Project

**Generating text using LSTM GAN in PyTorch**

**Introduction**

Text generation is one of the common tasks in text mining and natural language processing (NLP). This project focuses on usage of generative adversarial networks (GANs) in the text generation. GANs are usually used for image data, in that case their components are CNNs, for working with text, RNN architecture should be used.

The second assumed model is a Markov chain model, a simple model that can perform satisfactory on a sufficiently big training dataset. Predictions of the model are then classified by the discriminator to directly measure its performance and thus indirectly evaluate the quality of the text generated by the GAN. Absolute quality of the generated text is also evaluated manually by a human using a labeling of fakes amongst the real comments.

Each training sample contains a flag, marking binary presence of negative sentiment in the text, such as racism or attack. Even though this information is not utilized in the models in any way, its possible usage as an additional input to the generator network is discussed and analyzed.

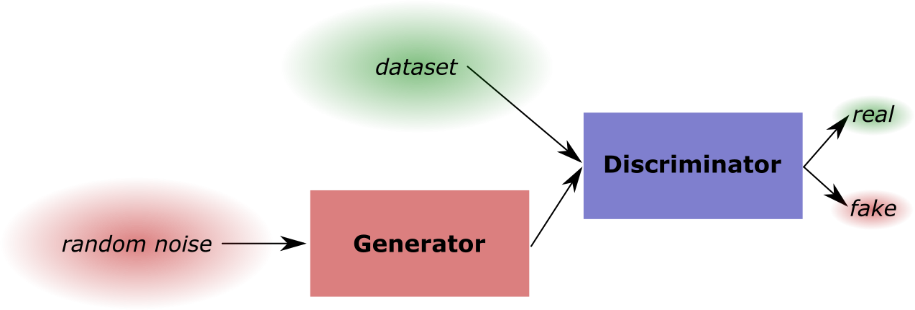
**Theory**

In the statistical approach the generated text is prediction of a certain model, fitted to train data. A kind and quality of the training set and the chosen model are two important factors for the quality of the generated text.

Recurrent neural network (RNN) is a deep learning model for sequential data, such as time series or natural language, whose elements are mutually related and order dependent. RNNs are sequence length independent. The inputs of the network for each time step are the network input and internal network state of the previous time step , the outputs are the network output and internal network state utilized in the next time step. The weights to be trained on data are (transforming ), (transforming ) and (transforming ). The is a non-linear activation function.

One of the problems in deep learning is vanishing/exploding gradient, the numeric instability during the backpropagation. An alternative to RNN solving this issue is using the LSTM architecture of a neuron, which contains a memory cell improving numerical stability. (Vasilev I. 2019)

Generative Adversarial network (GAN) is very popular model for data generating. Its results, forged images and videos called *deep fakes*, are well-known even by the lay public, but it can be used for generating any kind of data. GAN framework consists of two networks: generator and discriminator , two networks reciprocally training each other. produces data as similar as possible to the training dataset, as an input it uses random noise. It is trained using , that learns to classify the generated samples shuffled among the training samples (binary label true/fake). is backpropagated, then labels for are produced from predictions, the misclassified samples are labeled as correct. This training cycle is repeated until the losses of and converge, same as in case of single network training. (Goodfellow I. 2014)



The second assumed model for comparison and performance evaluation is a Markov chain model, based on principles of Hidden Markov model (HMM). Training the model means constructing the *next-word* distribution for each word of the vocabulary or transition matrix in terms of Hidden Markov models. There are several methods for simulation from the Markov chains: filtering, smoothing and Viterbi algorithm. Filtering uses only samples prior the current time step of simulation (), smoothing and Viterbi algorithm use all the samples (), the latter is the only one producing valid output according to the transition matrix. (Fraser 2008)

Sentiment is an emotion or feeling contained in the text connected with an opinion of the author. Categories of sentiments are connected to the human emotions, they might be oriented positively, neutrally or negatively and a certain intensity. Sentiment might have an emotional or rational origin. The sentiment analysis is a task to discover sentiments and through them opinion of the author. This opinion is the reason for the observed sentiments. (Cambria E. 2017)

**Data**

The dataset was published on [figshare.com](https://mendeley.figshare.com/articles/dataset/Cyberbullying_datasets/12423407) as part of online *datathlon* [Data Sprint #13: Cyberbullying](https://dphi.tech/practice/challenge/42#data) held by *DPhi* community. The data collection consists of 8 csv files of overall size 171 MB, containing user comments from various social network platforms like Kaggle, Twitter, YouTube and Wikipedia Talk. The data contains text and binary flag marking the presence of cyberbullying or sentiment in terms of text mining. Cyberbullying aspects include behavior such as hate speech, aggression, insults and toxicity. (Elsafoury 2020)

The data files are downloaded and merged as a single format. The samples are deduplicated with text equality since the files have heavy overlap. All sentiment flags are unified as a single binary cyberbully-present flag. Then each text is tokenized and stored in the file data/words.txt. This data-engineering step takes about 45 minutes and is being done separately in src/fetch.py and executed out of the training pipeline.

File data/words.txt contains 196292 distinct comments and 3 attributes: text (stringified list of words), label (0/1 sentiment flag) and source (twitter/youtube/kaggle/unknown). An example line of the file is

|  |
| --- |
| "['able', 'to', 'do', 'better', 'than', 'that']",0,unknown |

The word is mapped to numeric value as an index of word2vec in gensim package. This makes every different word distinct including case-differences and a word is represented as a scalar, integer. On the other hand, the mapping removes all the semantic connections such as synonyms capturable by more complex mappings. The word2vec model is trained on the data, the vocabulary size is 307967.

Last step in the input data processing is pad each comment to fixed sequence length. By observing the word count distribution. From the plot it is apparent that most of the comments have less than 500 words. If all sentences longer than 1200 words are dropped, memory requirements and computational complexity are drastically reduced.

|  |  |
| --- | --- |
| *Sentences of all lengths* | *Sentences longer than 1200 dropped* |
|  |  |
|  |  |
|  |  |
|  |  |

If different mapping is used (e.g. 100 feature vector per word), the memory requirements are even greater.

**Method**

For documentation of the data and the architecture of the models several parameters is used. The longest sentence in the training dataset is words long and the training dataset contains words. The single word has , so it is represented by a scalar value. Network architectures are partially parameterized with and for both Generator and Discriminator. Parameters of the training itself are batch size , learning rate and number of epochs .

Generator neural network input and output shapes are . First two layers are LSTM, making the Generator independent on the with hidden size of . The output layer is dense with neurons, connecting all the time step outputs of the previous LSTM layer.

Discriminator neural network input is  and output . Its structure is almost the same as the one of Generator with one extra dense layer at the end with neurons, so the network produces only value per sample. The last layer requires constant, previously defined of each sample, so the sentences must be aligned to it by padding.

Generator

Discriminator

GAN – executed on Google Colab

Contra-technique - HMM generator

(Liu 2020)(Mosquera 2018)

(Inkawhich 2017)(Robertson 2017)

**Results**

Embedding mapping [] of the input text data for the first trained model was implemented with a simple incrementation method “*as word occurs*”. The words get theoretically random numbers assigned, which is probably a reason, why discriminator fails to distinguish between the test data and the data generated by Markov Chain generator. The generator results are classified correctly.

whoamiyouare

Therefore, embedding strategy putting semantically similar words closer to each other was used. For semantical closeness *word2vec* from package *gensim* was used.

Evaluation of the results

**Discussion**

*Most of work should be here. Present your analysis of the results that you obtained, discuss the possibilities and limitations of your approach and compare your study to related work.*

According to the experiments, embedding strategy is an important part of the modelling and a wrong decision ends up in unsatisfactory results.

**Conclusion**

*Built on the discussion – answer the question what new knowledge you take away from your project.*

**Resources**

Cambria E., Das D., Bandyopadhyay S., Feraco A. 2017. „A Practical Guide to Sentiment Analysis.“ Cham: Springer. doi:10.1007/978-3-319-55394-8.

Elsafoury, Fatma. 2020. „Cyberbullying datasets.“ 6. doi:10.17632/jf4pzyvnpj.1.

Fraser, Andrew. 2008. *Hidden Markov models and dynamical systems.* Philadelphia: Society for Industrial and Applied Mathematics. doi:ISBN 978-0-898716-65-8.

Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y. 2014. „Generative Adversarial Networks.“ Montréal, 10. 6. doi:1406.2661.

Inkawhich, Nathan. 2017. „DCGAN Tutorial.“ PyTorch. https://pytorch.org/tutorials/beginner/dcgan\_faces\_tutorial.html.

Liu, Kanghui. 2020. „Pytorch 20: Implement RNN(LSTM) Generate Random Text.“ 28. 6. https://www.bigrabbitdata.com/pytorch-20-implement-rnnlstm-generate-random-text/.

Mosquera, Diego Gomez. 2018. „GANs from Scratch 1: A deep introduction. With code in PyTorch and TensorFlow.“ 1. 2. https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f.

Robertson, Sean. 2017. „NLP From Scratch: Generating Names with a Character-Level RNN.“ PyTorch. https://pytorch.org/tutorials/intermediate/char\_rnn\_generation\_tutorial.html.

Vasilev I., Slater D., Spacagna G., Roelants P., Zocca V. 2019. *Python Deep Learning.* 2nd Edition. Birmingham: Packt Publishing. doi:978-1-78934-846-0.