**Generating text using LSTM GAN in PyTorch**

732A92 Text Mining

Martin Beneš [marbe619]

2021-01-16

**Abstract**

The aim of the project is to generate text using generative adversarial networks (GANs), where both components are recurrent/LSTM neural networks. The training data set is a collection of internet comments from YouTube, Twitter and Kaggle also containing a (binary) negative sentiment indicator. The model output is evaluated using the result of Markov chain trained on the same data. Absolute quality of the text is evaluated manually against a test set. Sentiment input integration into LSTM GAN is discussed.

**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.

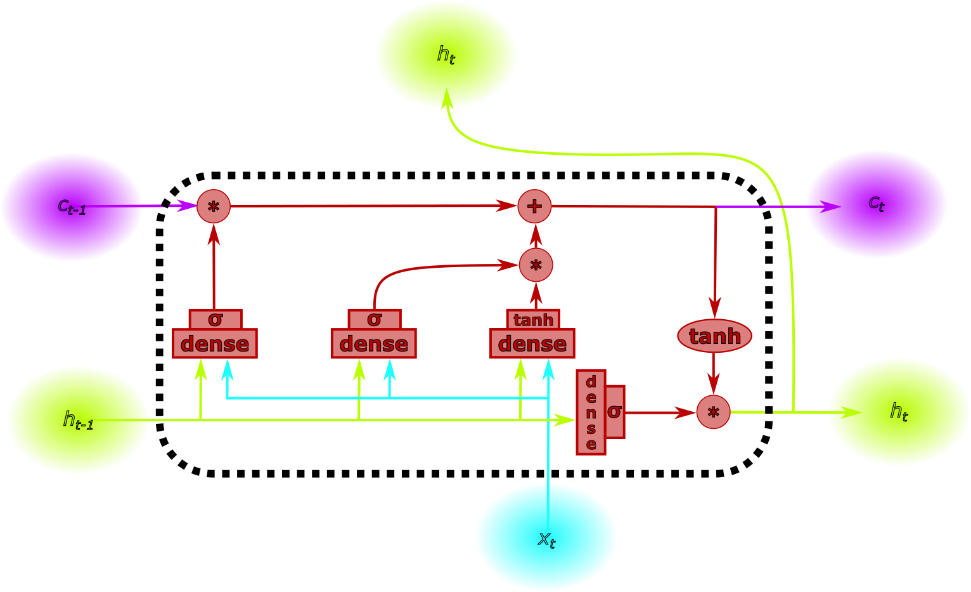
Each training sample contains a flag, marking binary presence of negative sentiment in the text, such as racism or attack. Even though this information is not utilized in the models in any way, its possible usage as an additional input to the generator network is discussed and analyzed.

**Theory**

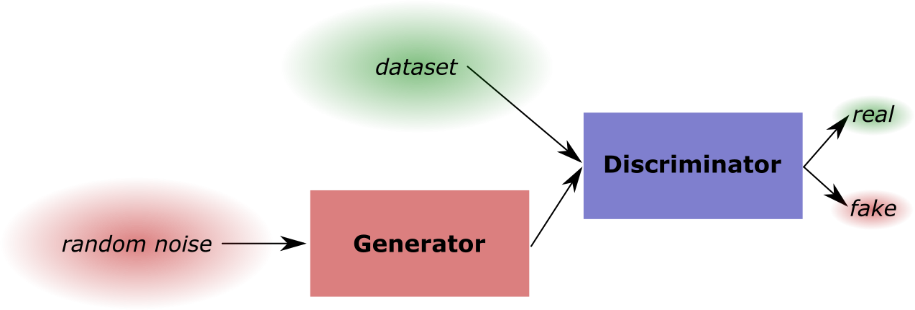
In the statistical approach the generated text is prediction of a certain model, fitted to train data. A kind and quality of the training set and the chosen model are two important factors for the quality of the generated text.

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 *cyberbully* 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

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

Right before the training a mapping using embedding is done, a projection of the text to a numeric tensor (e.g. vector or matrix). This paper uses three different methods: scalar embedding, word2vec and Bert. The former two introduce the same computational issue: to construct the models (especially Discriminator), the length of the sentences should be constant and the insufficiently long sentences are padded with special <*not-a-word*> token.

However the distribution of dataset has several extremely long samples (the longest is words long), while most of the sentences have at most several hundred words. Padding brings a gigantic overhead for the training data storage. Thus extreme sentences are dropped in order to normalize the to a bearable value; if sentences longer than words are ignored, memory requirements and computational complexity is 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. For instance embedding with word2vec strategy amplifies this problem in relation with used , its output dimension. Thus, embedding vector mapping is computed in each iteration to avoid massive memory usage. This way each batch is passed through embedding mapping times.

**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 differs for each embedding strategy: for the scalar incremental it is , for word2vec it is and for Bert it is variable based on the text (cropped to ) (**update**). Network architectures are partially parameterized with and for both Generator and Discriminator, the dropout probability for . Training configuration includes batch size , learning rate and the number of epochs .

As mentioned in the previous section, this paper uses three different methods: scalar embedding, word2vec and Bert. The scalar embedding supposes that words can be simply projected to a single number – this in general does not project any semantical similarities into the embedding representation and ptus all responsibility on the neural network to learn as much as possible from the dataset.

Second embedding strategy uses pretrained word2vec model from gensim package with vector length . This model has trained the vector representation for the words on a many various datasets and hence projects the similar words closer to each other (in cosine distance). Disadvantage of this mapping is fixed vocabulary. (Mikolov 2013) Walkaround for this bottleneck is “closest word” matching using string matching from *rapidfuzz.*

A strategy not having the property of constant vocabulary size is Bidirectional Encoder Representations from Transformers (abbr. Bert), which as word2vec uses Transfer learning. In addition, unlike the two embeddings before, Bert works on a sub-word level: it does tokenize the text based on its vocabulary of words and sub-words, not based on the spaces. This representation is in particular helpful in the such cases as the data used in this paper, since internet communication is very specific - informal words, emojis, slang/argot, etc. (McCormick 2020)

Generator neural network input and output shapes are . First two layers are LSTM, making the Generator independent on the with hidden size of . Output of the last LSTM is flatted along the single sentence and fed to output dense layer with neurons and output , returned reshaped to .

Discriminator neural network input is  and output . Its structure is fairly similar to the Generator’s. Two LSTM layers of the same shape are in Discriminator followed by dropout, after them follows a single hidden dense layer with dropout and ReLU activation function and nodes. The final output dense layer has nodes and an output size , activated by sigmoid to produce value between (fake) and (real). This final layer brings the requirement for constant and is the reason why padding alignment had to be used in the training data. (Liu 2020)(Mosquera 2018)(Inkawhich 2017)(Robertson 2017)

Markov Chain model can be understood as a special case of Hidden Markov model, where the latent space is vocabulary of the training set and the emission matrix is the identity of appropriate shape. After experiments with library hmmlearn, built on scikit-learn, it was discovered that MultinomialHMM is not suitable for such usage in text mining, because transmission matrix is implemented as a NumPy matrix a.k.a. dense matrix. Hence, memory requirements are , where is vocabulary size, in case of the training set *cyberbully* , and grow quadratically.

Instead, a custom Markov Chain model with sparse transition matrix was implemented as a class MarkovChain in the file [src/markov.py](https://raw.githubusercontent.com/martinbenes1996/732A92-project/main/src/markov.py). For each sentence, let us denote its ith word as , number of its occurrences and a bigram of words and as . Then the probabilistic definition of the implemented Markov Chain model is

**Results**

The Markov Chain model yielded results saved in output/markov.txt. For a human, the sentences do not make sense – the only dependence is a probabilistic measure of following word, which makes the generated sentence changing the point every few words for a human reader.

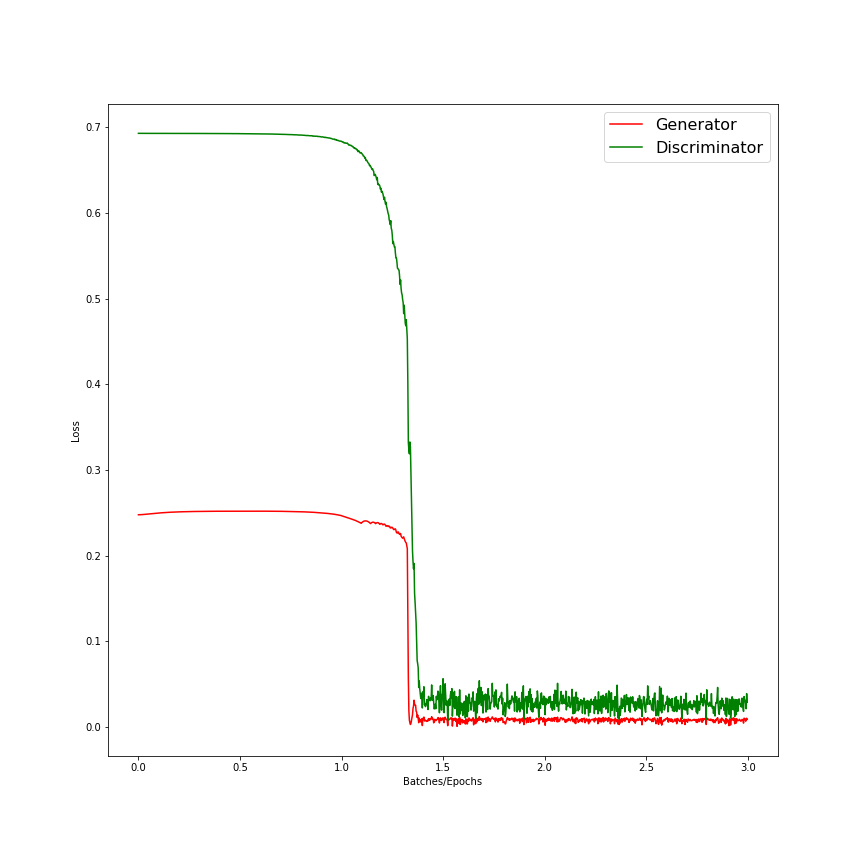
|  |
| --- |
| capetown and figuratively to statements properly as part of VandalProof SpK. as sarcasm is essential condition is denied |

|  |
| --- |
| remainings in Western Australia and the Omagh bombing of something Wikipedia poster show actual 10-string guitar |

|  |
| --- |
| abisharan talk page if they have little insignificant instances of vandalism work on WSD closed minded |

Scalar embedding mapping [] of the input text data for the first trained model was implemented with a simple incrementation method “*as word occurs*”.

whoamiyouare



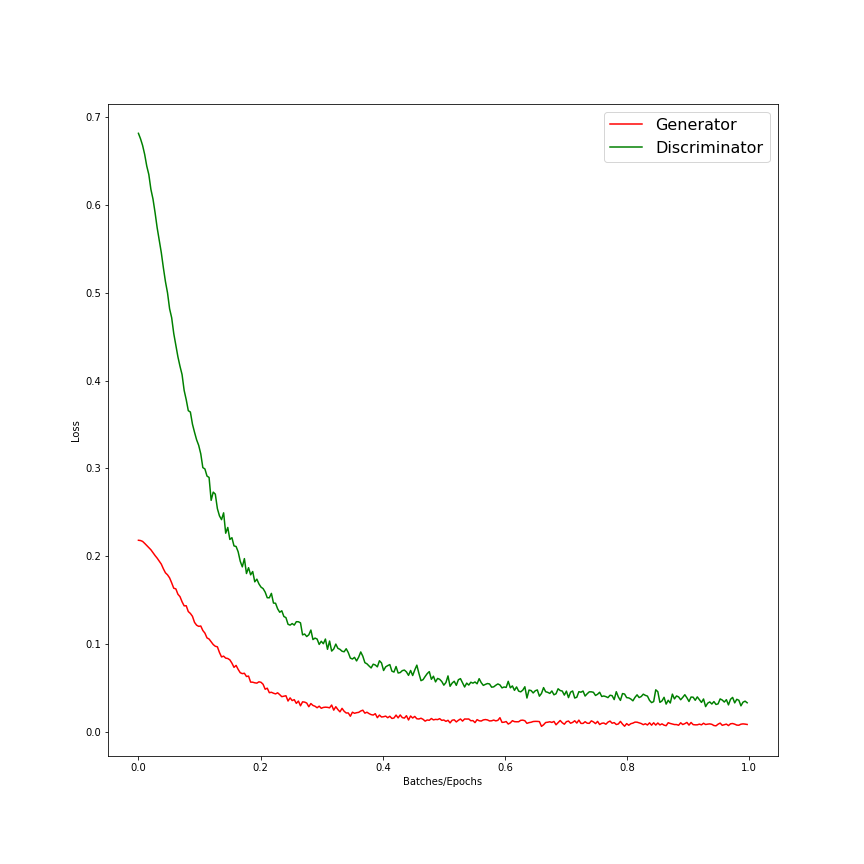
During the second epoch a point in gradient is reached when errors of Generator and Discriminator abruptly fall towards zero. This is not a good sign of training. After that, the errors do not seem to be changing.

The model performance on test data, generator output and reference Markov chain are shown in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Results of the model with scalar incremental embedding** | | | |
|  | Test data (real) | Generator (fake) | Markov chain (fake) |
| Accuracy (threshold 0.5) |  |  | 0.7% |
| MSE (soft output) |  |  |  |

The model performs extremely well on test data and on generator output but incorrectly classifies MC samples as real.

Embedding using word2vec comes with pretrained model and fixed vocabulary. Missing words are replaced with the closest word strategy by string matching from *rapidfuzz*.



Training time of the network gets longer: not only each training iteration contains the embedding of the batch, but the network gets more complicated due to increase too. A single training epoch takes about minutes, which is why had to be reduced to . The errors gradually descend towards zero – this is a good sign for the training, since abrupt change might mean a local minimum, while gradual change means there is a significant extreme point the gradient descends to.

The model performance on test data, generator output and reference Markov chain are shown in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Results of the model with word2vec embedding** | | | |
|  | Test data (real) | Generator (fake) | Markov chain (fake) |
| Accuracy (threshold 0.5) |  |  | 0% |
| MSE (soft output) |  |  |  |

The model results are very similar to the previous ones, the model performs extremely well on the fake data produced by the Generator but fails in classifying the fake data from the Markov chain.

The dataset cyberbully contains data that are quite messy when a word separation by spacy is used. There are many misspelled words, informal words and thus the closest word matching might have been a reasonable solution. Alternative solution is using sub-word embedding Bert.

BERT

BERT

BERT

BERT

BERT

BERT

BERT

BERT

BERT

BERT

BERT

BERT

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.*

The model with scalar incremental embedding distinguishes very well between the generator output and the real data.

According the loss plot the model converged, and by using a different output of the generator and a test data the results of the estimation come with higher reliability of the measured performance. We can postulate that the model generalized above the training dataset well from these. Even though the words get (theoretically) random numbers assigned by the embedding, the GAN flexibly trained to distinguish real data from the ones produced by it.

However, regarding the performance on independent text generator, we observe that the almost all artificially produced samples from Markov Chain model were classified as real. From this it may be assumed that the discriminator considers the samples to be closer to the true data than what Generator is producing or that the Discriminator relies on features contained in the Generator output, that are not present in the output of the Markov Chain generator. The model fails to extrapolate ability to distinguish the fake data out of the Generator output, which might imply that the Generator output is not of high quality.

Possible alternations to the scalar incremental model could be assigning closer numbers to words similar by meaning, but such seriation would be computationally intensive when vocabulary gets great, the exhaustive list of all possible permutations has size for vocabulary of size .

The model using word2vec embedding has results very similar to the model with scalar incremental embedding for all three groups. In this case the model is more decided about the groups (MSE) and thus has higher error for MC, even though the was decreased to , which should reduce the risk of overfitting.

In case of word2vec one could use more features to better distinguish between the words, but since the model fails entirely to discriminate the external fake artificially generated text, a good idea could be extending the GAN framework by adding more fake data sources, e.g. output of various generative models, which could make the framework better extrapolate out of the Generator output; from the models we could name GAN instances fitted to different datasets or reinforcement learning generators, based on syntactic learning. (Chen 2018)

Bert

The bottleneck of working with Generative Adversarial Network are high computational and resource requirements. The training has been performed on Google Colab in a form of Jupyter Notebook. Usage of GPUs however is limited for time and direct activity (interaction) with the page. Thus, the development is very slow and the long-running tasks must be carefully planned and can not be ran without supervision, e.g. over night. (Google 2020)

**Conclusion**

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

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

A single GAN with word2vec embedding learns successfully to distinguish its own output from the true data but it does not perform that well with artificial data from different text generator. This data could form a second dataset to raise the performance of the GAN and make it better extrapolate out of the Generator output.

**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.