Deep Learning - Nagy Házi Feladat

Reddit-BobTheBuilder

Oláh Gergely	**Oczot Balázs**	**Bunth Tamás**
FWXYG1	**PRAMRX**	**IWVRAI**
olahkoo@gmail.com	*balix18@gmail.com*	*btomi96@gmail.com*
https://github.com/olahkoo	*https://github.com/balix18*	*https://github.com/Wastack*

Abstract

Our goal was to create a deep neural network that can generate comments based on comments on reddit written by real people. We implemented three different models for the problem and we compared them to each other based on their ability to generate meaningful comments.

1 Introduction

Natural language processing[7] is a very popular field in deep learning[9]. There are different approaches to it, and we implemented three of them to see which one is the most suitable method. The three methods are character based, word based and word2vec based approach.

The first one is to think of comments as character sequences, so the network only predicts one character at a time. We use a recurrent neural network for this problem, more specifically L(ong) S(hort) T(erm) M(emory) network. The word based approach is very similar to this, except this time the text is interpreted as a sequence of words and the model predicts a word based on the previous words.

The last approach is a word2vec-based solution where we predict the next word by defining a context for each word and examining the similarity of these contexts.

2 Implementation

2.1 Obtaining data

In order to get a major number of Reddit comments we decided to use the Rest API of Reddit.

The content of comment_collector.py is responsible for downloading the relevant data from the website. To achieve that it was also required to register a new account, so we can use its id to obtain the data.

The script stores the data in a json file called *askreddit.json*. It should be put to the data directory, so the below methods could find it.

2.2 Training and testing

We made three different solutions:

- Character based,
- Word based,
- Word2vec based.

These methods require different preprocessing, training and testing approach. The following section will explain the above solutions in detail.

2.2.1 Character-based

The training and testing are implemented in the *character_based_lstm_learning.py* source file. The training is implemented using keras, so it is required to be installed. The GPU version is also recommended as the training is relatively slow (1 - 1.5 hours on a 1050 TI). The hyperas package is also needed.

The training starts with a hyperparameter optimization phase, where we train our model for 10 epochs. After that, we choose the best model and further train it with 10 epochs 9 times, and save each iteration into the models directory. The batch size used for the training is 128. This process can be invoked by calling the train_model() function at the end of the file.

The model architecture is a classic LSTM[6] network, where the input dimension is (40, 1), the output dimension is (n, 1) where n is the number of unique characters. There are two inner lstm layers, each with 512 memory cells and a dropout layer. The dropout probabilites are 0.4 and 0.25. After the second LSTM layer there is a dense layer which is the output layer and uses softmax activation. The hyperparameters used for optimizing were the number of cells in the LSTM layers, the value of dropout layers, and the batch size used for the training.

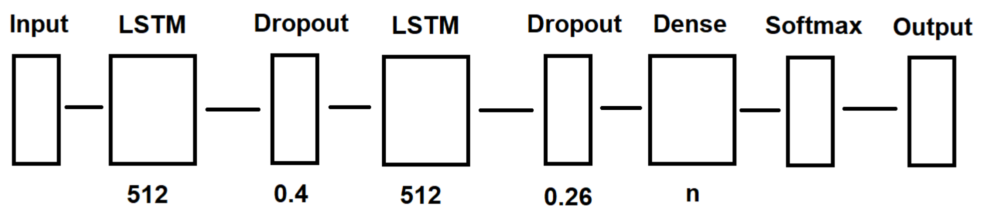

Figure 1: Structure of the model

If we want to test the network we can call the test_data() function. It reads one of the previously trained networks (the last one by default), chooses a random comment from the data set, then generates 400 additional character based on that comment. The iteration parameter of the function determines which model to load from 0-9, where 9 is the one which we trained for the most epochs.

If you don't want to do both training and testing, you have to comment one of the above function calls at the end of the *character_based_lstm_learning.py file* and running it like that.

The generated output of the final model is far away from being meaningful. It was generated from a random comment chosen from the dataset.

Well (awkward pause) we have a lot to tal abduteddl the his thctisg tha dedlssssss andmssssss.than wael i witreltttddds, iruntoogey menn dn bno fdet your enre   Hi suuddtdt teke mentsdtn aev aeraggggs fee haadt tan hacddu Int I haaa bv thek so whek giytttr,oe hedd ihrp

aeenss,sot'mrletnsn moectp mt abwlpt tfteoe tot aneledg oo mu

ahuranin moxh ma   Mnccosgii mit he aooogsgsns. In yor siill hepost be oae fimmdr a yanr woat eo h

The reason for this is probably the model being too small. Also the model probably needs more training, which we did not have the computing performance for. However, this is a lot better than the model after the first few iterations, as that could only repeat a few characters after each other.

Usage**:**

- Make sure the *static-data/single-submission-comments-3.json* exists.
- Simply run the *character_based_lstm_learning.py* file (comment out *train_model()* to test without training).

2.2.1 Word-based

The word based deep learning is a separate part of the project, which aims to solve the same problem with a different approach.

The relevant files are *word_preprocess.py* and *word_learning.py*.

The content *of word_preprocess.py* is responsible for preprocessing the downloaded Reddit comments (supported by *comment_collector.py)* in a manner that it can be used to create a word-based learning on comments later on.

Our goal is to create a bidirectional map of words which maps the words to a one-hot encoded integer representation. Also, each word is paired up with an n-gram context[1], which basically means the words occurred in the same line before the current word.

That way we can predict the next word based on the n-gram context.

Regarding the model itself, we used an LSTM network for learning the coherence of the words, and softmax is used as the activation of the output. The design of the model can be found in word_learning.py. The output of the network is vector of probabilities representing the likelihood of a word coming after the current word in a sentence. As a future plan we would like to take rating of the comments into consideration as well.

The hyperparameters of the model were chosen with a manual approach based on our intuition and several attempts. With adam optimization and categorical crossentropy as a loss function the model seems to work pretty well.

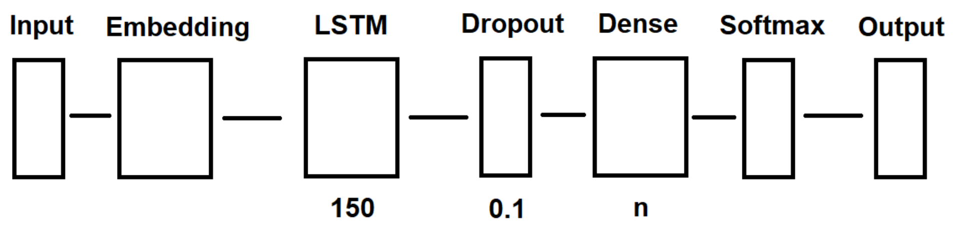

Figure 2: Structure of the model

After training the model for 100 epochs, it generated the following sentence:

Honestly, the camera has always blown my mind. It creates a lasting picture of something that happens in the real world, and definitely a memorable jump level all the calculated stuff now there's a reason it still holds up after all these years from just a spoiler of a little times and

The first few words are given, the generated text starts with *definitely*. At first glance it may seem like proper English, the model learned some common phrases like *still holds up after all these years*, but overall, the text has no meaning. The meaningful parts may be due to overfitting, and copying the exacts same words from the dataset.

Usage:

- Make sure *data/askreddit.json* is available (run *comment_collector.py* if not).
- Run *word_learning.py*. The output of the algorithm can be found in the output.

2.2.2 Word2vec approach

As an alternative to the above two solutions we created a Keras implementation of a word2vec approach which aims to create a comment based on popularity and based on some kind of similarity between two words.

The *word2vec_learning.py* file contains the preprocessing and the learning part of this method as well. The preprocessing part is slightly different compared to the solutions above. From the comments we need the following information:

- An integer representation of each word[8]. To achieve that we created a bidirectional map of integer-word pairs.

- The mean "score" of each word. It means the we calculated the mean popularity of those comments which contains at least once the current word. In order to achieve that it was also needed to calculate the occurrences of the words.

This implementation of *word2vec*[5] uses Negative sampling[1] in order to replace the expensive *softmax* activation with a simple sigmoid activation. We decided to use *Skip-gram*[2] which predicts surrounding context based on the target word. As an alternative we tried out using *CBOW*[2], but Skip-gram turned out to be more effective in this scenario.

As a similarity measure - which is the core of the word2vec learning process - we decided to use cosine similarity score[3], because it is used nowadays in most of the projects with similar problems.

The learning process is the following:

- The network takes two words - represented as an integer - as an input.
- The two integers are converted to a vector representation with the help of a network layer. This is an embedding layer which has a functionality similar to a lookup table. The goal of the network is to teach this lookup table to return "similar" vectors to similar words, and vectors with more distance in case the input words are not similar.
- We create a dot product of the two vectors which represents the two input words. Also calculate the cosine similarity score.
- The output of the network is a simple node with sigmoid activation. The output will be close to 1 if the words were similar. It will be close to 0 in case the words are used in a really different context.

We can use the above network to predict words with similar context. Although this is a big success it is also needed to use up the popularity score of the words too. In order to do that we decided to do the following:

1. Teach the *word2vec* neural network with the preprocessed comments.

2. Start a sentence with a random word. The word "the" is used for testing.

3. Run the prediction on each word in the dictionary paired up with the current word. Find the top n similar words. $n = 8$ is used for testing.

4. Choose the one with the maximum popularity score from the top n similar words. This will be the next word. GOTO 2.

Theoretically the above algorithm can run forever although it will run into an infinite cycle most likely. In order to avoid that we added a somewhat random choosing factor to the algorithm.

After training the model for 30000 epochs, it generated the following sentence:

The felt existence. together, these call. conceivable entirely conceivable support known rendition strange listening quiet prose schizophrenic award basketball jackson spoils

Only the first word is given, but still not meaningful, although it seems that some consecutive words might be really in each others context.

Sometimes, if the model trains for a longer time, then it's generalizing capabilities take controll, like in the following generated sequence:

The you about but it. and we movie but its much is the where is that is that where the that

Usage:

- Make sure *data/askreddit.json* is available (run *comment_collector.py* if not).
- Run *word2vec_learning.py*. It will take a while. One can see the predicted words in the output.

2.3 Summary

None of the above methods worked like we expected, they are still far from being able to create meaningful comments. The word-based model is not scalable and could not work with a bigger dataset. Character based and word2vec both has more potential, but they are also computational heavy, and hard to optimize.

4 Future Plans

The character based and the word2vec has some more potential, we should try to build bigger models and/or train them more.

5 References

[1] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Delia Pietra, Jenifer C. Lai, 1992, Class-based n-gram models of natural language, https://dl.acm.org/citation.cfm?id=176316

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, Jeff Dean, NIPS, 2013, Distributed Representations of Words and Phrases and their Compositionality, http://papers.nips.cc/paper/5021-distributed-representations-of-words-andphrases

[3] P.-N. Tan, M. Steinbach & V. Kumar, 2005, Introduction to Data Mining, Addison-Wesley

[4] Sundermeyer, Martin / Schlüter, Ralf / Ney, Hermann (2012): "LSTM neural networks for language modeling", In *INTERSPEECH-2012*, 194-197.

[5] Elena Tutubalina, Zulfat Miftahutdinov, Sergey Nikolenko, Valentin Malykh, 2018, Sequence Learning with RNNs for Medical Concept Normalization in User-Generated Texts https://arxiv.org/abs/1811.11523

[6] Sepp Hochreiter and Jürgen Schmidhuber, 1997, Neural Computation

[7] Najoung Kim, Kyle Rawlins, Benjamin Van Durme, Paul Smolensky, 2018, Predicting the Argumenthood of English Prepositional Phrases, https://arxiv.org/abs/1809.07889

[8] Afroz Ahamad, 2018, Generating Text through Adversarial Training using Skip-Thought Vectors, https://arxiv.org/abs/1808.08703

[9] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, 2015, Deep learning, https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf