Introduction to Machine Learning

Project Midterm Checkpoint

Adversarial Attacks on Language Models Using Text-GAN

By:
Zahra Bashir, Mahtab Farrokh, Fatemeh Tavakoli

Course Instructor:
Prof. Alona Fyshe

April 17, 2021

Contents

List of Figures

1 Introduction

It has been shown that Deep Neural Networks [12] are vulnerable to small perturbations on input data. These perturbations will fool the network and lead to unexpected outputs. Injecting these modified data into the model is called an adversarial attack [17], which has recently become a hot topic. For example, it is possible to mislead a CNN model [11] to predict a dog instead of a cat by adding noise to a cat picture. Notably, these perturbed input instances can not deceive humans. As another example, in the language model context, a positive sentence can wrongly be predicted as a negative one. This project focuses on generating adversarial examples to fool language models on sentiment analysis tasks. For this purpose, we use Generative Adversarial Networks(GANs) [5].

This problem is significant from two aspects. First, many decision-makings are dependent on deep learning models' results, which can directly or indirectly affect humans' lives. Finding these adversarial examples suggests that neural networks are not ready to be used in critical-security applications. So, adversarial attacks help us detect the blind spot of our trained model to be able to defend it. Second, adversarial attacks indicate a flaw in the trained model. It can raise questions on the theory behind the deep learning models. In most cases, we need the model to think the way humans do, and if it does not, it can cause critical issues because it has not learned a human-like intelligence. Third, many adversarial attacks are done manually or using some algorithms. Therefore, instead of designing or using algorithms, we let a deep neural network generate the modified example. So, we aim to automate generating adversarial examples created explicitly for the victim model by using a GAN [19].

2 Related Works

There are two categories of adversarial attacks on Neural Networks. The first one is the "White-Box" attack in which we have access to the internal parameters of the model, and the other one is the "Black-Box" attack in which we only have access to the input and output of the victim model. The fast gradient sign method (FGSM) [4] is one of the white box attacks that has been proposed in "Explaining and harnessing adversarial examples," which uses backpropagation gradient to make new adversarial samples. In this regard, "TextFool" [13] applies the idea behind the FGSM to the text domain. It will consider the magnitude of the cost gradient instead of using the sign of the cost gradient as used in FGSM. Also, they introduced three types of attacks (specific to text data): insertion, modification, and removal.

In the concept of GAN-based attacks, the goal is to create more natural adversarial examples. "Generating adversarial examples with adversarial networks" [18] suggested a method to use a GAN model (AdvGAN) to add noise to an input image to fool the victim network. This paper's main idea is to combine the victim's loss and the discriminator's loss to be used in the GAN training procedure.

"Generating Natural Adversarial Examples" [20] proposed a method to generate adversarial examples for image classification, textual entailment, and machine translation. Their proposed model has two parts: A GAN and an inverter. First, they train the generator in a WGAN structure to create x^* samples similar to the real data. Then, they use an inverter to map the real data samples(x) to a latent space(z) and train it to minimize the reconstruction error of x. They use iterative stochastic search to add perturbation to the latent low-dimensional (z). As the last step,

they feed the perturbed z to the generator to get adversarial examples.

Considering the Adv-GAN method, we decided to use GANs to generate adversarial examples. However, since the mentioned paper only attacks computer vision models, we incorporate this idea into the text domain. Following their method, we add the victim's loss to the discriminator's loss. Since we are dealing with text data from a discrete space, we need to find an appropriate method for our data representation. "Adversarial Text Generation Without Reinforcement Learning" [3] helped us reach the idea of using an autoencoder to encode the text into a latent space. This latent vector is fed into our GAN as an input. Then, the GAN network generates adversarial data in the latent space. Afterward, using the decoder, we decode the output representation into the real text space (sentence-based). Instead of using an autoencoder to solve the discrete space problem, we can use pre-trained word embedding to map the input text into an embedding vector space(encoder part) and vice versa for the decoder part.

The most recent work which uses GAN to create adversarial examples in the text-domain is "Generating Natural Adversarial Examples." The paper uses the iterative stochastic search algorithm, which explicitly mentions that this algorithm is computationally expensive. The AdvGAN approach, which is mainly applied to the images, has not been tried on textual data. So, we came up to use the AdvGAN idea in the text domain. We expect our new method to be computationally efficient compared to the previous works. Furthermore, since the text data has a discrete space, its gradient is not differentiable, and backpropagation will not work. Consequently, limited research has been done on attacking text data so far. Specifically, limited work has been done on applying GAN for automating the generation of adversarial data.

3 Data

We use the IMDB movie reviews dataset [7] for our project. This dataset consists of 50k user reviews labeled with positive and negative sentiments. In this dataset, we have 25k positive and 25k negative samples. There are over 89k unique words in this dataset. Here is a positive sample review from the dataset:

"One of the very best Three Stooges shorts ever. A spooky house full of evil guys and "The Goon" challenge the Alert Detective Agency's best men. Shemp is in top form in the famous in-the-dark scene. Emil Sitka provides excellent support in his Mr. Goodrich role, as the target of a murder plot. Before it is over, Shemp's "trusty little shovel" is employed to great effect. This 16 minute gem moves about as fast as any Stooge's short and packs twice the wallop. Highly recommended."

The IMDB dataset consists of a sequence of movie review words, in which a number represents each word. In the pre-processing phase, we convert the numbers to a one-hot discrete vector representation. The dataset has 25k train-labeled, 50k unlabeled, and 25k test-labeled instances. This dataset has 2 class labels(binary); when it is equal to one, it means the review is positive, and zero shows a negative review. We only used the labeled examples because we focused on fooling classifier models(supervised task). The IMDB sentiment dataset has over 89K unique words, but we only used the top 2000 repeated words. Each word that is not in the top 2000 dictionary will be replaced by unknown, and we show it by "?". Moreover, in producing the adversarial examples phase, we limited the number of unknown words that a sentence can have and discarded any review with more than five unknown words. In this way, we make sure that setting 2000 dictionary words will not have any adverse effects on the victim model prediction. So, working on the top 2000 repeated words is sufficient.

This dataset does not have any fixed length for reviews and contains so many unique words. So, we need to pre-process this dataset. First, we limited the number of words dictionary to 2000. Then we limited the length of reviews by 200. The reason that we limited our number of words is to reduce the dimension and sparsity. Finally, We made a sequence of one-hot vectors of the words in the sentence. So, the final data we worked on has instances in the shape of 200*2000 representing a sentence.

Notabley, for training GAN and making the attack, we limited our input data to positive reviews only. The reason is that we wanted to focus on one label and then do our attack using that to get negative reviews as a result. Also, we limited the data to those in which the victim model predicts them correctly(positive) to make sure if after adding noise, the result changes, it is our work's effect and not the error of the victim model.

In this project, we aim to fool language models in a way that humans can understand the correct meaning of the sentence, but the victim model fails. Therefore, interpretability is essential. At the end of our work, we need to make sure that our generated adversary examples are interpretable by humans because we want to generate meaningful reviews which fool the victim sentiment classifier. For this aim, we used LIME(local interpretable model-agnostic explanations)[14] to explain the decisions made by a Naive Bayes classifier and compared the results of the original and the perturbed data which are fed into the model (as the test data).

4 Analysis / Methodology

We applied adversarial attacks on the victim language model using a GAN to generate adversarial examples. Figure 1 illustrates the flow of data through the algorithm to performance and demonstrates the overall architecture of our Text-GAN model, which mainly consists of encoder/decoder, GAN, and victim model. We will discuss these parts in detail.

4.1 Encoder/Decoder

The encoder takes a movie review sentence as input and encodes it to a representative vector. The decoder will take a vector (from the decoder output space) as input and reconstruct the original sentence. In the following subsections, two encoder/decoder approaches will be explained.

4.1.1 Autoencoder

An autoencoder is a type of neural network used to learn efficient data codings in an unsupervised approach. In our work, we care about the meaning of these data codings because we want to add noise to the latent space representation of original reviews and derive new meaningful reviews which can fool our victim model. To this end, we used Variational AutoEncoder(VAE) [10], that their latent spaces are, by design, continuous, allowing easy random sampling and interpolation. In a baseline autoencoder, the encoder's output is a latent space vector of size n. In contrast, VAE's output is two vectors of size n: a vector of means, μ, and another vector of standard deviations, σ. The latent space representation in VAE drives using a sampling function over the random variable i with respect to the μ and σ variables. The data is passed through an embedding layer; then, it will be fed into the encoder. The encoder consists of a bidirectional LSTM [16] and several dense layers. The decoder consists of two LSTM [6] layers and a dense layer.

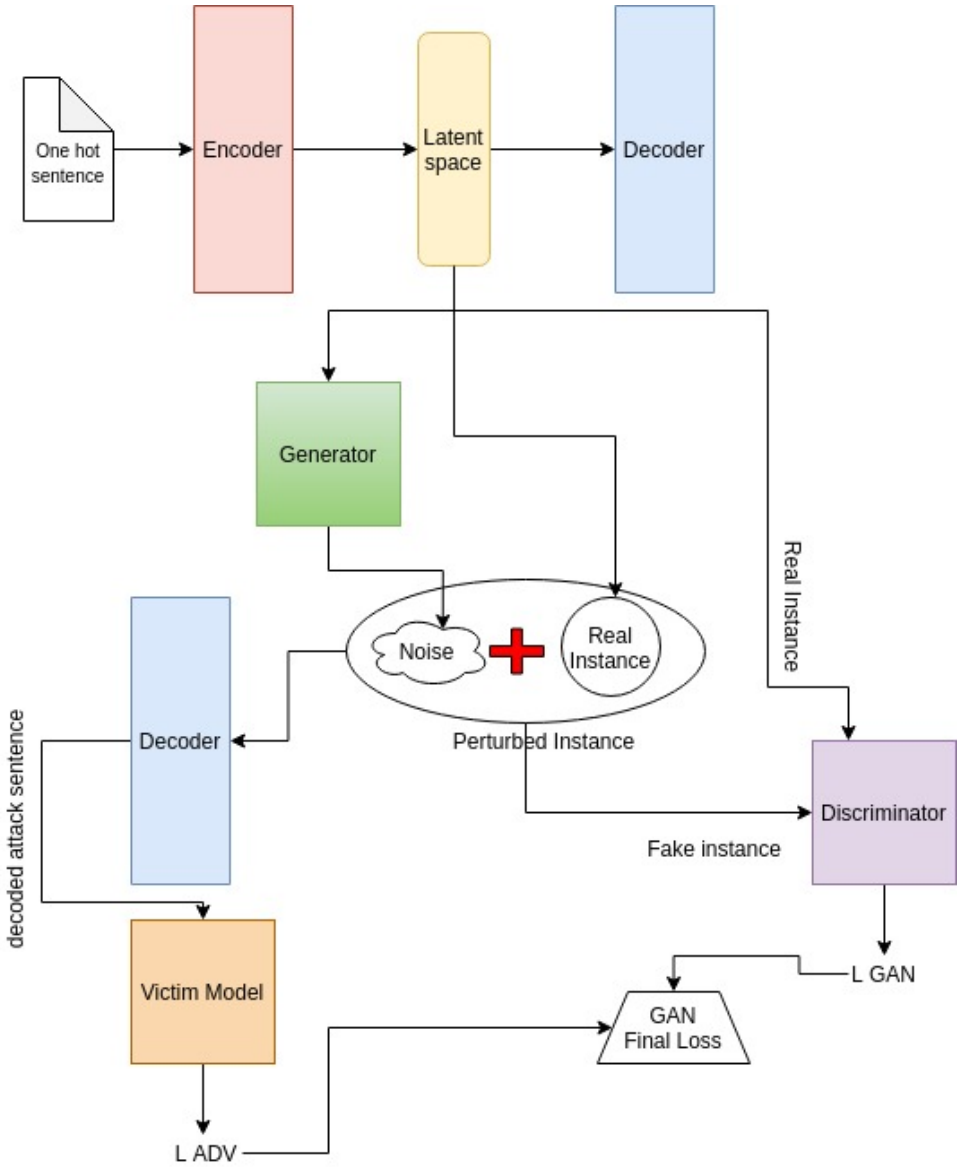

Figure 1: Block diagram of all components. We repeated the decoder component in two places of our diagram for simplicity.

We want to know that in this model, how much our decoded data is close to the preprocessed input data. So, by measuring the difference between these two, we can evaluate the performance of the autoencoder. Another evaluation approach is finding the correlation between the latent vector's values and the input vector; the lowest the correlation, the better. The loss function definition is essential in evaluating the autoencoder. We defined it as a combination of the binary cross-entropy and KL divergence that is more suitable for the complex distribution of our data.

The hyperparameters in VAE are the number of layers and the number of units in LSTMs. We have an embedding layer before the encoder part with the number of outputs set to 64 and a latent space size set to 200. The number of units represents LSTM memory, so we have to set it to a number that is not that low that cannot include a full-sentence sequence, and not that high to be useless for our case. The number of layers represents how deep our model is. It is proved that for most of the tasks, 2 or 3 layers of LSTM are enough to detect more complex features. So, we

used two LSTM layers for both parts. Another hyperparameter is the batch size which is set to 10, and the epoch size, which is set to 30. Notably, most of the hyperparameters are chosen based on previous experience and trial and error.

It is to be noted that the autoencoder's accuracy was not sufficient for this task, and it will be discussed in detail in the result result section 5.3. So, we considered the following approach using a pre-trained word embedding module.

4.1.2 Pre-trained Word Embedding

We used GloVe(Global Vectors for Word Representation) [9] word embedding, which is an unsupervised learning algorithm for obtaining vector representations for words where similar words cluster together. The advantage of GloVe is that it does not rely just on local statistics (local context information of words) but incorporates global statistics (word co-occurrence) to obtain word vectors. [8] For instance, when we print the nearest neighbors of the word "frog" in this setting, it will indicate Litoria, Leptodactylidae, etc., which are so close to the frog in meaning (they are different species of frog).

Given each review(one hot sentence representation), we compute each word's GloVe output, which maps each word to a vector space of size 50, and this is the encoder part. This vector will then go to the decoder part, and the decoder will find the nearest neighbor to this vector and return it as a word. It is to be noted that GloVe provided functions for all of this process.

Since GloVe is a pre-trained model, we did not need to set any hyperparameter except for the embedding size that we set to 50. (it is a common size in this representation) Also, there is no need to check the performance and accuracy of this pre-trained model.

4.2 Victim model

The victim can be any textual model that would be used for sentiment analysis ranging from simple models such as Naive Bayes, Decision Tree, and SVM [2], to more complicated ones such as Deep Neural Network [12], Convolutional Neural Network(CNN) [11], and Recurrent Neural Network (RNN). Train and test accuracy and loss are the metrics to measure performance, and the reason for using them is that they are simple and easy to work with. Notably, we only need the loss from this victim for our attack, and we do not train the victim model. We work with the victim model in a semi-white-box setting. There is no need to have access to the victim model after the GAN model is trained in a semi-white-box attack. The victim's loss will be derived using binary cross-entropy on the prediction of the victim.

Here we have the formulation for the victim's loss:

$$L_{ADV} = E_x loss_f(x + G(x), target) \tag{1}$$

Here, the victim's loss function can be binary cross-entropy for our sentiment analysis task. If we consider our work as a white-box attack, the loss function is defined with respect to the victim's algorithm. E_x is the expected value over all real data examples. Also, the target is the class function that we want to make the victim predict that. In other words, it is the opposite of the true class label, and it will lead the victim to be fooled. We fed the victim's loss to the generator to create perturbation that, when added to the input vector, fools the victim model and seems real.

We applied k-fold cross-validation on the training set to prevent overfitting. Setting k = 10 is a typical value for k. We trained on the nine folds of our data and predicted the remaining fold. We repeated this process on all folds and reported the folds' average accuracy. By doing so, we will have a more generalized model.

The hyperparameters for the Naive Bayes model are alpha, binarize, fit_prior, and class_prior. The two important hyperparameters are alpha and binarize. Alpha is used for smoothing(in our work, we have no smoothing, zero by default). Binarize is a threshold for sample features to be binarized. (it is also set to zero in our code)

4.3 GAN

The GAN consists of two parts: a generator and a discriminator.

4.3.1 Generator

The input to the generator is the output of the encoder (latent space vector). The generator generates some noise according to the input data, and then we add this noise to our input data to generate the adversarial example. This is what makes our GAN different from regular GANs. Also, we define the performance as how it can generate data close to the real data that can fool the discriminator. For now, we work with the generator loss for its performance evaluation, but we can use the idea of Inception Score[15] for our future works. We also checked the created sentences ourselves to make sure they seem real and natural.

The generator has six layers. 1) An LSTM with 128 units, 2) a Danse layer with 50 units, 3) LeakyRelu layer with alpha = 0.1, 4) Dropout layer with 0.2 probability, and 5) a dense layer with latent vector size as the number of units and a *tanh* activation.

4.3.2 Discriminator

The discriminator aims to predict whether an input review is fake or real. The goal of this part is to help the generator to create natural adversarial examples. In the discriminator part, the more it can detect the data being fake or real, the more accurate its model is. Our model's performance can be measured using metrics like accuracy, loss, precision, recall, and F1-score.

The discriminator has six layers as well. 1) An LSTM with 128 units, 2) a dense layer with 100 units, 3) LeakyRelu layer with alpha = 0.2, 4) dense layer with 100 units, 5) LeakyRelu with alpha = 0.2, 6) a 1 unit dense layer with *sigmoid* activation as the final discriminator prediction.

The baseline GAN loss is:

$$L_{GAN} = E_x log D(x) + E_x log(1 - D(x + G(x))) \tag{2}$$

$$L = \alpha \times L_{ADV} + \beta \times L_{GAN} \tag{3}$$

Here α and β are the hyperparameters that need tuning. We used an ADAM optimizer for GAN training with a learning rate = 0.0002 and beta to 0.5. The beta hyperparameter controls the decay

rate. Furthermore, we used *LeakyRelu* as the activation in middle layers to avoid the dying Relu problem for negative values causing overfitting. The alpha value defined the output for negative values instead of zero. So, instead of zero, we have x*alpha for negative values.

4.4 Statistical Significance

Null hypothesis: Applying test data on the Naive Bayes model and getting a wrong prediction is by chance and has nothing to do with the noise added by the GAN. We executed the code ten times with the original non-perturbed sentence as test data and achieved some accuracies. Then, we can reject or accept the null hypothesis by running the code using the perturbed test data.

4.5 Interpretability

Interpretability means that models' decision-making is explainable. We can discuss that from two aspects; from a GAN's perspective, like many other machine learning models, GANs typically work as a black-box module that cannot provide a complete explanation of their generation process. However, from a victim's perspective, we can interpret the Naive Bayes model using LIME(local interpretable model-agnostic explanations). We fed the regular sentence and the perturbed sentence to the classifier. Then, We plotted the results by LIME so that we can figure out which changes caused the classifier to mispredict the noisy sentences' class.

5 Results

5.1 DATA

We chose five of the words that are among the 100 topmost frequently used words in our samples which have positive or negative sentiments, such as the words "bad," "worst," "waste," "great," and "best." We plotted the occurrence of each of the words in positive and negative samples separately.

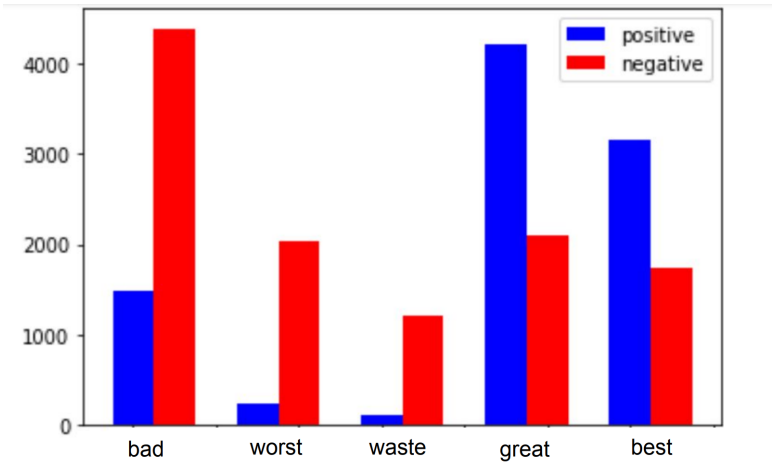

Figure 2: Frequnecy of some of the widely used words in the IMDB dataset in negative and positive reviews.

figure 3 indicates the word cloud of IMDB movie reviews on the whole dataset.

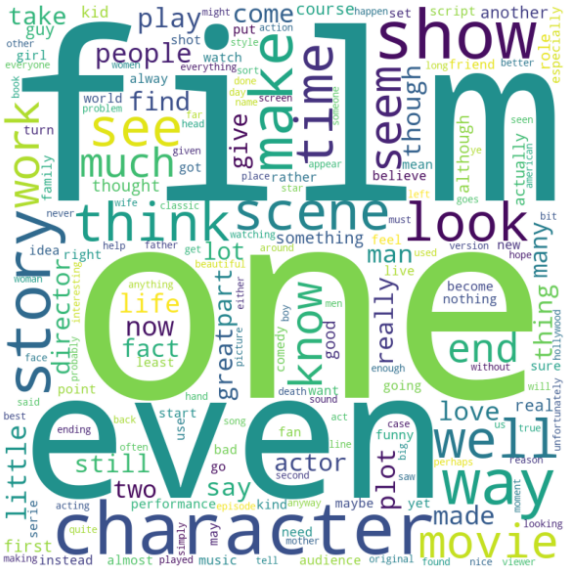

Figure 3: Word cloud of imdb dataset.

We chose the maximum length of the sentences using the results from figure 4. As it is shown in the plot, most of the sentences have sentence lengths less than 200. Based on this result, we chose the maximum sentence length of 200 to cover most of the sentences entirely and be computationally efficient.

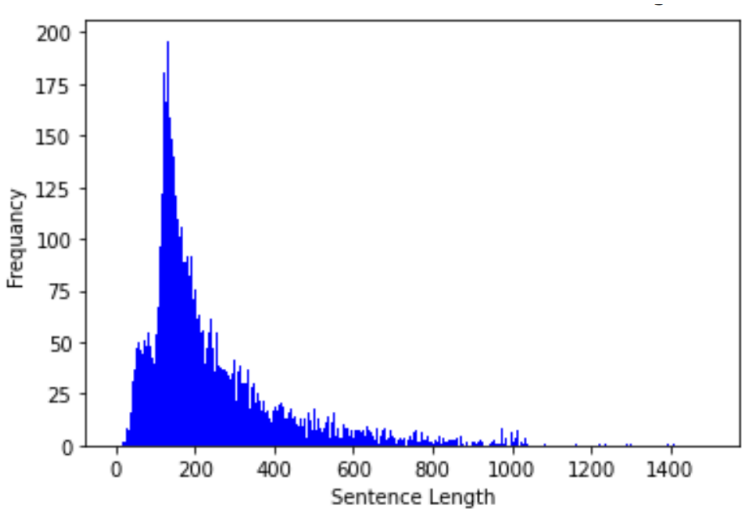

Figure 4: Frequency of sentences based on their length

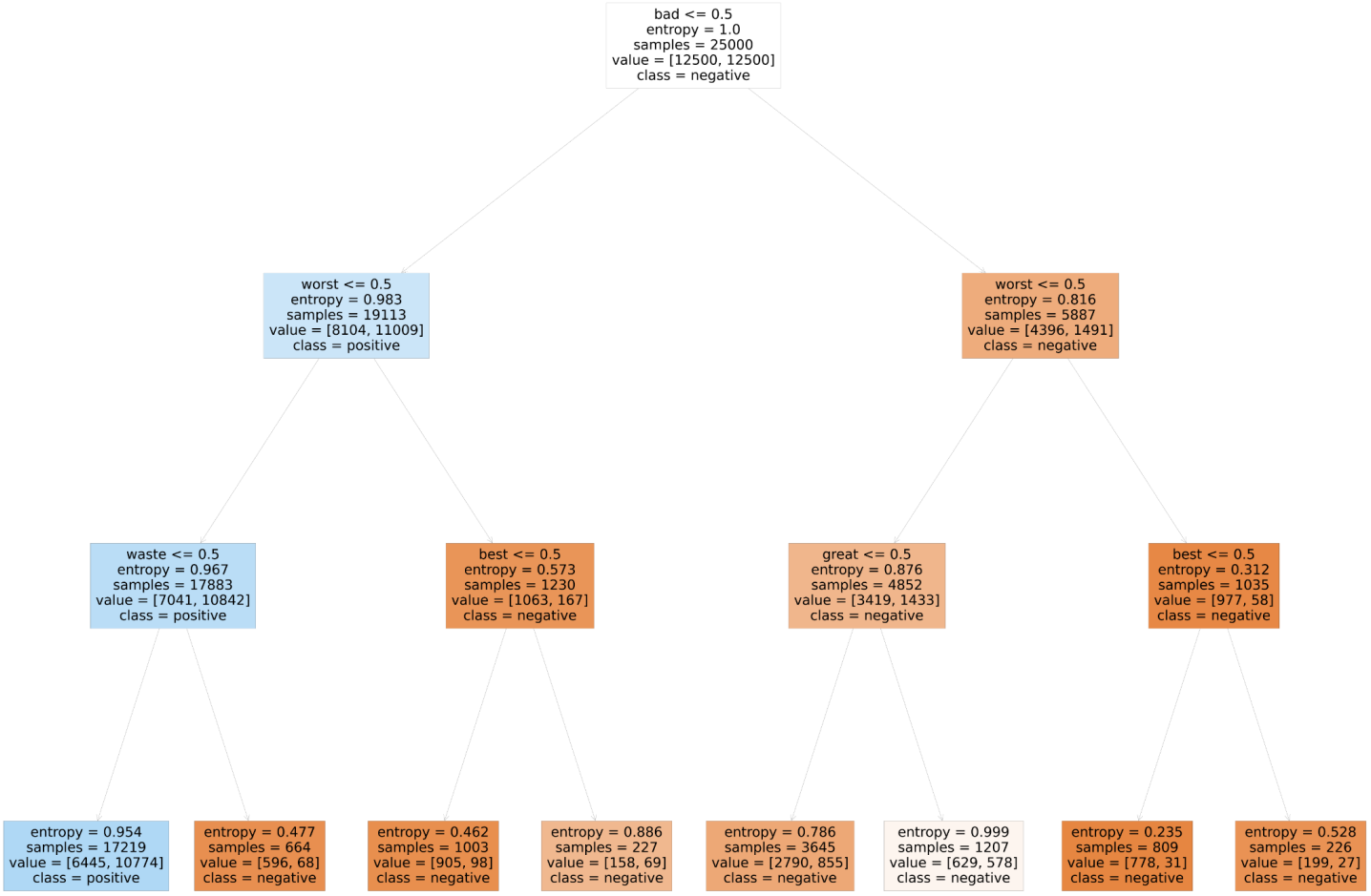

Figure 5: Visualization of data using decision tree

We limited the max_depth of our tree to 3 for better visualization. You can understand this from high entropy values as well. The max_depth used in our implementation is not limited, and the only max_leaf is set to 140. Therefore, we reached better results in our implementation with low entropy. figure 5 shows the result of the sentiment classification of an example in our decision tree approach.

5.2 Victim model

For the victim model, we trained and tested the Naive Bayes, SVM, and Decision Tree, on the IMDB sentiment dataset, and the accuracy result is shown in Table 1.

Our focus was on the Naive Bayes model for this project. In other words, we only tested the adversarial examples generated by GAN on a Naive Bayes model. We reached an average accuracy of 81.84% on the training set and 82.60% on the test set using 10-fold validation. We could get the average accuracy of 90.35% on the training set and 89.95% on the test set on this model with a standard-setting, but since we limited the length to be 200, our result had a lower accuracy.

Table 1: Accuracy of victim model before attack

Model	Train Accuracy	Test Accuracy
Naive Bayes	0.818	0.815
Decision Tree	0.778	0.753
SVM	0.985	0.898

5.3 Autoencoder

At first, we trained the autoencoder on 3000 data samples and tested 1000 data samples. Each review sentence was truncated to the length of 300, and if the review's length was less than 300, we added zero paddings to the start of the review. The AE's results were not satisfying because of using a limited part of the training data, and the average accuracy was 51.40%. This result is shown in figure 6.

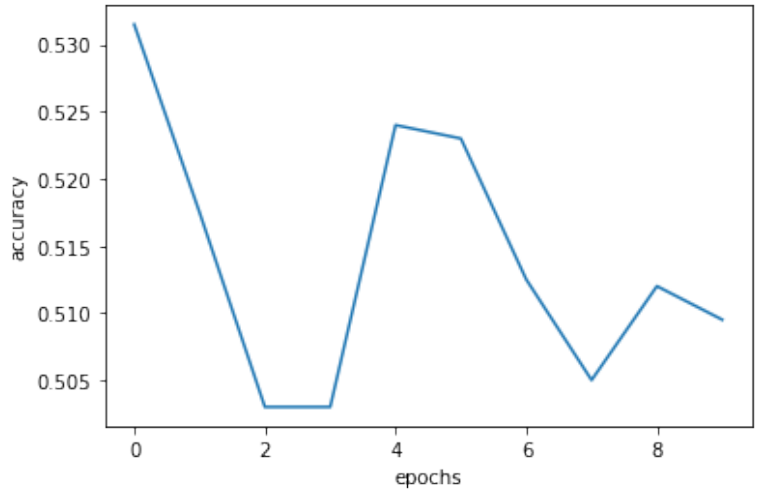

Figure 6: Autoencoder test accuracy on 10 epochs trained on only 12% of the whole dataset.

After some investigation, we found out that our model only learned to put zero paddings to the beginning of the sentence and added: "the" for the rest of the sentence. It showed that our model was not capable of training, and we had an under-fitting problem. Therefore, the accuracy of 51.40% was not valid.

Since increasing the epoch and training for a long time did not solve the problem, we decided to increase the training dataset to 20k data samples and reduced the reviews' length to 200 words. We trained the model longer on GPUs for 24 hours. Due to decreasing the length of reviews and zero paddings, we saw that our AE's accuracy dropped to 28%. Considering this poor result, we decided to use a more complex autoencoder. We used Variational Autoencoder(VAE) and defined a custom loss that combines binary cross-entropy and KL-divergence. The custom loss increased the accuracy of VAE by 16 percent, and the final accuracy became 44.35%, which is shown in figure 7. However, this accuracy is not acceptable for our work, and it shows that we need to add more layers to the VAE and train the model for a longer time. Due to the lack of time and resources, we decided to use a pre-trained word embedding named GloVe.

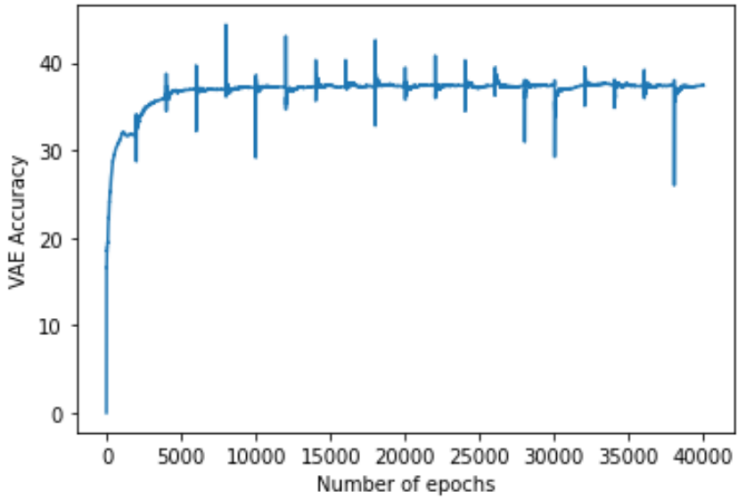

Figure 7: Variational Autoencoder accuracy, trained on 20k data samples for 24 hours.

5.4 GAN

The GAN code is executed on 15 epochs and with a batch size of 10. We tested different hyper-parameters and executed the code several times to reach the appropriate results. For the GAN's customize loss that we explained in section 4.3 we found out $\alpha = 5$ and $\beta = 2$ generates the best result.

Table 2: GAN results

Model	Loss	Accuracy
Generator	214.9	-
Discriminator	1.51	0.732

Figure 8 shows the discriminator's loss during 15 epochs.

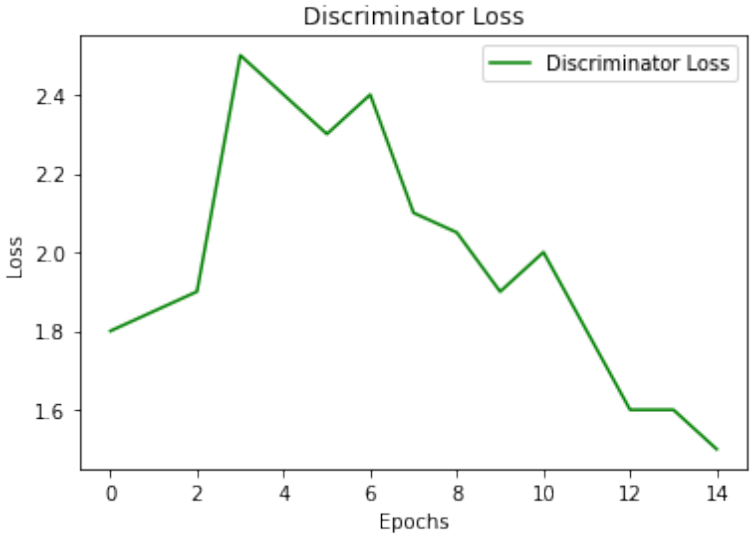

Figure 8: Discriminator's loss executed on 15 epochs

As we discussed before, we used LIME to interpret the generated adversarial examples and check how well our model learned to fool the victim model. In figure 9, an original positive review example is shown, and this is the review content:

"i see ? of student films this is ? james ? is a good director he moves the camera tells the story and uses music in a way that is far ? for his years no wonder he got a feature from this film br br"

Figure 10 shows the perturbed review example, and the content is:

"i see ? of teacher movies the is ? c ? is a good director gets makes normal camera tells the story and uses music in a way can is situations fat for this years no wonder he getting a thing from makes film br br"

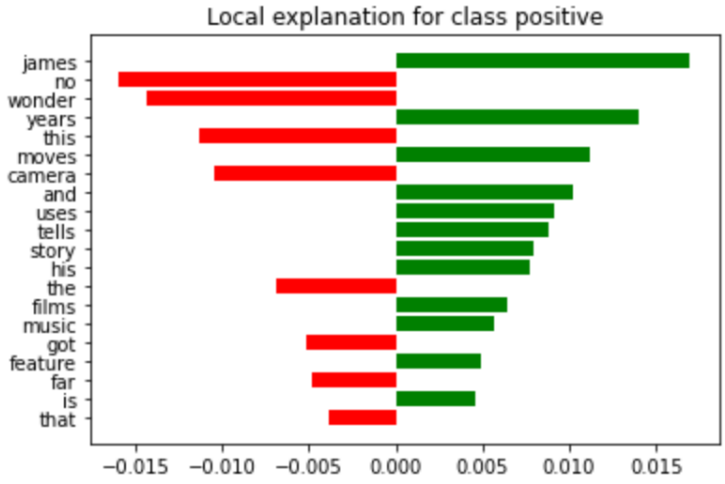

Figure 9: Lime interpretation of a positive original review

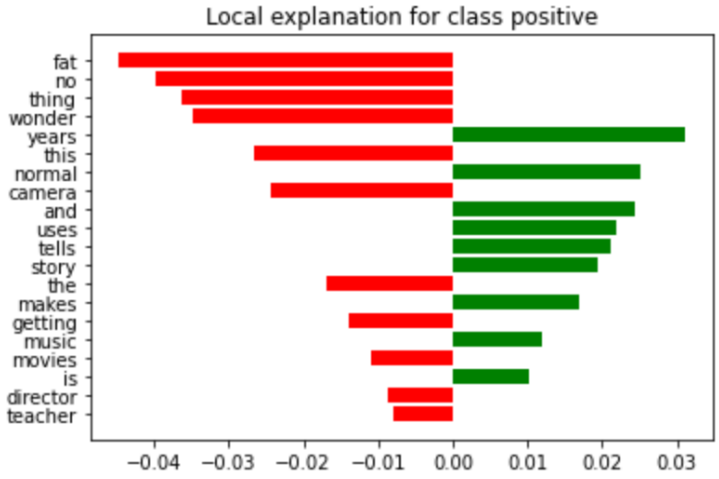

Figure 10: Lime interpretation of a perturbed positive review

In the following items, we investigate each word that the model has changed:

- "student" to "teacher": we see that in the figure 10figure 10, the "teacher" word has a negative sentiment. The "student" word did not appear in figure 9, and it shows that it did not have an important impact. Since "student" and "teacher" are in the same category, and both do not have a negative or positive impact, it shows that the Naive Bayes did not give the same probability to the same category words.
- "james" to "c": interestingly, the name "james" has a strong positive sentiment in the reviews, and when our model changed it to "c," the review sentiment became negative. This finding shows that the Naive Bayes does not understand names, and human names do not change the review's sentiment.
- "films" to "movies": again, movies and films have the same meaning, but we see that "films" has a positive sentiment, and "movies" has a negative sentiment. This finding shows that the Naive Bayes does not have a proper understanding of the language model.
- "feature" to "thing": similar words that show no sentiment but in Naive Bayes, "feature" is a positive word and the "thing" word that has a strong negative impact.
- adding "fat" noise: the Text-GAN model interestingly found the "fat" word has a strong negative impact and added this noise to the review.

To be noted, the perturbed review that we investigated in detail is still a positive review from a human point of view.

6 Conclusions

In this project, we automated generating adversarial examples by using a text-GAN. The Text-GAN algorithm has three different modules: A GAN, A victim model, and an encoder/decoder. The proposed method can be used on any textual victim model. For now, we attacked a sentiment analysis task using a Naive Bayes model. Also, we used an encoder/decoder module to map our sentences to a latent space vector with the ability to decode them. Generally, we generated noisy

reviews and fed them to the victim model to fool the model but not humans. Our takeaways from this project were firstly knowledge expansion. We learned that defining loss function has a significant impact on models' performance. We learned how to define custom losses and apply them in Keras. Also, training VAEs is a time-consuming procedure. Furthermore, we learned to use external servers and work with cloud computing systems. Therefore, in his project, we learned some implementational tricks and several research-oriented points.

7 Discussion

We proposed a method that can automatically find adversarial examples and attack textual victim models, and this method can be used in both good and bad ways. Text-GAN can be useful to be used as a tool to discover the model's weakness points and analyze how well a textual model has learned about the language model. So, text-GAN can be a helpful tool for researchers to find flaws in their models before releasing and can have a positive effect on developing more robust language models. However, text-GAN is subject to misuse. An adversary can use text-GAN to attack other textual models that are incorporated with sensitive data. If our model works fine and finds flaws in a victim model, it will be beneficial for researchers who want to make sure if their model is flawless. Our work will give them a better insight into whether their model is working correctly or not.

Releasing this work may have disadvantages for some researchers since their models are open-source because people can easily attack their models using text-GAN. Therefore, the blind spots of their models will be revealed, and they may suffer from information leakage, which is not desirable for them. Notably, the dataset we used in our work was publicly available, so we are not involved with sensitive information, and there is no ethical concern about that.

8 Future Works

Due to time limits, we could not test our model performance on different victims. We plan to test our model's generalizability by testing it on other victim models(like SVM and Deep Neural Networks) in different NLP tasks(e.g., text generation task).

For the generator performance evaluation, we can use the idea of Inception Score [15] that is used for image data. The details about this approach is explained in Section 11.1.

The statistical test happens to be our next future work. We brought the details of that in Section 11.2, but because executing our code took quite a long time, we could not do a statistical test practically.

As discussed before, the variational autoencoder's accuracy was not sufficient for our task, and we replaced the approach by using pre-trained word embeddings. If we had enough time, we could tune autoencoder hyperparameters to get better results. The other idea that came up to our mind was using pre-trained word embedding in the initial layers of autoencoder (instead of an Embedding layer). By doing so, we may see better results for the autoencoder.

Besides, the GloVe pre-trained embedding can be fine-tuned on our IMDB dataset to be more compatible with our project.

Besides, if we had more time and resources, we could set the number of vocabularies in the IMDB dataset to a larger value, like 10k. But now, due to the constraints, we set limited the number of

words to 2000.

We can also change our GAN model to a WGAN[1] model, which has been proved to be better in results in some of the tasks. We can examine whether it performs better in this task or not.

9 Acknowledgement

Thanks, professor Alona Fyshe for providing computational resources for us from Computing Canada and her effective feedback on our work, which was constructive.

10 Task Distribution

- Literature Review : [3 days] Everyone
- Data preprocssing : [2 days] Mahtab, Zahra
- Implementing GAN: [7 days] Zahra, Fatemeh
- Implementing several victim models: [1 day] Everyone
- Implementing Autoencoder: [7 days] Mahtab
- Implementing pre-trained embedding: [2 days] Mahtab, Zahra
- Merging three differnet modules: [6 days] Everyone
- Setting up computational resources: [3 days] Mahtab, Zahra
- Result analysis including Visualization with LIME, significance test, etc: [2 days] Mahtab, Zahra
- Debugging the code: [3 days] Mahtab, Zahra
- Writing Reports: [4 days] Everyone

It is to be noted that these tasks have been done concurrently, and the timing has some overlaps. Also, each day is representative for 8 hours.

We all three spend the same time amount on the project. Depending on the challenges we faced, some tasks took a long time.

11 Appendix

11.1 Inception score

This is how the inception score works. It uses entropy to measure how an image is probable. This score aims to measure $P(y|x)$, which means how predictable our generated sample (x) is. For calculating this conditional probability, in the Inception Score, they classify the x image to see how high the $P(x|y)$ is. Here, in the case of text data, we use a robust pre-trained classifier trained on the IMDB reviews dataset, and then we get the $P(x|y)$, in which x is a noisy sentence generated by the generator. The higher this probability, the better the performance of the generator.

11.2 Statistical test

We define our statistical test as follows:

- Case 1: We will attack the victim model 20 times with natural examples, and store the model's accuracies in a vector, and calculate the mean value for accuracies.
- Case2: We will attack the victim model using adversarial examples 20 times, and again store the model's accuracies in a vector, and calculate the mean value once again.

In the statistical test, we have a null hypothesis concept which is opposite of the alternative hypothesis, and is a commonly accepted fact that we have to define for this task. Our null hypothesis definition is "the difference of the mean value for accuracies of case 1 and case 2 is less than a p-value. This lets us recognize that if these two cases have the same accuracies, is it by chance or not."

Our alternative hypothesis definition is that "the attack has been effective and the accuracy of the victim model has dropped, and it is not by chance."

We set the significant level to alpha= 0.1, and then we define the p-value as $P(case2|Nullhypothesis)$. After calculating the p-value, if $p - value < alpha$, we reject the null hypothesis, which means that our attack has been effective and the probability of being by chance is low. Otherwise, if $p - value >= alpha$, we do not reject the hypothesis, and it means that we cannot prove the effectiveness of our attack by this test.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[2] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018.

[3] David Donahue and Anna Rumshisky. *Adversarial Text Generation Without Reinforcement Learning*. 2019. arXiv: 1810.06640 [cs.CL].

[4] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[5] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[6] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[7] *IMDB Dataset of 50K Movie Reviews*. 2019. URL: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

[8] *Intuitive Guide to Understanding GloVe Embeddings*. 2019. URL: https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010.

[9] Christopher D. Manning Jeffrey Pennington Richard Socher. *GloVe: Global Vectors for Word Representation*. 2014. URL: https://nlp.stanford.edu/projects/glove/.

[10] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[13] Bin Liang et al. "Deep Text Classification Can be Fooled". In: *CoRR* abs/1704.08006 (2017). arXiv: 1704.08006. URL: http://arxiv.org/abs/1704.08006.

[14] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "" Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[15] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[16] M. Schuster and K. K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[17] Christian Szegedy et al. "Intriguing properties of neural networks". In: *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199.

[18] Chaowei Xiao et al. "Generating Adversarial Examples with Adversarial Networks". In: *CoRR* abs/1801.02610 (2018). arXiv: 1801.02610. URL: http://arxiv.org/abs/1801.02610.

[19] Wei Emma Zhang et al. *Adversarial Attacks on Deep Learning Models in Natural Language Processing: A Survey*. 2019. arXiv: 1901.06796 [cs.CL].

[20] Zhengli Zhao, Dheeru Dua, and Sameer Singh. *Generating Natural Adversarial Examples*. 2018. arXiv: 1710.11342 [cs.LG].