggesting methods to improve text based classification.

Meanwhile [11] attempts to qualitatively compare the different techniques for understanding the ways a network understands and explains data.

Similar to [9], [12] proposes a new technique, dubbed Layer wise Semantic Accumulation (LISA), which attempts to both explain the decision making process of a neural network, and its decision making patterns, specific to text classification problems.

[13] takes a slightly different approach, by monitoring a network's decision making process with regards to character level learning patterns. By extending the contextual decomposition technique, they compare models to show that they implicitly discover various inherent linguistic rules.

The most generalized technique currently in use is customizing the dataset, such that each input has some track-able metadata, whose relevance to the output can be monitored across several iterations [11]. This will lead to a better understanding of how that specific input effected the output.

4. Approach

The method proposed in this paper is inspired by [14], where the weights of a neural network are tracked and analyzed, in order to

predict their trajectory, thus optimizing the computation of weight distributions.

In this paper, the weights of a network are analyzed, for the purpose of understanding how they attribute to the problem of sarcasm detection, within a fixed dataset, using a specific neural network.

The idea is that specific nodes of the network will provide more information on the actual problem, in this case sarcasm detection. Moreover, the relationship between the values of different weights can also encapsulate the decision making process.

If the weights can be predicted, and the pattern understood, it should ideally inform us of the exact elements of the network responsible with dealing with the task at hand, as well as their importance in solving the task. Using this, the network can be optimized immensely for future datasets or similar problems.

With this method in mind, the network from [6] is used as the base neural network for sarcasm detection. The network is run over the same twitter dataset and using the same methodology as mentioned in the original paper. Over multiple iterations, the weights of the networks are saved as numpy arrays, along with the accuracy for that specific iteration.

Considering the concept of gradient descent, as the original network continues to optimize itself and produce a better accuracy, it will continuously provide weights which should be closer to the "optimal" values. Here, a dense neural network will take this stream of values per epoch, and consider their relation to the accuracy, as they approach optimal values.

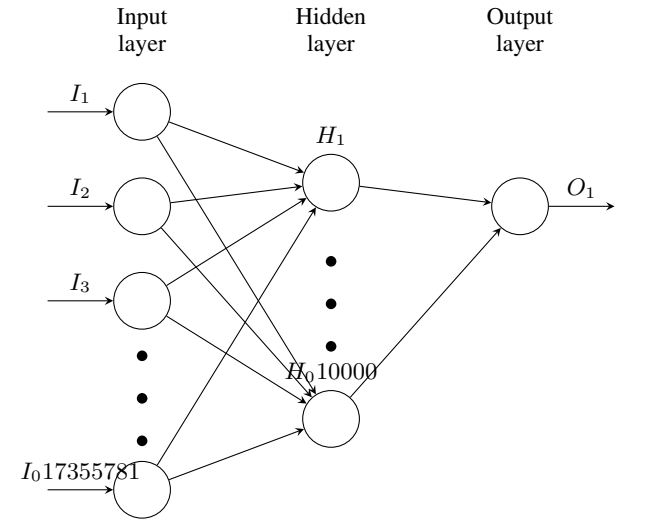


Figure 1. DNN used for learning weights

5. Evaluation and Experiments

The first step of this project involves setting up the network from [6]. The authors have made their code and dataset public on github. After downloading the repository, the "simpler" version of the detection model is used. The "simpler" model was used as it was the most robust out of the others. All required files are uploaded onto Google Colab, batch size set to 8 with epochs at 30, and the training initiated.

For every epoch, the weights of the model were saved as a hdf5 file format, along with the loss and accuracy for that specific epoch.

The weights stored in all the hdf5 files are first read, and the complete data is stored in a csv file, where the weights represent the columns, and each row contains the values of each weight per epoch. This data is coupled with the accuracy and loss information.

Predicted Accuracy	Actual Accuracy
50.73	
61.17	
87.40	

Table 1. Accuracy comparison between actual and predicted values.

Finally, a simple DNN is setup as seen in 1. This DNN has an input size of 17355781 (17355780 weights and 1 accuracy), with 10000 nodes in the hidden layer, followed by one node for output. This DNN takes the csv file as input. The network is then trained over a subset of the csv, and the remainder is used for testing.

Due to various limitations, this current work is limited to a csv containing around 17 epochs, of which 15 are used for training and 2 for test.

Finally, after the DNN has been trained, it will be used to fine-tune the original network (sarcasm detector). For this first step, the original network will be made to run more epochs, and those weights will be recorded along with accuracy per epoch. The DNN will be tested by passing it the new weights, and matching the predicted and actual accuracy's. This will show how well our DNN has understood the network.

This experiment was completed for a few epochs, and even restarted for new initialized weights, and the results are shown in the table 1

6. Future Work

This is merely a proof of concept, that training the weights of a network in a DNN would allow us insight into the decision making process of a neural network (specific to the network and task, generally to the dataset).

The following are changes that are yet to be implemented:

- Re-design architecture of DNN for generating most relevant weights to the task, instead of accuracy.
- Generate more epochs for training.
- Apply sarcasm detector on other datasets for seeing variance across datasets.
- Test DNN on a different problem for further validation

A. hdf5 to CSV conversion

```
import h5py
import numpy as np
import pandas as pd
import csv
import os

accuracy=[0.5121, 0.6096, 0.7629, 0.8000, 0.8459,
          0.8568, 0.8755, 0.8955, 0.9143, 0.9163,
          0.9232, 0.9282, 0.9337, 0.9376, 0.9394,
          0.9465, 0.9487]

acc=0

with open('data.csv', 'w', newline='') as myfile:
    wr=csv.writer(myfile, quoting=csv.QUOTE_ALL)
```

```
for file in os.listdir("data/"):
    if file.endswith(".hdf5"):
        print(file)

        dataset=h5py.File(
            os.path.join("data/", file), 'r')

        conv1d_4_m=dataset['optimizer_weights']
            ['Adam']['conv1d_4']
            ['bias']['m:0'][:]
        conv1d_4_v=dataset['optimizer_weights']
            ['Adam']['dense_8']
            ['bias']['v:0'][:]

        dense_8_m=dataset['optimizer_weights']
            ['Adam']['dense_8']
            ['bias']['m:0'][:]
        dense_8_v=dataset['optimizer_weights']
            ['Adam']['dense_8']
            ['bias']['v:0'][:]

        dense_9_m=dataset['optimizer_weights']
            ['Adam']['dense_9']
            ['bias']['m:0'][:]
        dense_9_v=dataset['optimizer_weights']
            ['Adam']['dense_9']
            ['bias']['v:0'][:]

        embedding_4_m=dataset['optimizer_weights']
            ['Adam']['embedding_4']
            ['embeddings']['m:0'][:]
        embedding_4_v=dataset['optimizer_weights']
            ['Adam']['embedding_4']
            ['embeddings']['v:0'][:]

        lstm_4_m=dataset['optimizer_weights']
            ['Adam']['lstm_4']
            ['lstm_cell_4']['bias']
            ['m:0'][:]
        lstm_4_v=dataset['optimizer_weights']
            ['Adam']['lstm_4']
            ['lstm_cell_4']['bias']
            ['v:0'][:]

        row=[]

        row.append(accuracy[acc])
        acc=acc + 1

        for val in conv1d_4_m:
            row.append(val)

        for val in conv1d_4_v:
            row.append(val)

        for val in dense_8_m:
            row.append(val)

        for val in dense_8_v:
            row.append(val)

        for val in dense_9_m:
            row.append(val)

        for val in dense_9_v:
```

```

        row.append(val)

for val in embedding_4_m:
    for v in val:
        row.append(v)

for val in embedding_4_v:
    for v in val:
        row.append(v)

for val in lstm_4_m:
    row.append(val)

for val in lstm_4_v:
    row.append(val)

print("Data read")
wr.writerow(row)
print("Row Saved")

```

```

# Compile model
model.compile(
    loss='mean_squared_error',
    optimizer='adam')
return model

estimators = []
estimators.append(
    ('standardize', StandardScaler()))
estimators.append(
    ('mlp',
     KerasRegressor(
         build_fn=larger_model,
         epochs=3,
         batch_size=5,
         verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score
    (pipeline, X, Y, cv=kfold)
print("Larger: %.2f (%.2f) MSE" %
      (results.mean(), results.std()))

```

B. DNN Code

```

from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

dataframe = read_csv(
    "data.csv",
    delim_whitespace=True,
    header=None)
dataset = dataframe.values

X = dataset[1:,: ]
Y = dataset[0:1,: ]

def larger_model():
    # create model
    model = Sequential()
    model.add(
        Dense(17355781,
              input_dim=17355781,
              kernel_initializer='normal',
              activation='relu'))
    model.add(
        Dense(10000,
              kernel_initializer='normal',
              activation='relu'))
    model.add(
        Dense(1,
              kernel_initializer='normal'))

```

References

- [1] Chopra, Sumit, Michael Auli, and Alexander M. Rush. "Abstractive sentence summarization with attentive recurrent neural networks." Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016.
- [2] Shane Harris, Ellen Nakashima, Craig Timberg. "Through email leaks and propaganda, Russians sought to elect Trump, Mueller finds." The Washington Post. 2019.
- [3] Ellen Nakashima, Karoun Demirjian, Philip Rucker. "Top U.S. intelligence official: Russia meddled in election by hacking, spreading of propaganda." The Washington Post. 2017.
- [4] "Main Points of Mueller Report" AFP. 2019.
- [5] Sarah McBride. "Yeah right! Sarcasm poses problem for computer algorithms in U.S. election" Reuters. 2015.
- [6] Ghosh, Aniruddha, and Tony Veale. "Fracking sarcasm using neural network." Proceedings of the 7th workshop on computational approaches to subjectivity, sentiment and social media analysis. 2016.
- [7] Jena, Amit Kumar, Aman Sinha, and Rohit Agarwal. "C-net: Contextual network for sarcasm detection." Association for Computational Linguistics. 10.18653/v1/2020.figlang-1.8 (2020).
- [8] Bansal, Srijan, et al. "Code-switching patterns can be an effective route to improve performance of downstream NLP applications: A case study of humour, sarcasm and hate speech detection." arXiv preprint arXiv:2005.02295 (2020).
- [9] Arras, Leila, et al. "Explaining recurrent neural network predictions in sentiment analysis." Association for Computational Linguistics 10.18653/v1/W17-5221 (2017).
- [10] Jacovi, Alon, Oren Sar Shalom, and Yoav Goldberg. "Understanding convolutional neural networks for text classification." Association for Computational Linguistics 10.18653/v1/W18-5408 (2018).
- [11] Arras, Leila, et al. "Evaluating recurrent neural network explanations." Association for Computational Linguistics 10.18653/v1/W19-4813 (2019).
- [12] Gupta, Pankaj, and Hinrich Schütze. "LISA: explaining recurrent neural network judgments via layer-wise semantic accumulation and example to pattern transformation." Association for Computational Linguistics 10.18653/v1/W18-5418 (2018).
- [13] Godin, Frédéric, et al. "Explaining Character-Aware Neural Networks for Word-Level Prediction: Do They Discover Linguistic Rules?" Association for Computational Linguistics 10.18653/v1/D18-1365 (2018).

- [14] Franchi, Gianni, et al. "TRADI: Tracking deep neural network weight distributions." arXiv preprint arXiv:1912.11316 (2019).