|  |
| --- |
| **Deep Learning - Nagy Házi Feladat** |

**Reddit-BobTheBuilder**

**Oláh GergelyOczot Balázs Bunth Tamás**

**FWXYG1 PRAMRX IWVRAI**

<olahkoo@gmail.com>[*balix18@gmail.com*](balix18@gmail.com)[*btomi96@gmail.com*](btomi96@gmail.com)

[*https://github.com/olahkoo*](https://github.com/olahkoo)[*https://github.com/balix18*](https://github.com/balix18)[*https://github.com/Wastack*](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.

The deep learning based algorithm aims to learn the reasons of popularity of comments. After that it produces popular comments based on the previously learned correlations.

1 Introduction

Natural language processing is a very popular field in deep learning. There are two different approaches to it, one is the character based and the other is the word 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 other approach is a word‑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. This process can be invoked by calling the train\_model() function at the end of the file.

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.

Usage**:**

* Simply run the *character\_based\_lstm\_learning.py* file.

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.

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. 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* 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 it would make sense to add a random choosing factor to the algorithm, but it is not implemented yet.

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

**4 Future Plans**

**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

**Acknowledgments**

[0] https://github.com/leerob/Space\_Invaders

A fenti játékot vettük alapul a megjelenítésünkhöz.

**References**

[1] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, Satinder Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games http://papers.nips.cc/paper/5859-action-conditional-video-prediction-using-deep-networks-in-atari-games.pdf

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou,  
Daan Wierstra, Martin Riedmiller. Playing Atari with Deep Reinforcement Learning  
https://arxiv.org/pdf/1312.5602.pdf

[3] Katrin Becker, J. R. Parker. All I Ever Needed to Know About Programming, I Learned From Re-writing Classic Arcade Games  
<https://pdfs.semanticscholar.org/a721/8b87d575aafec96d8f0cd1f431a5dd2f7fd0.pdf>

[4] Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, Marzyeh Ghassemi. Deep Reinforcement Learning for Sepsis Treatment  
https://arxiv.org/pdf/1711.09602.pdf

[5] Eleni Vasilaki. Is Epicurus the father of Reinforcement Learning?  
https://arxiv.org/pdf/1710.04582.pdf

[6] Mahipal Jadeja, Neelanshi Varia, Agam Shah. Deep Reinforcement Learning for Conversational AI  
https://arxiv.org/pdf/1709.05067.pdf

[7] Matthew D. Zeiler. ADADELTA: AN ADAPTIVE LEARNING RATE METHOD  
https://arxiv.org/pdf/1212.5701.pdf

[8] Yuxi Li. DEEP REINFORCEMENT LEARNING: AN OVERVIEW  
https://arxiv.org/pdf/1701.07274.pdf

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal Policy Optimization Algorithms  
<https://arxiv.org/pdf/1707.06347.pdf>

[10] TensorForce Documentation  
https://media.readthedocs.org/pdf/tensorforce/latest/tensorforce.pdf