Federal State Autonomous Educational Institution for Higher Education
National Research University Higher School of Economics

Faculty of Computer Science
Educational Program
Applied Mathematics and Information Science

TERM PAPER

Research project

"Application of the Pretrained GPT Model in Natural Language Processing"

Prepared by the student of group 182, 3rd year of study,
Kotelnikov Akim

Supervisor:
invited teacher, Simagin Denis

Moscow 2021

Contents

1 Abstract

1.1 Abstract

Deep learning methods are widely used in various areas of life. A lot of deep learning methods help to solve complex problems and some of these methods outperform previous techniques. One of the most popular application of deep learning is Natural Language Processing (NLP). This work has two main goals: to implement model that detects humor using ruGPT3(Russian-GPT3)[1] which is based on Generative Pre-trained Language Model (GPT-2 (Alec Radford et al. (2019))) and compare it with other solutions; to implement the model that generates Russian jokes using ruGPT3 model.

1.2 Аннотация

Методы глубинного обучения активно используются в различных сферах человечкой жизни. Они помогают решать сложные задачи и превосходят в качестве уже существующие подходы. Одной из таких задач является обработка естественного языка (Natural Language Processing). Данная работа имеет две основные задачи: с помощью модели ruGPT3 (Russian-GPT3), основанной на Generative Pre-trained Language Model (GPT-2 (Alec Radford et al. (2019))), решить задачу определения юмора и сравнить полученный результат с другими решениями данной задачи; создать модель на основе ruGPT3, которая генерирует шутки на русском языке.

2 Key words

NLP, jokes, Transformers, GPT, ruGPT, unsupervised learning, text, humor detection, text generation

[1]https://sbercloud.ru/ru/warp/gpt-3

3 Introduction

3.1 Subject of research. Relevance.

There are many tasks related to NLP, e.g. text generation (generate text based on the sentence), text classification (spam detection) , translation, gap-filling (fill in the missing word in the sentence), etc. Some of these tasks can be solved using word embeddings, such as word2vec (Tomas Mikolov et al. (2013)) or GloVe (Global Vectors) (Jeffrey Pennington et al. (2014)), or using RNN (Recurrent Neural Network) and seq2seq (sequence-to-sequence) model (Ilya Sutskever et al. (2014)). But there is a significant progress in text recognition and text generation tasks due to the emergence of Transformer (Ashish Vaswani et al. (2017)) and the possibility to pre-train it on unlabeled dataset. So, the GPT model (Alec Radford et al. (2018)) that is based on Transformer has achieved state-of-the-art results on a suite of diverse language tasks, e.g. Textual Entailment and Commonsense Reasoning.

Humor has a great influence in human life. This is an important part of communication, which makes it easier to communicate with each other and helps to establish relationships with other people. Humor detection and generation of humor can help chatbots and virtual assistants to become more human. For example, virtual assistant should response differently depending on whether the user joking or not. The simplest structure of the joke consists of two parts: *set-up* and *punchline*. "Set-up" is the phrase that literally setting up the joke and "punchline" is the phrase or word (or even just a sound) that ends the joke and makes it funny. There are many reasons, why punchline makes people laugh, e.g. the element of surprise or the wordplay. When structure of the joke is so simple, it is expected that model can understand word dependencies and learn how to detect humor. As for long jokes, the structure is not as simple, thus the problem is more complex, the dependencies between words are more complex and the reasons, why people laugh, are incomprehensible.

3.2 Goals and objectives of the work

The first goal of the work is to compare transformer-based classification models (ruGPT3, especially) with the earlier solutions for humor detection problem.

The second goal is to implement generator of the Russian jokes using the pre-trained ruGPT3 model. We will train different sizes of ruGPT3 models and then compare them. Also we will train model that will determine whether the joke is real or generated by ruGPT model. This will lead us to the task of implementing the Generative Adversarial Network (GAN) model.

4 Main part

4.1 Related works

4.1.1 Attention

Attention technique is very important to understand how Transformer works. The first time attention (Dzmitry Bahdanau et al. (2014)) was introduced in order to improve seq2seq (Ilya Sutskever et al. (2014)) model. Attention is the way to build a connection between encoder states and state of decoder at the current step. Briefly, at each decoder step we compute a score between the current decoder hidden state and all encoder hidden states, then we apply softmax to these scores to get vector of attention weights and, finally, we compute the weighted sum of the scores and corresponding attention weights. These technique partially solves the main seq2seq problem: impossibility of decoder to focus on the different parts of the input sentence at each step.

4.1.2 Transformer

Transformer (Ashish Vaswani et al. (2017)) is also an encoder-decoder paradigm, but unlike seq2seq model, encoder and decoder of Transformer consists only of attention blocks and feed forward blocks (feed forward block is just two linear layers network). It is important to note, that in Transformer's encoder tokens interact with each other all at once, while seq2seq model understand this only when it has received the entire sentence. Also, Transformer's encoder has self-attention and decoder has masked self-attention. Self-attention is attention between input tokens, whilst masked self-attention is restriction of the decoder to look at the next words (so, for each word attention is applied to all previous words). Since encoder receives all tokens at once, Transformer somehow needs to take into account the position of the words in the sentence. In order to do that, the authors of the Transformer apply positional encoding. So, the main advantage of Transformer over seq2seq model is that Transformer can see the whole sentence at once and constructs dependencies using self-attention. Also Transformer has lower computational complexity compared to the RNN models. Transformers and attention mechanisms perform state-of-the-art scores in many NLP tasks (in original paper the authors provide results about machine translation and English constituency parsing). Read the original paper (Ashish Vaswani et al. (2017)) in order to see the full Transformer architecture.

4.1.3 Generative Pre-Training (GPT) model

GPT model (Alec Radfford et al. (2018)) was introduced by OpenAI research group and achieved state-of-the-art results on many sentence-level tasks from the GLUE benchmark (Alex Wang et al. (2018)). The architecture of the model is a 12-layer Transformer decoder (without encoder-decoder attention). The model is trained with the standard cross-entropy loss and at each step predicts a probability distribution of the token under the condition of the previous predicted tokens in some context window. This model can be easy adapted to supervised target tasks, such as sentence classification, sentence entailment, sentence similarity, etc. The OpenAI GPT-2 model (Alec Radford et al. (2019)) largely follows the details of the OpenAI GPT model. The main difference is in normalization layers. Also GPT-2 has different variations depending on the size of the model. It is important to note, that GPT-2 model achieves state-of-the-art scores on a variety of domainspecific language modeling tasks, although this model was not trained on any of the data specific to any of these tasks.

4.1.4 ruGPT3

ruGPT3 is the model based on the GPT2 original model and it was pretrained on Russian texts (e.g. Russian Wikipedia, fiction, dialogues). The model has several variations (small, medium, large) depending on the size of the model. As for ruGPT3Large model, it was trained with sequence length 1024 using transformers library[2] by SberDevices[3] team on 80 billions tokens for 3 epochs.

4.1.5 Bidirectional Encoder Representations from Transformers (BERT)

BERT model (Jacob Devlin et al. (2018)) is also a language model based on Transformer. BERT model architecture is just a Transformer encoder, but it is interesting how model is trained. The BERT's authors came up with two training objectives for unlabeled data: Next Sentence Prediction (NSP) Objective and Masked Language Modeling (MLM) Objective. The NSP objective is binary classification task, namely to determine whether the two given sentences are consecutive. The main purpose of the MLM objective is that model try to predict masked words in the given sentence (some of the words are specially masked before being fed into the model). For more details, see the original paper. The BERT model outperformed GPT model on GLUE datasets.

4.1.6 ruBERT

ruBERT (Yuri Kuratov et al. (2019)) is BERT model that was trained on Russian Wikipedia and Russian news data by DeepPavlov[4] group. Model has 12 layers of transformer's encoder, 12 attention heads and 180 million parameteres in total.

[2]https://huggingface.co/transformers/
[3]https://sberdevices.ru/
[4]https://deeppavlov.ai/

4.1.7 ColBERT: Using BERT Sentence Embedding for Humor Detection

The authors of the paper (Issa Annamoradnejad et al. (2020)) proposed the baselines for the humour detection problem and the model based on BERT that detects humour with very high accuracy. The main idea is to develop model that consider account the whole joke as one sentence and at the same time process two parts of the joke (set-up and punchline) separately, as two different sentence. This idea allows model to find both the sentence-level dependencies and word-level. They achieved 0.98 accuracy on the Joke dataset compared to 0.91 accuracy with XLNet model (Zhilin Yang et al. (2019))

4.1.8 Humor detection in Russian

In (Ermilov et al. (2018)) paper authors solved the humor detection in Russian task using different types of word and/or sentence embeddings and Support Vector Machines (SVM) algorithm. Afterwards, Blinov et al. (2019) expanded the dataset that was used in previous paper and they tested a new ULMFiT (Howard et al. (2018)) model for text classification (humor detection) and compared it with the SVM solutions.

4.1.9 Humor generation

Humor generation is not well researched field yet, but we found one work (Joswin et al. (2019)) whose authors fine-tuned GPT2 model in order to improve classifier that learns to distinguish between jokes and non-jokes. Also they analyzed the attention mechanism difference in jokes and non-jokes sentences.

4.1.10 Text-GANs

It was shown (Caccia et al. (2020)) that MLE models still outperform text-GANs. But nevertheless, one of the most popular text-GANs is RelGAN (Relational GAN, Nie et al. (2019)). This GAN consists of relational memory generator. The main idea of relational memory is to consider a fixed set of memory slots and allow for interactions between memory slot sby using the self-attention mechanism. The RelGAN's discriminator is CNN classifier that uses multiple embedded representations . In more details, discriminator maps input (output of generator: $y \in \mathbb{R}^{max\_len \times |vocab|}$) into S multiple representations using $\{W_e^{(s)}\}_{s=1}^{S}$ matrices and then independently passes these embeddings through CNN classifier. The final loss is an average of S losses. To deal with the non-differentiablity problem, the authors used Gumbel-Max trick:

$$y_{t+1} = one\_hot(argmax_i(o_t^{(i)} + g_t^{(i)})) \tag{1}$$

where o_t is generator's output logits, $y_{t+1} \in \mathbb{R}^{|vocab|}$ next token prediction and $g_t^{(i)} \sim -\ln(-\ln Uniform(0,1))$. But since $argmax$ is non-differentiable function, it is approximated using temperature τ and softmax function $\sigma(\cdot)$:

$$y_{t+1} = \sigma((o_t + g_t)/\tau) \tag{2}$$

We are going to use RelGAN's discriminator architecture and Gumbel-Max trick in our GAN implementation.

4.2 Our work

4.2.1 Data

First of all, we used FUN dataset from Blinov et al. (2019) to compare BERT-based and GPT-based classifiers with authors' solutions. This dataset consists of the train (250k texts), test (60k texts) and gold (1.6k human evaluated texts) datasets. By far, this is the largest Russian jokes dataset, in which lexical diversity between jokes and non-jokes is relatively small.

Although FUN dataset is very good for humor detection, the most of the jokes are quite long (average length is 20 words and maximum length is 78 words). We think that long length can be a problem for humor generation task. So we decided to construct our own dataset with short jokes. We collected approximately 120'000 jokes from nekdo.ru and humornet.ru web-sites using BeautifulSoup[5] library. The maximum length of the jokes does not exceed 40 words and the average length is 7-8 words. Unfortunately, these jokes have random structure and do not follow set-up/punchline pattern (FUN jokes also do not). As for negative class (not jokes), we found Russian news dataset[6] and took 120'000 titles. The average length of the titles is 6-7 words.

4.2.2 Humor detection (classification)

All the code was written using PyTorch, scikit-learn and transformers libraries. You can find our code and our new dataset here[7].

Starting with the FUN dataset, we implemented two models: BERT-based and GPT2-based classifiers. We used pretrained ruBERT model by DeepPavlov. We added a single linear layer that takes as an input the last hidden-state of the first token in sequence ([CLS] token) and outputs vector of size 2. The authors of the BERT model claim that this first token contains information about the whole sentence. Also we used ruBERT tokenizer. We chosed the batch size 64, AdamW optimizer with learning rate $4 \cdot 10^{-5}$. The model was trained for 2 epochs (65 min per epoch) on a single GPU (Quadro P5000). We achieved 0.910 F1-score on the test dataset and 0.886 F1-score on the gold dataset. So, ruBERT classifier outperformed all previous solutions proposed in Blinov et al. (2019) on the test dataset (but not the gold one). All results can be seen in Table 4.1. It is worth noting, that after the second epoch accuracy and F1-score haven't increased significantly, so one epoch is enough. Then we trained ruGPT3Large classifier. We added one linear layer that takes final token of the embeddings sequence as an input. Since GPT model process the sentence from left to right, the last token contains information about the whole sequence. We used the same hyperparameters as we used for ruBERT classifier. But instead we trained this

[5]https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[6]https://www.kaggle.com/yutkin/corpus-of-russian-news-articles-from-lenta
[7]https://github.com/rotot0/jokes-project

model only for 1 epoch, which took approximately 150 minutes. We achieved 0.936 F1-score on the test dataset and 0.901 F1-score on the gold dataset. And ruGPT3Large classifier outperformed ruBERT classifier and all previous solutions on the test and on the gold datasets. This result completes our work with FUN dataset.

Model	**Test F1-score**	**Gold F1-Score**
Baseline SVM	0.798	0.803
ULMFun	0.907	0.890
ruBERT	0.91	0.91
ruGPT3Large	0.936	0.901

Table 4.1: Classification models summary, FUN dataset

Then we moved on to the our dataset and did the same. First of all, we split data into train, validation and test dataset in 3:1:1 proportion. For the baseline we chose SVM classifier with RBF kernel. We lemmatized text and removed stopwords using Natural Language Toolkit (NLTK)[8] library. Then we applied CountVectorizer and obtained matrix with 48'000 rows (size of vocabulary) and 160'000 columns (size of the train dataset). Training process took around 65 minutes on CPU. We did not tune hyperparameters and achieved 0.964 accuracy and 0.956 F1-score on the test dataset. Such good performance may be due to the big difference in vocabularies (jokes vocabulary and news vocabulary).

The next step was to train ruBERT model. For training we used a batch size of 32 and AdamW optimizer with learning rate 10^{-5}. We trained the model for 2 epochs and achieved 0.999 accuracy on the validation dataset. The training process took around 140 min (70 min per epoch) on a single GPU (Quadro P5000). On the test dataset we got 0.998 accuracy and 0.997 F1-score. It is worth noting that after the first epoch, we have already achieved 0.997 accuracy and 0.996 F1-score.

After that, similarly to the FUN dataset, we trained ruGPT3Large classification model. We trained this model with batch size 32 and Adam optimizer with

[8]https://www.nltk.org/

Model	Accuracy	F1-score	Training time
CountVect + rbfSVM	0.964	0.956	65 min
ruBERT	0.998	0.997	70 min/epoch
ruGPT	0.997	0.997	110 min/epoch

Table 4.2: Classification models summary

learning rate 10^{-5} for 2 epochs. Training process took 220 minutes (110 minutes per epoch). This model achieved 0.997 accuracy and 0.997 F1-score on the test dataset.

Since the accuracy is very high, it is interesting to check what sentences are missclassified by ruBERT and ruGPT3 classifiers (Table 4.3 below). Usually model missclassifies jokes (i.e. real jokes that model classifies as non-jokes) that consist of political and formal words. However it is hard to explain why models classify some news titles as a jokes. Also there is nothing special in those sentences that ruBERT and ruGPT3Large models classify differently, so we do not provide them.

Sentence	True label
Трое неизвестных отобрали у прохожего паспорт и порвали. Неизвестных стало четверо	joke
Как поступить с хакером: Интернет голосует	not joke
Из-за аномально тёплой зимы вот уже пятое заседание не спит Медведев	joke
Новым генпрокурором станет нынешний руководитель аппарата правительства	not joke

Table 4.3: Examples of missclassified sentences. The first two for ruBERT, the other two for ruGPT3

As a conclusion for this part (Table 4.1 above), ruBERT model showed better results compared to ruGPT3Large model. Although difference in accuracy and F1-score is not significant, the ruBERT model took less training time.

4.2.3 Generation task

For jokes generation we trained two models: ruGPT3Small and ruGPT3Large. The main differences are in the sizes of the models, i.e. number of transfomer's decoder layers (12 and 24), number of attention heads (12 and 16), hidden size (768 and 1536).

We took the jokes that we collected by ourselves and added two special tokens: the beginning of the joke and the end of the joke. Fine-tuning process is just a training process using MLE (Maximum Likelihood Estimation), i.e. it just trains to predict next token using Cross-Entropy loss. We used the same hyperparameters for both models: batch size 1, Adam optimizer with learning rate $2 \cdot 10^{-5}$. Also we used gradient accumulation technique and gradient clipping technique. It is important that due to the lack of VRAM we had to use batch size=1 (and we used it for the smaller model because of better comparison with the larger one). ruGPT3Small model was trained for 5 epochs (55 min per epoch). ruGPT3Large model was trained for 3 epochs (3h20min per epoch). All train loss graphics can be seen in Appendix. Generating new sentences can be done using different methods. The simplest one is greedy search, but this method leads to many repetitions and lack of diversity. These problems can be solved using Beam search (Markus Freitag et al. (2017)) and top-k (or top-p) sampling. In short, beam search keeps the most likely *num_beams* of sequences at each time step and eventually choosing the sequence that has the overall highest probability. Top-k and top-p sampling is used for diversity generation: at each step we pick not the best token, but one from the best k tokens (or those tokens whose probability are at least p). We used beam search with $num\_beams = 5$ and top-k sampling with $k = 7$. Also there is an important hyperparameter called *repetition penalty*(Keskar et al. (2019)) that penalizes tokens that have been generated before. We used $repetition\_penalty = 8.0$. As for evaluation, there are some metrics for generation models (i.e, BLEU, Preplexity), but human evaluation is still the best method. Despite this, we calculated Cross-Entropy (CE) loss on the test dataset. The results can be observed in the Table 4.4.

The Table 4.5 below shows the two examples of generation. We generated these sentences by feeding models first two words (i.e. "Зачем мне" and "В чем"). As it can be seen, tuned ruGPT3Large model generated something very

Model	CE Loss
ruGPT3Small	4.16
ruGPT3Large	3.86
Tuned ruGPT3Small	3.72
Tuned ruGPT3Large	3.25

Table 4.4: CE loss on the test dataset

similar to a real jokes (at least we can see the humor in this). ruGPT3Small model generated something more or less meaningful, but there is no humor in its sentences. As for generation time, it varies greatly, but usually does not exceed 2 sec for Large model and 1.5 sec for Small model.

Generated Sentence	Model
Зачем мне это нужно?- спросила она.—Потому что я хочу, чтобы ты стала моей женой.—Я не могу	not tuned Large
Зачем мне это надо? Я не хочу, чтобы кто-то об этом знал.	tuned Small
Зачем мне жена, если у меня есть любовница?	tuned Large
В чем смысл жизни?Смысл жизни в том, что наша жизнь нужна Богу. Бог не всемогущ над собой, и ему что-то нужно.	not tuned Large
В чем смысл жизни? В том, чтобы не думать о завтрашнем дне.	tuned Small
В чем сила, брат? В ньютонах!	tuned Large

Table 4.5: Examples of generated "jokes" from the two first words

Although the two examples above shows that ruGPT3Large generates decent jokes, it is a quite rare situation. We generated around 100 jokes by taking the first two words of sentences from test dataset. Then we looked through all 100 generated sentences and only 7 out of them can be taken as jokes. Also quality of generation slightly depends on the first two words, specifically model generates more or less good jokes about politics, women, relationship. Perhaps, this is due to the train dataset and its content.

4.2.4 Discriminator for generated jokes

We generated approximately 6500 jokes using each model (6500 using Large, 6500 using Small). Generation was done using 6500 samples from test dataset (we took first two words, as before). Then we added 6500 real jokes from train dataset. So we got 13000 samples, 6500 in each class. Then we did absolutely the same what we did in "Classifitacion task": split data on train/validation/test dataset, then we trained CountVectorizer + SVM classifier and ruBERT classifier. We achieved 0.718 accuracy with SVM and 0.882 accuracy on the test dataset using ruBERT for ruGPT3Large generated samples. For ruGPT3Small generated samples: 0.781 accuracy for SVM and 0.884 for ruBERT. ruBERT was trained for 10 epochs using the same hyperparameters as before. As we can see ruBERT achieved a slightly better performance on dataset with jokes that were generated by the smaller model. And SVM achieved noticeably better results on the same data. Now we can use these ruBERT model to find good generated jokes.

We want to find those generated jokes that discriminator classifies to positive class (i.e. real jokes) with high probability. In theory, these jokes should be good jokes since our discriminator was trained to distinguish real jokes from generated jokes. Unfortunately, this approach does not work well: the most of the generated jokes from the test dataset that model classifies as real jokes (probability at least 0.5) are not not jokes at all. And on the contrary, some good generated jokes,that look like real, are classified as a generated jokes. Some examples are provided in the Table 4.6. But this idea with discriminator lead us to train GAN.

4.2.5 GAN for jokes

As it was mentioned before, GANs have not succeeded in text-generation tasks, yet (Caccia et al (2020)). Nevertheless we decided to try improve ruGPT3Small using RelGAN's discriminator and Gumbel-Max trick (see Related Works section). So, we sampled examples from ruGPT3SMall using Gumbel sampling with a maximum sequence length equal to 32 and then pass it through discriminator with

Generated joke	$p_{joke\ is\ real}$	**Our Score**
Объявление. Купим б/у авто в любом состоянии. Не битый, не крашеный, без пробега по дорогам России.	0.57	joke
Быстро сказанное слово не воробей, вылетит - не поймаешь.	0.98	not a joke
Путин всегда прав!	0.9	not a joke
Женщина отличается от мужчины тем, что она умеет готовить борщ и стирать носки.	0.61	joke

Table 4.6: Examples of "discriminator" output (probability) that generated joke is a real joke

two embedding matrices (see Related Works section or original paper for more details Nie et al. (2020)). We have tried to use two different losses: standard GAN loss (Goodfellow et al. (2014)) and Relativistic standard GAN (RSGAN) loss (Jolicoeur-Martineau (2018)). The equations for these two GAN's losses can be seen below (Equation 3, Equation 4).

$$\begin{gathered} L_{disc}^{standard} = -\mathbb{E}_{x \sim p_{data}} \ln D(x) - \mathbb{E}_{\hat{x} \sim p_{gen}} \ln(1 - D(\hat{x})) \\ L_{gen}^{standard} = \mathbb{E}_{\hat{x} \sim p_{gen}} \ln(1 - D(\hat{x})) \end{gathered} \tag{3}$$

$$\begin{aligned} L_{disc}^{rsgan} &= -\mathbb{E}_{(x,\hat{x}) \sim (p_{data}, p_g)} \ln sigmoid(D(x) - D(\hat{x}) \\ L_{gen}^{rsgan} &= -\mathbb{E}_{(x,\hat{x}) \sim (p_{data}, p_g)} \ln sigmoid(D(\hat{x}) - D(x)) \end{aligned} \tag{4}$$

At each epoch we trained generator for 50 steps and discriminator for 10 steps. We used Adam optimizer with learning rate equal to $5 \cdot 10^{-3}$ for generator and $3 \cdot 10^{-6}$ for discriminator. The GAN was trained for 150 epochs (6 hours) using standard GAN loss and RSGAN loss. Unfortunately, in both cases the discriminator's loss was decreasing and generator's loss was increasing (see Appendix). And Cross-Entropy loss on the test dataset increased significantly (up to 30 compared to 3.7 before) and generation result sometimes is just a sequence of brackets or commas. Also after training we freezed discriminator and tried to train generator using that freezed discriminator and generator couldn't "beat" (or even improve generator's loss) freezed discriminator after 20 generator epochs. So this may be not the best way to fine-tune GPT model or we should try different hyperparameters or train GAN longer.

4.2.6 Summary

We solved the problem of humor detection using ruGPT3Large and ruBERT model and showed that ruGPT3-based classier model outperform previous solutions on the FUN dataset Blinov et al. (2019). Also we constructed new jokes dataset and solved the same humor detection problem. On our dataset models achieved approximately the same performance (0.997 accuracy), but ruGPT3 model training proccess took 57% longer compared to ruBERT training time.

As for joke generation, we trained two different ruGPT3 models: small and large. The quality of generation is poor, but overall the Large model generates better examples than the smaller model. The ruBERT model can distinguish well between generated and real jokes, but this is a rather inefficient way to select good generated jokes.

After that we trained GAN model using fine-tuned ruGPT3small as generator and RelGAN's discriminator. Judging by the small number of experiments, we figured out that this not an efficient way to improve GPT model.

4.2.7 Further work

- It is a good idea to try using ruBERT model on the FUN dataset with joke decomposition (i.e. consider the joke as set-up + punchline) like in Annamoradnejad et al. (2020).

- To find a better dataset in which jokes have uniform structure. Then we can hope that the generator will learn this structure and will generate better jokes.

- To train ruGPT3Large model longer

- To do more experiments with GAN

References

1. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013. In ICLR Work-shop Papers

2. Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation. 2014. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543

3. Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks. 2014. nAdvances in neural information process-ing systems. pages 3104–3112

4. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. 2017. InAdvances in neural informationprocessing systems, pp. 5998–6008, 2017.

5. Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever. Improving Language Understandingby Generative Pre-Training. 2018. Technical report, OpenAI

6. Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, Ilya Sutskever. Better Language Models and Their Implications. 2019. Technical report, OpenAI

7. Rumelhart, David E; Hinton, Geoffrey E, and Williams, Ronald J. Learning internal representations by error propagation. 1986. In Nature, Volume 323, Issue 6088, pp. 533-536

8. Jacob Devlin, Ming-Wei Chang, Kenton Lee, andKristina Toutanova. BERT: Pre-training ofdeep bidirectional Transformers for language under-standing. 2019. nProceedings of the 2019 Conferenceof the North American Chapter of

the Associationfor Computational Linguistics: Human LanguageTechnologies, Volume 1 (Long and Short Papers),pages 4171–4186, Minneapolis, Minnesota. Associ-ation for Computational Linguistics.

9. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. 2015.In ICLR'2015

10. Issa Annamoradnejad. Neural ColBERT: Using BERT Sentence Embedding forHumor Detection. 2020.

11. Alex Wang, Amanpreet Singh, Julian Michael, Fe-lix Hill, Omer Levy, and Samuel Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In Proceedings of the 2018 EMNLP Workshop Blackbox NLP: Analyzing and Interpreting Neural Net-works for NLP, pages 353–355, Brussels, Belgium.Association for Computational Linguistics

12. Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. 2019.

13. Yuri Kuratov, Mikhail Arkhipov. Adaptation of Deep Bidirectional Multilingual Transformers for Russian Language. 2019.

14. Markus Freitag, Yaser Al-Onaizan. A Beam Search Strategies for Neural Machine Translation. 2017.

15. Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, Richard Socher. CTRL: A CONDITIONAL TRANSFORMER LANGUAGE MODEL FOR CONTROL LABLE GENERATION. 2019.

16. Anton Ermilov, Natasha Murashkina, Valeria Goryacheva, and Pavel Braslavski. Stierlitz Meets SVM:Humor Detection in Russian. 2018.

17. Vladislav Blinov, Valeriia Bolotova-Baranova, and Pavel Braslavski. Large Dataset and Language Model Fun-Tuning for Humor Recognition. 2019.

18. Jeremy Howard and Sebastian Ruder. 2018. Universallanguage model fine-tuning for text classification. InACL, pages 328–339.

19. Emil Joswin, Aditya Choudhary, Raghav Garg. DeepHumor - Generation, Classification and Analysis of Humor. 2019.

20. Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau,Laurent Charli . LANGUAGE GANS FALLING SHORT. 2020.

21. Weili Nie, Nina Narodytska, Ankit B. Pate. RELGAN: RELATIONAL GENERATIVE ADVERSARIAL NETWORKS FOR TEXT GENERATION. 2020.

22. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in Neural Infor-mation Processing Systems, 2014

23. Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. 2018

4.3 Appendix

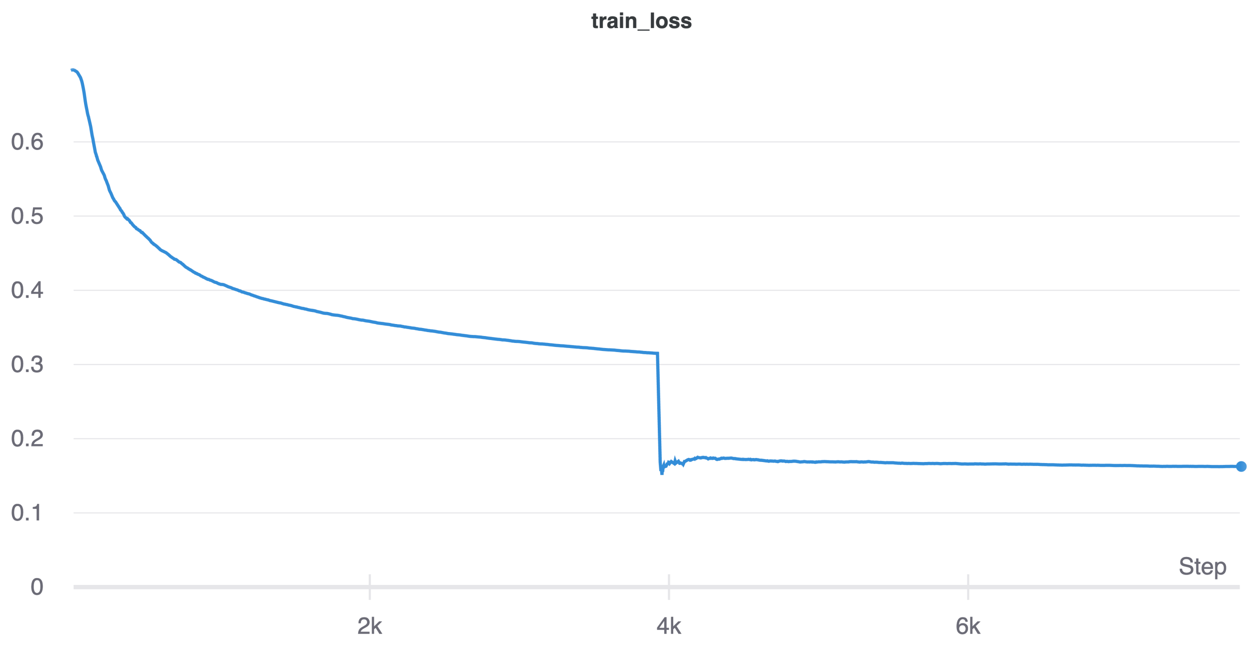

Figure 4.1: Cross-Entropy Loss for ruBERT training on FUN dataset.

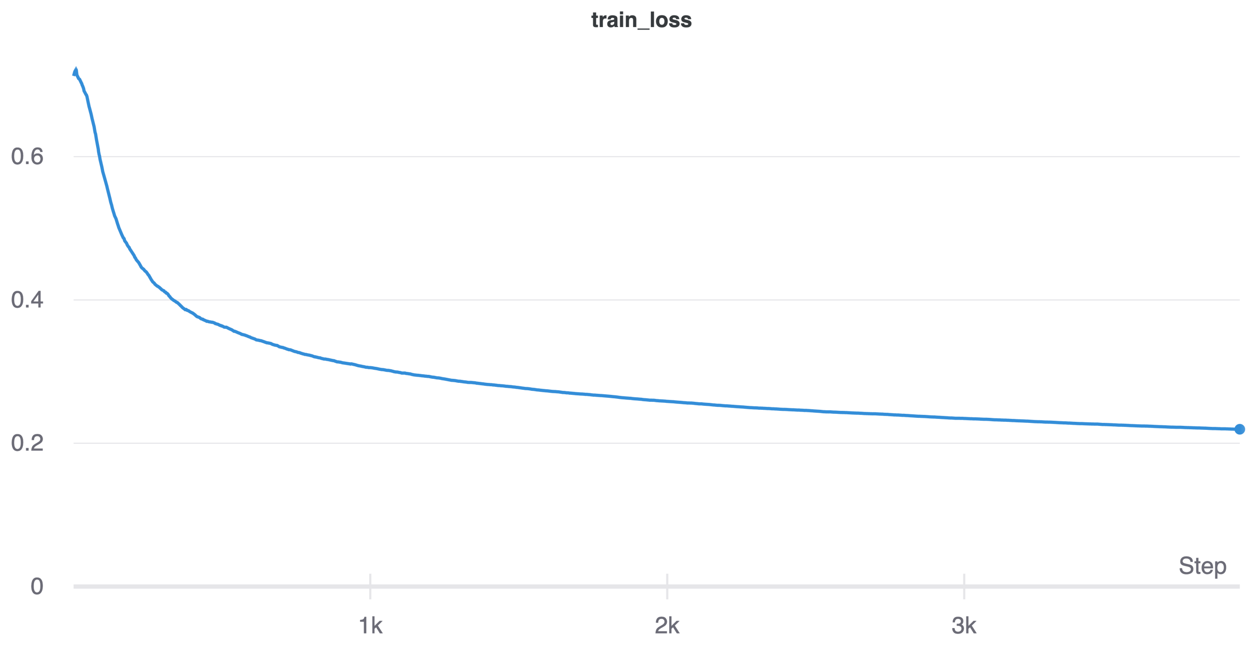

Figure 4.2: Cross-Entropy Loss for ruGPTLarge training on FUN dataset.

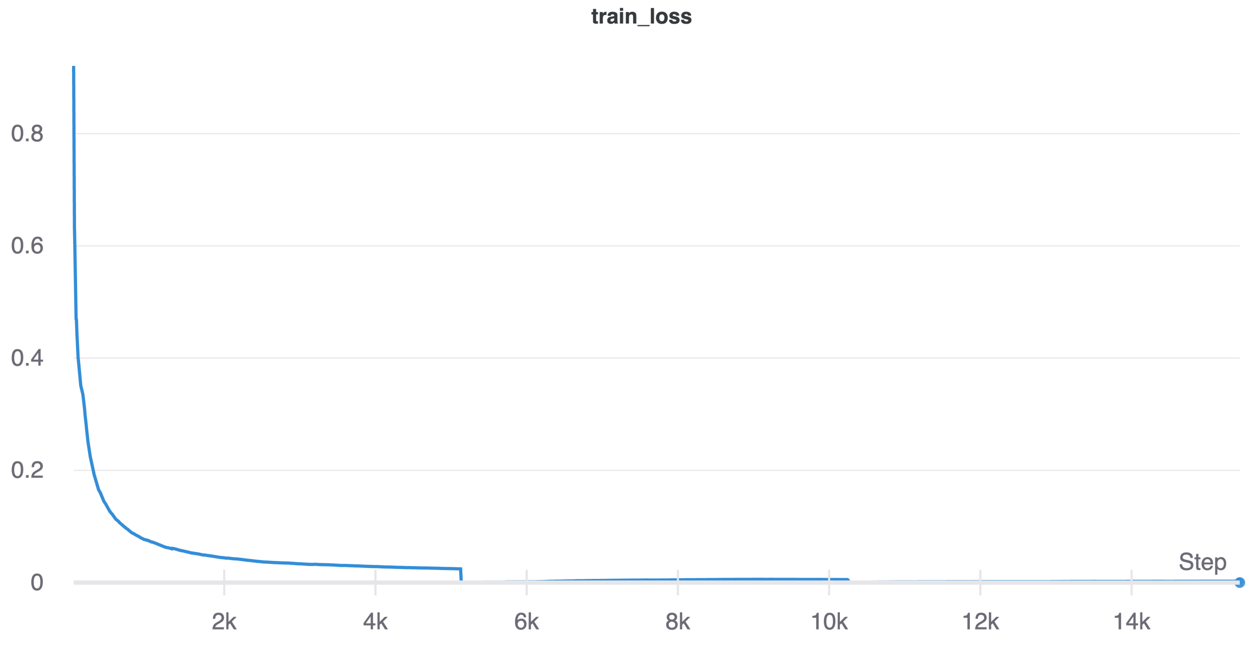

Figure 4.3: Cross-Entropy Loss for ruBERT training on our dataset.

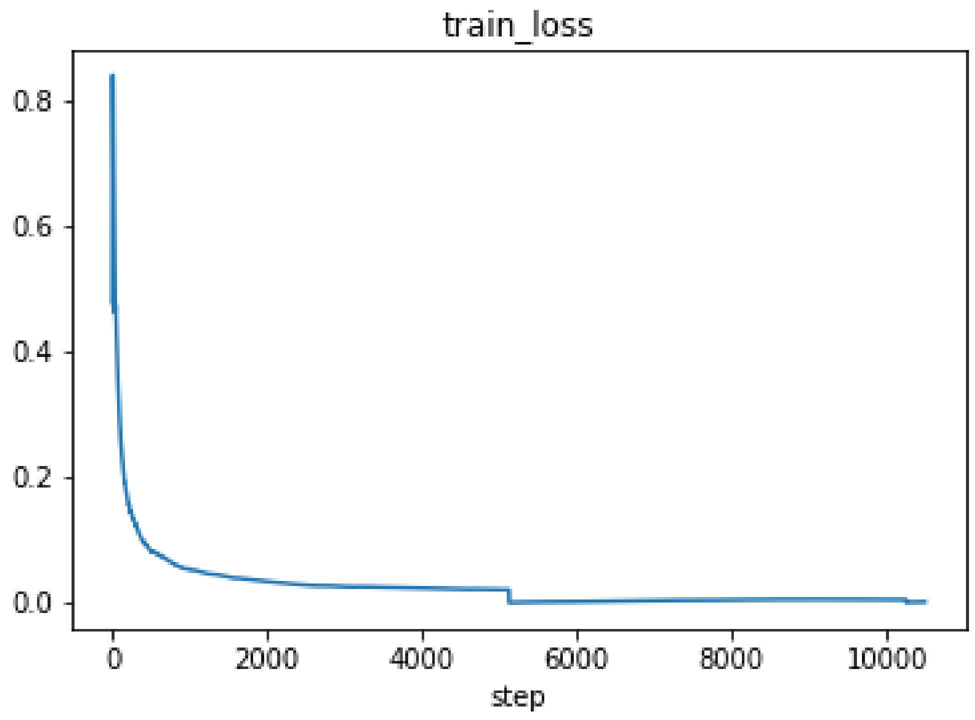

Figure 4.4: Cross-Entropy Loss for ruGPT3Large training on our dataset.

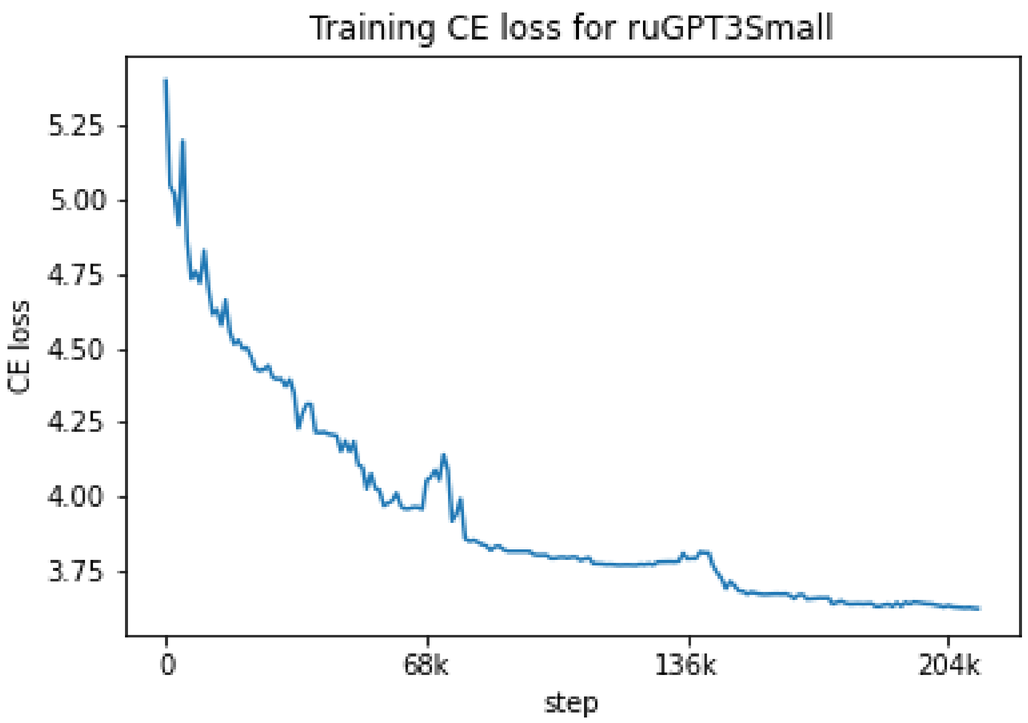

Figure 4.5: Cross-Entropy Loss for ruGPT3Small fine-tuning for the first three epochs.

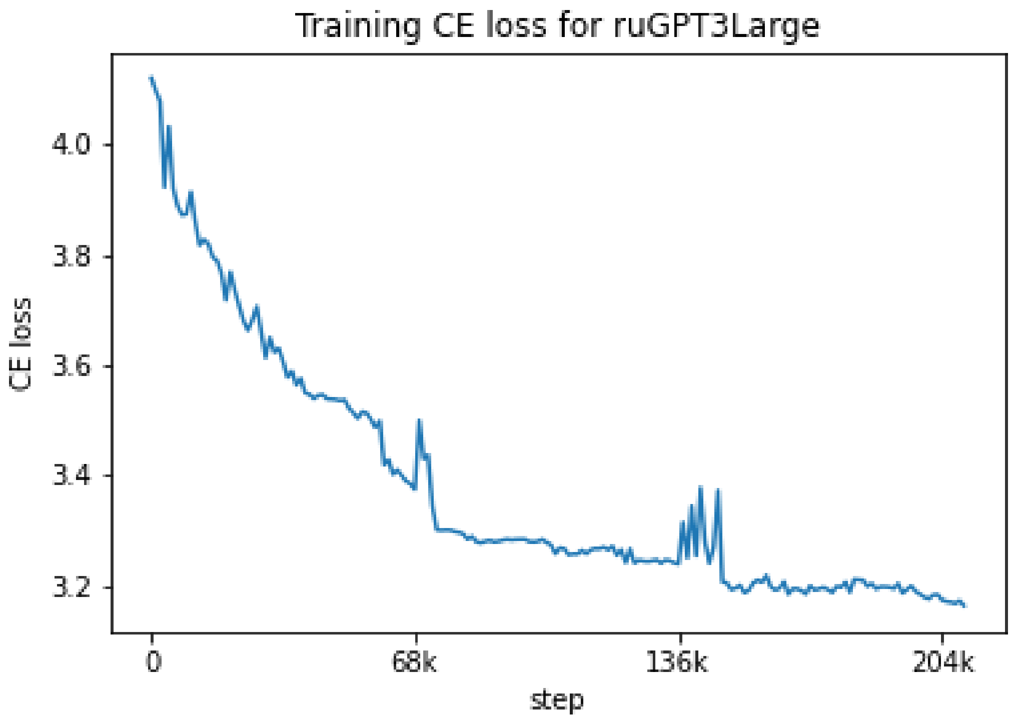

Figure 4.6: Cross-Entropy Loss for ruGPT3Large fine-tuning for the first three epochs.

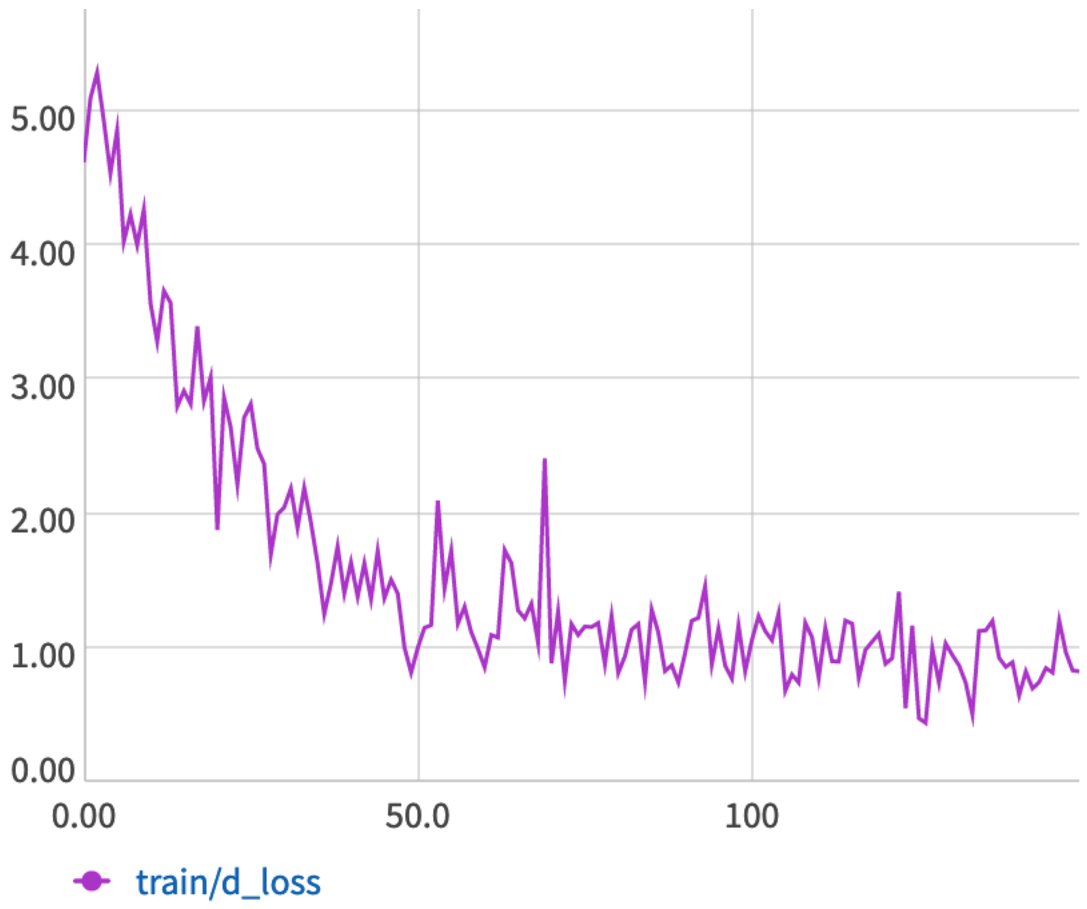

Figure 4.7: Discriminator loss. Standard GAN loss.

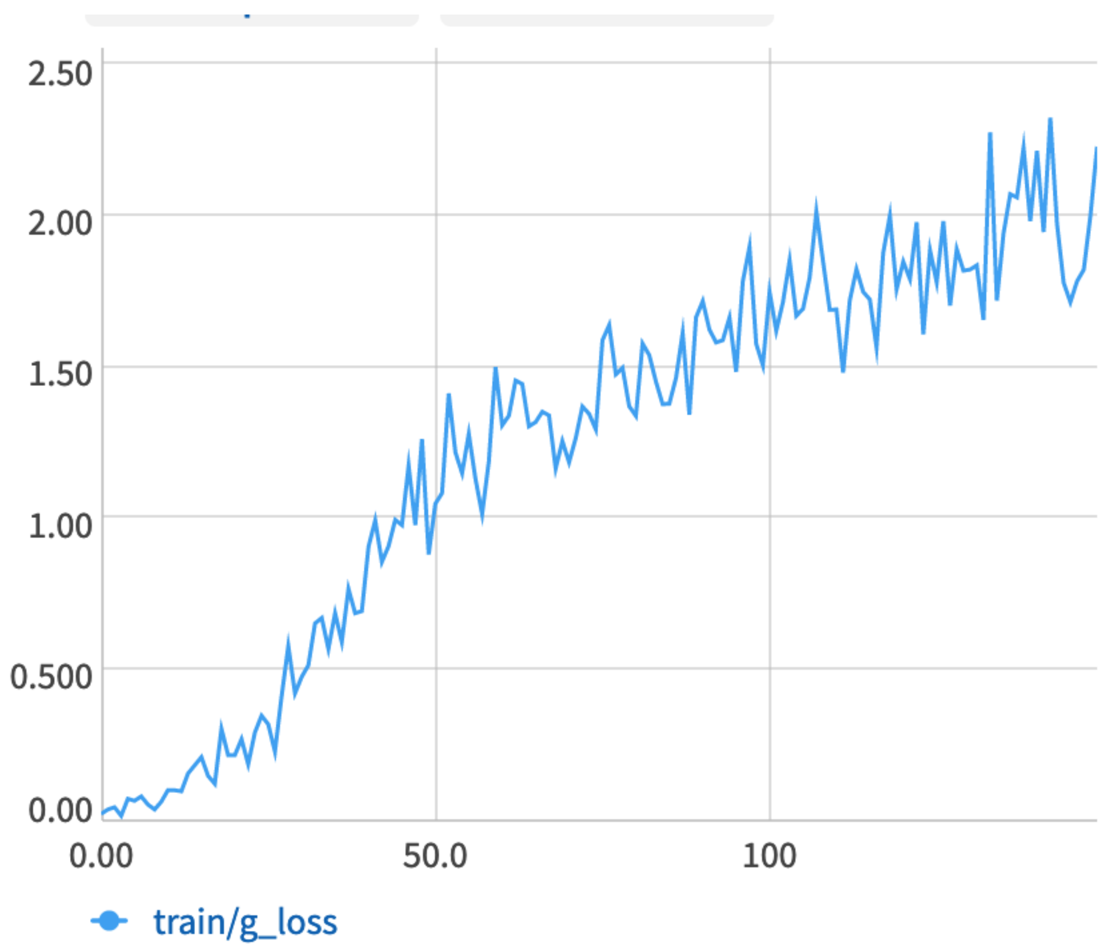

Figure 4.8: Generator loss. Standard GAN loss.

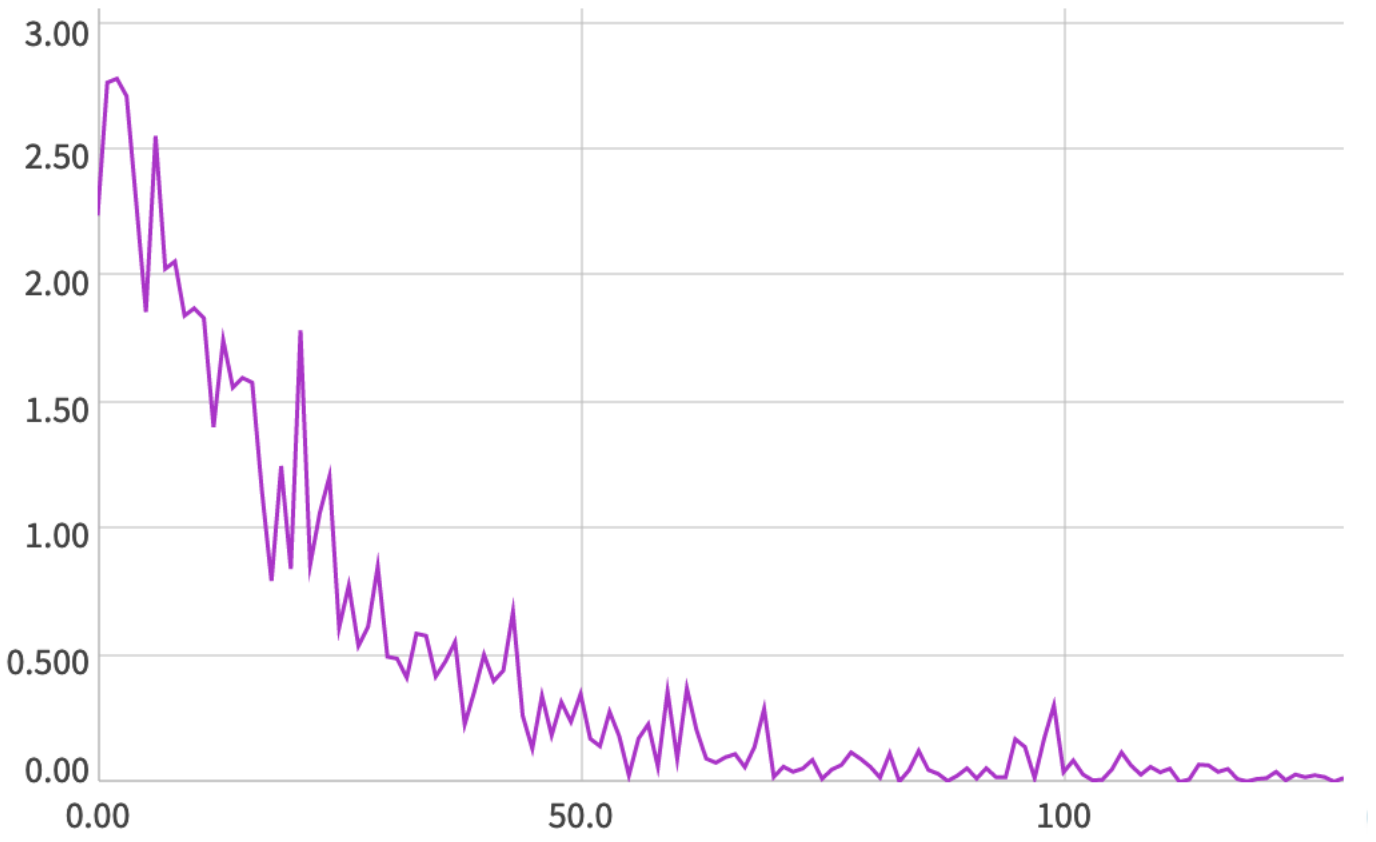

Figure 4.9: Discriminator loss. RSGAN loss.

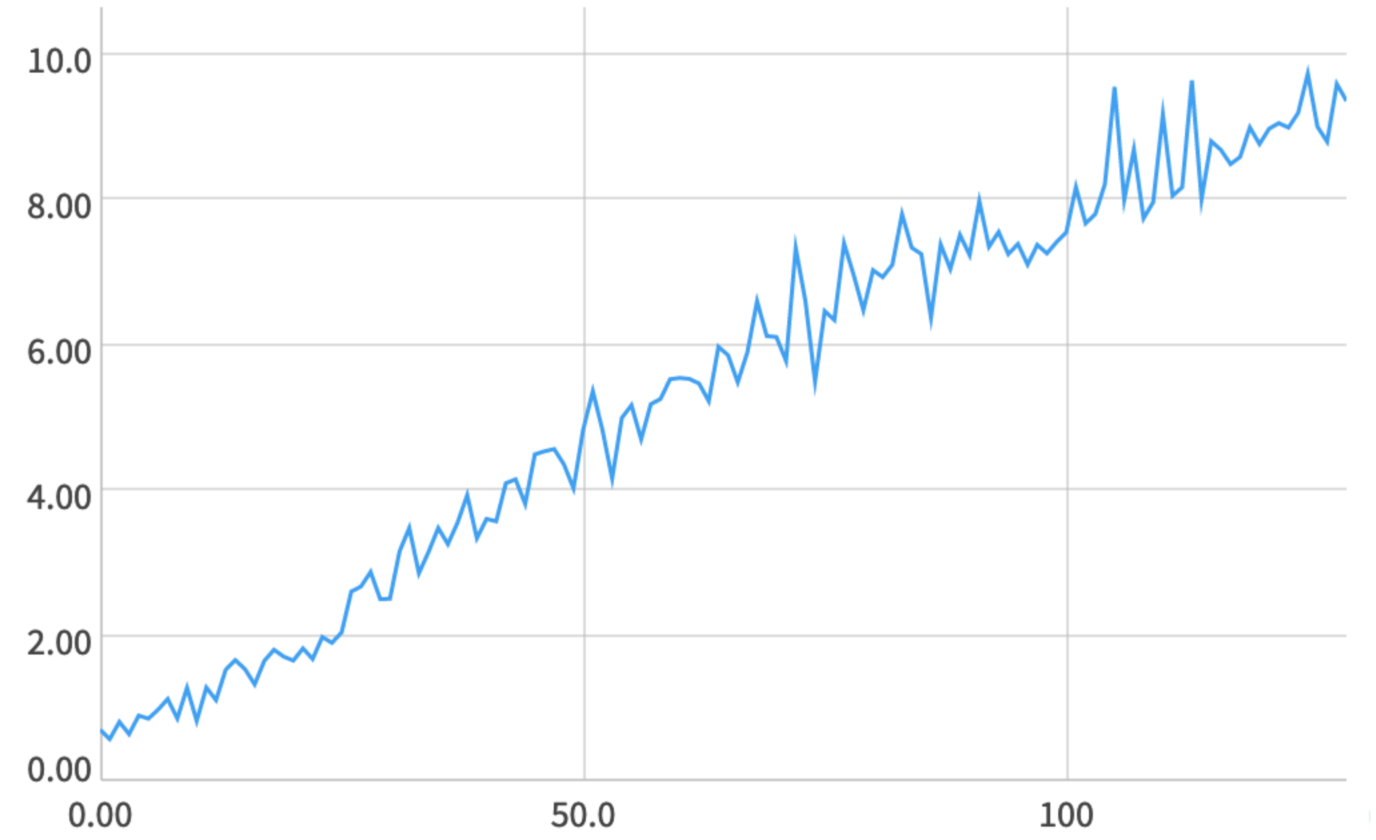

Figure 4.10: Generator loss. RSGAN loss.