Writing Style Transfer - NLP Final Project

Avraham, Elior
elior.av@gmail.com

Boniel, Natali
nataliboniel@gmail.com

Abstract

This paper focuses on writing style transfer based on non-parallel text. Our task focus on changing the text style, with the challenge of keeping the original content of the text as possible. We propose a sequence to sequence model for learning to transform a text from a source domain to a different target domain in the absence of paired examples. Our goal is to learn the relationship of two given domains and creates two functions that can convert each domain to its counterpart to perform writing style transfer. In this paper we proposed a novel approach to learn cross-domain writing style transformation using Cycle GAN architecture.[1]

1 Introduction

Language style can be different between different people, cultures, and context.

Writing Style Transfer is a way to express text differently by transform the style from one to another (e.g., positive ↔ negative, English ↔ French etc.), with the preservation of its context.

In our work, we will present positive ↔ negative Style Transform implementation on non-parallel data, which means we have only the input data but have no target data to transform it to.
Our motivation for this research was to enhance writing Skills - convert informal to formal writing also, standardization of writing - an entire organization writes with the same standards.
We present the method, data, research, and evaluation.

2 Related Work

Generative Adversarial Networks (GANs): Recent methods adopt the GANs method for conditional data generation applications. The advantage of GANs is the idea of adversarial loss that forces the generated data to be identical to real data. This loss is particularly powerful for image generation tasks (Goodfellow, 2016). We adopt an adversarial loss to learn the mapping of our data text such that the transformed sentence will be identical to a sentence in the target domain.

Text style transfer from non-parallel text: Non-parallel (also called unpaired) style transfer on text has been studied in computer vision before (Shen et al., 2017; Dai et al., 2019) due to the reason that for many tasks, paired training data will not be available. Shen et al. (2017) used cross-alignment for this task. They separated the content from other aspects such as style, assuming a shared latent content distribution across different text corpora, and proposed a method that leverages refined alignment of latent representations to perform style transfer. They introduced the effectiveness of their method on three tasks: sentiment modification, decipherment of word substitution ciphers, and recovery of word order. We used their approach to preparation and classification of the unpaired text data.

Text style transfer combining non-parallel data and Cross-Domain GAN: (Zhu et al., 2017) demonstrate image-to-image translation using Cycle GAN. On their research, they presented an approach of cycle consistency loss for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. They presented their high results on several tasks where paired training data does not exist, including collection style transfer, object transfiguration, season transfer, photo enhancement, etc. On our research we adopt their method, but unlike them, we implemented it on text.

[1] Our code and data available at https://github.com/eliorav/writing-style-transfer

3 Method and Model

Learning to style transfer with only non-parallel data, as we had, is a challenging task since the associated style expressions cannot be learned directly. Therefore we used the Cross-Domain GAN architecture for our model, specifically the Cycle GAN network. We generated our output sentences using a sequence to sequence network trained by GAN architecture.

3.1 GANs

Generative Adversarial Network (GAN) is a generative model that try to understand the inner domain structure by adversarial learning (Goodfellow, 2016). It does that with two competing neural networks designed to generate estimated data related to the test set by learning the datas distribution. The two neural networks in GAN are the generator that generates new data instances that are intended to come from the same distribution as the training data, and the discriminator, that evaluates them for authenticity and determine whether they are real or fake. The goal of the generator is to generate fake data, means to lie without being caught by the discriminator. The goal of the discriminator is to identify data that coming from the generator as fake by learning the real distribution (see Figure 1).

The (D)escriminator and the (G)enerator play the following min-max game:

$$min_G\, max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

Maximizing success of real data classification while minimizing success of fake data discrimination.

The GAN cycle: The generator get Gaussian noise of the domain latent space and returns fake data of the domain representation. This generated data is fed into the discriminator alongside a stream of data taken from the actual ground-truth dataset. The discriminator takes in both real and fake data and returns probabilities with 1 representing a prediction of authenticity and 0 representing fake. According to the discriminator result, the model is fine tune the training (see Figure 2).

GAN Loss function (Goodfellow, 2016): Each network has a different goal, and thus a different loss function:

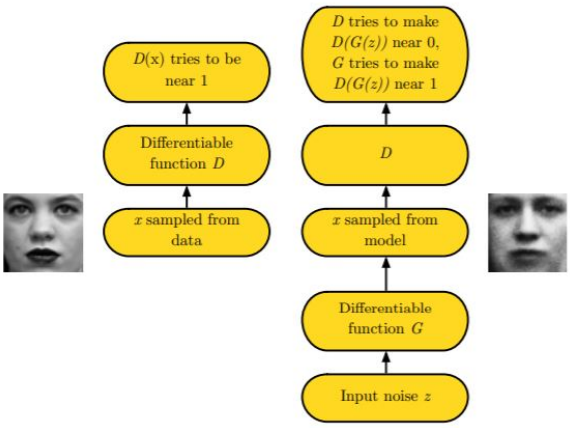

Figure 1: The GAN framework pits two adversaries against each other in a game (Goodfellow, 2016). Each player is represented by a differential function controlled by a set of parameters. Typically these functions are implemented as deep neural networks. The game plays out in two scenarios. In one scenario, training examples x are randomly sampled from the training set and used as input for the first player, the discriminator, represented by the function D. The goal of the discriminator is to output the probability that its input is real rather than fake, under the assumption that half of the inputs it is ever shown are real and half are fake. In this first scenario, the goal of the discriminator is for $D(x)$ to be near 1. In the second scenario, inputs z to the generator are randomly sampled from the models prior over the latent variables. The discriminator then receives input $G(z)$, a fake sample created by the generator. In this scenario, both players participate. The discriminator strives to make $D(G(z))$ approach 0 while the generative strives to make the same quantity approach 1. If both models have sufficient capacity, then the Nash equilibrium of this game corresponds to the $G(z)$ being drawn from the same distribution as the training data, and $D(x) = \frac{1}{2}$ for all x

The Generator loss:
$G_{loss} = \log(1 - D(G(z)))$ or
$G_{loss} = -\log(D(G(z)))$

The total cost:
$\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z^i)))$
or
$\frac{1}{m}\sum_{i=1}^{m} \log - \log(D(G(z^i)))$

The Discriminator loss:
$D_{loss_{real}} = \log(D(x))$
$D_{loss_{fake}} = \log(1 - D(G(z)))$
$D_{loss} = \log(D(x)) + \log(1 - D(G(z)))$

The total cost:

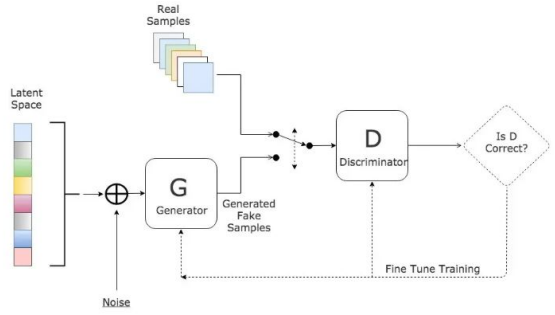

Figure 2: GAN Training Process. There are 3 major steps in the training: 1. Use the generator to create fake inputs based on noise. 2. Train the discriminator with both real and fake inputs. 3. Train the whole model: the model is built with the discriminator chained to the generator

$$\frac{1}{m}\sum_{i=1}^{m}\log(D(x^i)) + \log(1 - D(G(z^i)))$$

Cycle GAN network (Zhu et al., 2017) belongs to the set of algorithms named generative models of the unsupervised learning field, a sub-set of ML which aims to study algorithms that learn the underlying structure of the given data, without specifying a target value.

The Cycle-GAN architecture consists of two GAN networks. GAN model 1: The generator input is data from domain X and outputs the generated data to be from domain Y. The discriminator input is the output from the generator (fake data of domain Y), and real data from domain Y and output likelihood of data is from domain Y. GAN model 2 does the opposite. The network does not require paired data for training (see Figures 3 and 4).

GAN Loss function: The discriminator and generator models for a GAN are trained under normal adversarial loss like a standard GAN model.
Cycle Consistency Loss: to prevent the learned mappings G and F from contradicting each other (measures the L1 distance between x and F(G(x)) as well as between y and G(F(y))).
Adversarial loss: for matching the distribution of generated images to the data distribution in the target domain (both G and F)
Full objective is:

$$Loss(G, F, D_X, D_Y) = Loss_{GAN}(G, D_Y, X, Y) + Loss_{GAN}(F, D_X, Y, X) + \lambda Loss_{cycleGAN}(G, F)$$

3.2 Our Model

3.2.1 Generator

Our generator is sequence to sequence network based on (Sutskever et al., 2014) with noise in the form of 0.5 dropout rate for encoder and decoder. The generator is a regular Autoencoder (Baldi, 2012) consists of encoder in which the model learns how to reduce the input dimensions and compress the input data into a encoded representation as a single vector (context vector-abstract representation of the entire input sentence), and decoder in which the model learns how to reconstruct the data from the encoded representation to be as close to the target output as possible (the fake data of the other domain) by generating it one word at a time. Our encoder and decoder consists of embedding layer and LSTM with two layers. The decoder also consist of linear layer. We expand on the layers in section 5.4.

Prepering the data for training the generator: We tokenize each sentence, means we turn a string containing a sentence into a list of individual tokens that make up that string, e.g. ”good morning!” becomes [”good”, ”morning”, ”!”]. Also, we appends the $< sos >$ and $< eos >$ tokens, and converts all words to lowercase. We built the vocabulary for the source and target domains. The vocabulary is used to associate each unique token with an index (an integer) and this is used to build a one-hot encoding for each token (a vector of all zeros except for the position represented by the index). The vocabularies of the source and target domains are distinct. We only allow tokens that appear at least 5 times to appear in our vocabulary. Tokens that appear lower number of times are converted into the $< unk >$ (unknown) token. The vocabulary built from the training set. This prevents ”information leakage” into the model, giving artificially inflated validation/test scores. Since we used iterator to get batch of examples, we needed to make sure that all of the source sentences are padded to the same length, the same with the target sentences.

Build the generator: We built our model in three parts. The **Encoder**, the **Decoder** and the **sequence to sequence** model that encapsulates the encoder and decoder and provide a way to interface with each. The **Encoder** receive one word at a time. At each time-step, the input to the encoder LSTM is both the current word, x_t, as well as the hidden state from the previous time-step, h_{t1}, and

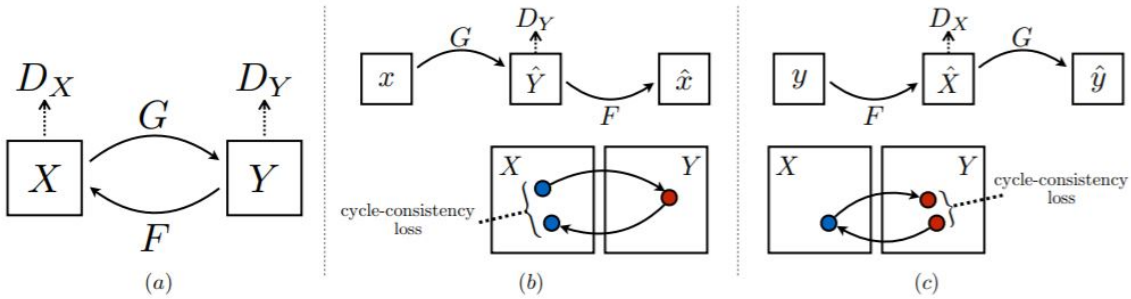

Figure 3: (a) Cycle GAN model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ (Zhu et al., 2017)

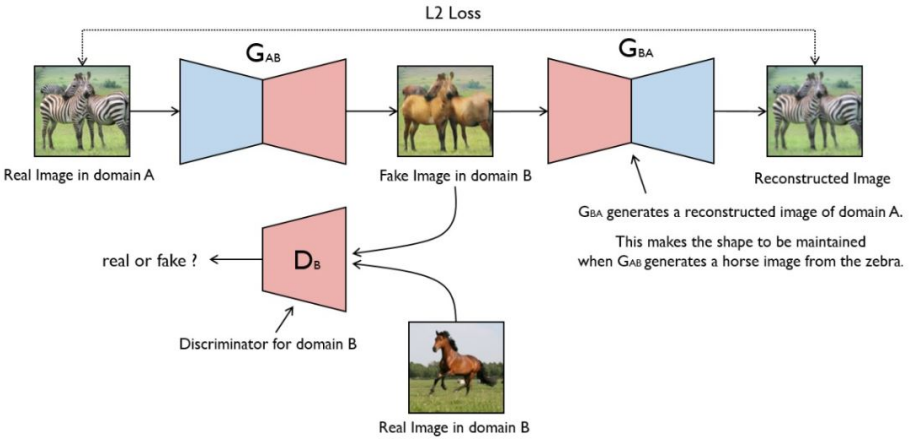

Figure 4: The Cycle GAN cycle, explained by images data

the encoder LSTM outputs a new hidden state h_t. The LSTM can be represented as a function of both of x_t and h_{t1}:

$$h_t = EncoderRNN(x_t, h_{t1})$$

We are using the LSTM (Long Short-Term Memory) LSTM. The initial hidden state, h_0, is usually either initialized to zeros or a learned parameter. Once the final word, x_T, has been passed into the LSTM, we use the final hidden state, h_T, as the context vector, i.e. $h_T = z$. This is a vector representation of the entire source sentence. After we had the context vector, z, we start with the **Decoder**. We decoding z to get the target sentence. Again, we append start and end of sequence tokens to the target sentence. At each time-step, the input to the decoder RNN is the current word, y_t, as well as the hidden state from the previous time-step, s_{t1}, where the initial decoder hidden state, s_0, is the context vector, $s_0 = z = h_T$, i.e. the initial decoder hidden state is the final encoder hidden state. Thus, similar to the encoder, we can represent the decoder as

$$s_t = DecoderRNN(y_t, s_{t1})$$

In the decoder, we need to go from the hidden state to an actual word, therefore at each time-step we use s_t to predict (by passing it through a linear layer) what we think is the next word in the sequence, $\hat{y}_t$.

$$\hat{y}_t = f(s_t)$$

The words in the decoder are always generated one after another, one per time-step. We always use $< sos >$ for the first input to the decoder, y_1, but for subsequent inputs, $y_{t>1}$. In the pre-trained phase, we train the generator as a regular autoencoder, which mean when we get a sentence A, the goal of the generator is to reconstruct the same sentence A from the latent space of the encoder. The goal of this process is to find a better initialization for the generator. In this process, we sometimes use the actual, ground truth next word in

the sequence, y_t and sometimes use the word predicted by our decoder, $\hat{y}_{t-1}$. This is called teacher forcing (see Figure 5).

Pre-Train the generator: As mentioned before, the input and output dimensions are defined by the size of the vocabulary. The embedding dimensions and dropout for the encoder and decoder can be different, but the number of layers and the size of the hidden states must be the same. We then define the encoder, decoder and then our Seq2Seq model. Then we initialized the weights of our model. In the paper they state they initialize all weights from a uniform distribution between -0.08 and +0.08, i.e. $\mathcal{U}(-0.08, 0.08)$. We define ADAM optimizer, which we use to update our parameters in the training loop. Next, we define our loss function. The CrossEntropyLoss function calculates both the log softmax as well as the negative log-likelihood of our predictions. When training our model, we always know how many words are in our target sentence, so we stop generating words once we hit that many. Our loss function calculates the average loss per token, we ignore the loss whenever the target token is a padding token. When we trained our model, we always knew how many words are in our target sentence, so we stop generating words once we hit that many. Once we have our predicted target sentence, $\hat{Y} = \hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_T$, we compare it against our actual target sentence, $Y = y_1, y_2, \ldots, y_T$, to calculate our loss. In our case, we expected the output to be the same as the output, then we use this loss to update all of the parameters in our model by the ADAM optimizer.

3.2.2 Discriminator

Our discriminator network is a sentiment classifier based on (Joulin et al., 2016). It consist of embedding layer and linear layer. We use bag of bi-grams (i.e. pair of words/tokens that appear consecutively within a sentence) to capture some partial information about the local word order. The bigram features are embedded and averaged to form the hidden variable, which is in turn fed to a linear classifier (see Figuer 6). We train the network with ADAM optimizer and use the BCEWithLogitsLoss loss function that carries out both the sigmoid and the binary cross entropy steps.

3.2.3 Hyper-Parameters

We set the hyper parameters according to the recommendations in other works and by trial and error (see Table 1).

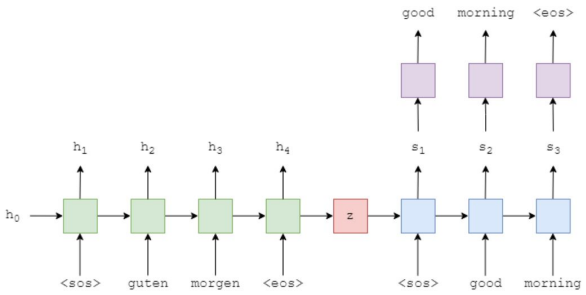

Figure 5: Example of sentence translation. It is the same process for our style transform task. The origin sentence, "guten morgen", is input into the encoder (green) one word at a time. At each time-step, the input to the encoder RNN is both the current word, as well as the hidden state from the previous time-step, and the encoder RNN outputs a new hidden state. Here, we have $X = x_1, x_2, \ldots, x_T$, where $x_1 =< sos >$, $x_2 = guten$, etc. When we have our context vector, z, we start decoding it to get the target sentence, "good morning". Again, we append start and end of sequence tokens to the target sentence. At each time-step, the input to the decoder RNN (blue) is the current word, as well as the hidden state from the previous time-step, where the initial decoder hidden state is the context vector. At each time-step we predict what we think is the next word in the sequence by passing it through a linear layer (purple)

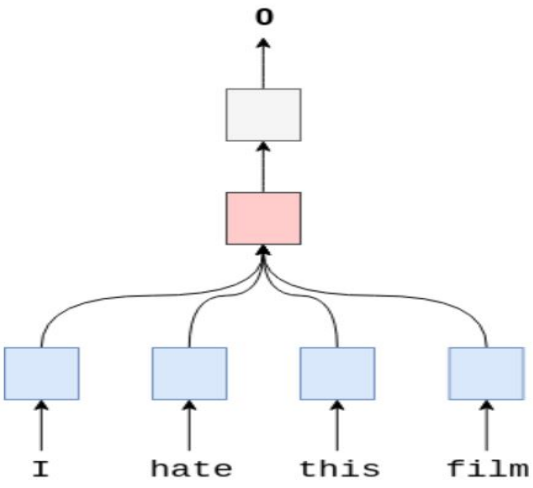

Figure 6: The Discriminator network. We calculates the word embedding for each word using the embedding layer (blue), then calculates the average of all of the word embedding (pink) and feeds this through the linear layer (gray)

Parameter	Meaning	Value
batch_size	size of the batches	16
n_epochs	number of epochs for cycle GAN training	50
g_n_epochs	number of epochs for generator training	10
decay_epoch	epoch from which to start learning rate decay	25
lr	adam: learning rate	0.0002
lambda_cyc	cycle loss weight	8.0
lambda_id	identity loss weight	5.0
lambda_adv	generator adversarial loss weight	1.0
b_1	adam: decay of first order momentum of gradient	0.5
b_2	adam: decay of first order momentum of gradient	0.999
enc_emb_dim	encoder embedding dimension	256
dec_emb_dim	decoder embedding dimension	256
d_emb_dim	discriminator embedding dimension	256
g_hid_dim	the hidden dimension of the encoder's/decoder's LSTM layer	512
g_n_layers	the number of hidden layers in the encoder/decoder	2
enc_dropout	dropout value for the encoder	0.5
dec_dropout	dropout value for the decoder	0.5
teacher_forcing_ratio	the probability to use teacher forcing	0.5

Table 1: The hyper-parameters of our model, set according to the recommendations in other works and after trial and error

4 Data

We run our experiments on YELP reviews dataset (licence) which does not reveal personal or private information.
The data set represent in JSON format and includes 5.2M reviews. Each review consists of several properties, the main ones are stars rating, useful, funny, cool and the text of the review. We concentrated on the stars rating and the text properties.

Data Preparation: Our final dataset, with which we trained and tested the model, was obtained after extracting, filtering and refining the original YELP dataset. We followed after (Shen et al., 2017). First we extract all sentences in the dataset by dividing each text of review to sentences. Given that neutral sentences are more common in long reviews, filter out all text of review that exceed 20 sentences (in the paper they filtered out if exceed 10 sentences). Then we further filtered the remaining sentences by eliminating those that had below 2 words and exceed 15 words, in order to remain meaningful sentences on the one hand, and on the other hand not too long because long sentences tent to be neutrals and the sentiment is less clear. Finally, we classified each sentence by the stars rate of the review. We classified positive sentences by stars rating of 5, which considered as positive review, and negative sentences by stars rating of 1, which considered as negative review, differently from (Shen et al., 2017), assuming all the sentences in a review have the same sentiment. This is clearly an oversimplification,since some sentences are sentiment neutral.
In order to improve the sentences division to each category, we tried to add another stage by adding classification to the divided datasets with a classifier that will help to filter out positive sentences from the negative dataset and the opposite, but we didn't get good results so we continue with the division we already had.

Dataset: We divided the data to 100K sentences for each category (positive or negative) for the train set, 20K sentence for each category for the validation set and 20K sentence for each category for the test set. Overall 200K sentences for training and 40K sentence for validation and test each.

5 Experiments

We applied our method to the positive $\leftrightarrow$ negative Style Transfer task. We report the BLEU score of these methods and visualize the resulting sentence representation.

5.1 Training the Discriminator

As we describe above, the discriminator is a classifier, usually a neural network, that outputs the probabilities for the sample being real or fake. Therefore, we searched after state of the art classifier. We found a comparison of error rate across different models, based on the results of reliable papers or sources. XLNet (Yang et al., 2019) was found to have the lowest error rate (NLPprogress, 2019) (see Figure 7) and so we first built the discriminator as XLNet. When we tried to train the model, the performance was low, we got stuck at a local optimum and the training lasted for hours and did not progress. From research we have done, we have found at (Yang et al., 2019) that the network training is not privately possible because it requires a huge computing power of 512 TPU v3 chips for 2.5 days, 245,000$ is required. Also, using a pre-trained XLNet was not possible because it is not recommended to use GAN on trained networks. Therefore, we built the discriminator network as described at the model section.

5.2 Training the Generator

In our first attempt to train the GAN (i.e. adversarial training of the generator and the discriminator) we noticed that the generator is not strong enough to compute against the discriminator since the discriminator error rate close to zero (always win the generator). We found that the generator is weak because it got gibberish and wrong classified sentences at the beginning of the training which made the discriminator beat the generator from the beginning (it was easy to recognize that G was lying) without having the ability to recover. Our goal was to strengthen the generator starting point against the discriminator, so we pre-trained it as a regular Autoencoder (Baldi, 2012) that learn a representation (encoding) for a domain and output the same input compressed. So in our final model we first training the two generators (positive to negative, negative to positive) as regular Autoencoder, and after that we train the Cycle GAN.

5.3 Prepare the Data

In our first attempt to classify the sentences to positive and negative domain, we followed (Shen et al., 2017) which says that after following standard practice, reviews with stars rating above three are considered positive, and those below three are considered negative. We divided our data as following, but we sow that the data needs more preparation because the negative data contained a large amount of positive sentences, and the opposite in the positive data. It caused to low performance and the network handled with this cases differently from time to time and not consistently (e.g positive sentence in negative data "i love the food at panera" changed to "i love the food at checkout"). We tried to clean the data manually, but the sentences amount we needed was too big and the cleaning took way too much time. Also, we tried to pre-trained a sentiment classifier, based on the same structure as the discriminator by using IMDB reviews dataset. The goal is to eliminate negative sentences from the positive dataset, and the opposite from the negative dataset. The effectiveness of the classifier was not sufficient enough. Instead, we classified the sentences only by their starts, a sentence that belongs to review with five starts consider as positive and sentence that related to review with one start consider as negative. By using this method, we manage to significantly reduce the number of sentences that do not belong to the domain.

5.4 Fine Tune of the Hyper Parameters

We first set the hyper parameters according to recommendations in previous works and we performed trial and error until we reached the final values. We also tried to improve results by increasing the number of layers of the encoder and decoder networks from two layers to four layers each (total of 8 layers) so that they could learn larger and stronger elements, but the experiment did not produce better results than we did before.

6 Evaluation

We used the BLEU score for the results evaluation on each generator for the test phase. BLEU is a popular evaluation metric for automatic evaluation of neural machine translation and many related papers using this metric as the evaluation metric. A high BLEU score primarily indicates that the system can correctly preserve content by retaining the same words from the source sentence as the reference.

7 Discussion

We measure our result in two aspects: sentences transformation and the BLEU score.

Model	Error	Paper / Source
XLNet (Yang et al., 2019)	27.80	XLNet: Generalized Autoregressive Pretraining for Language Understanding
ULMFiT (Howard and Ruder, 2018)	29.98	Universal Language Model Fine-tuning for Text Classification
DPCNN (Johnson and Zhang, 2017)	30.58	Deep Pyramid Convolutional Neural Networks for Text Categorization
CNN (Johnson and Zhang, 2016)	32.39	Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings
Char-level CNN (Zhang et al., 2015)	37.95	Character-level Convolutional Networks for Text Classification

Figure 7: Models evaluation based on error (1-accuracy; lower is better) on YELP dataset

Sentences transformation We can see that overall the model shows transformed sentences with good results. The model seems to have learned to include words, like 'always' and 'never', and understood parallel words, like 'amazing', 'awesome', 'fabulous' and 'terrible', 'awful'. Table 2 show some samples of sentences that have been transformed by our model.

We can also see that, although the data was not clean enough and it contained sentences with sentiment of the other domain (i.e. positive sentence in negative data and the opposite), the model handled well with those sentences by keep them as they were originally and not transform them (see some examples in Table 3).

On the other hand, we can see that the model sometimes found it difficult to understand the sentence context When it comes across ambiguity words or when it makes a sentence meaningless, apparently when it doesn't understand the meaning of certain words or doesn't know them at all. In Table 4 we present some examples of an unsuccessful examples.

BLEU score We can see that the results were good, but less than what we were aiming for. In table 5 we present the scores we got for each criterion. Figure 8 show the BLEU score per sentence length. We assume that mainly the size of the data we could train with the model and the requirement that the lengths of sentences be identical, which created a lack of flexibility in the transformation of the sentences in terms of extending or shortening the original sentence, led to these results. In section 8 (Suggestions for Future Work) we suggest several options to improve the results.

8 Conclusion

Writing style transfer using non-parallel data is a relatively rare field in the transferring language research. In this work, we executed the task with access only to non-parallel data and with deep learning networks that didn't trained before for this kind of task on text.

We presented our results, measured by the BLEU score. Our task, of transforming one text style to another from non-parallel data, was a challenging task to evaluate, since the BLEU score measurement required actual target, that we didn't have. Therefore, we calculated the BLEU score of the transformed sentence against the original one. That means that a BLEU score of 100 is not sufficient for the evaluation (the sentence does not change at all), but still, we aim for the higher score as possible.

We have presented innovation and originality in the model by converting and implementing a state-of-the-art image transformation method to a text transformation method.

Suggestions for Future Work

In the future, we would like to improve our model and results in some ways. (a) Further refining and preparation of the data to be more qualitative, as we tried in this work but had not enough resources to put this effort. (b) Training the model with biggest dataset. In our research, we have limited it to due to computing constraints. (c) Use more sophisticated networks for the model components,

From positive to negative

Original	Transformed	BLEU score
loved this place	hate this place	44.105042
definitely will return !	definitely not return	23.265897
the service was good and the food was great	the service was mediocre and the food was terrible	39.281465
staff is really friendly and helpful	staff is so rude and par	24.733286
there is no waiting	there is no answer	38.44001

From negative to positive

Original	Transformed	BLEU score
will never be coming back	will be coming back	38.44001
this place is disgusting !	this place is great !	38.44001
worst service i've had in a long time	best service i've had in a long time	88.011174
service was so slow !	service was so fast !	38.44001
will not be back !	will definitely be back !	38.44001

Table 2: Examples of style transformed sentences produced by our model alongside the original sentence and the BLEU score

Domain	Sentence
Positive	that one sucks
Positive	drinks super expensive
Negative	hotel is nice
Negative	great location and atmosphere

Table 3: Examples of un-transformed sentences because they had sentiment from the other domain and not the domain where they were found. BLEU score is 100 since haven't change any word in the sentence

Original	Transformed
very reliable , professional and efficient	very condescending , & disappointed
everyone here is fabulous	everyone here is unacceptable
the service is great too	the service is too ?
seriously , go now	seriously , go tonight
basically a repeat of others	actually a variety of mine
would n't go back	would n't go wrong

Table 4: Examples of unsuccessful style transformed sentences produced by our model alongside the original sentence and the the model difficulty

Criterion	Score
loss GAN from positive to negative	2.309
loss GAN from negative to positive	1.480
BLEU score generator of positive domain	37.463
BLEU score generator of negative domain	40.092

Table 5: Losses and BLEU score for each criterion after test

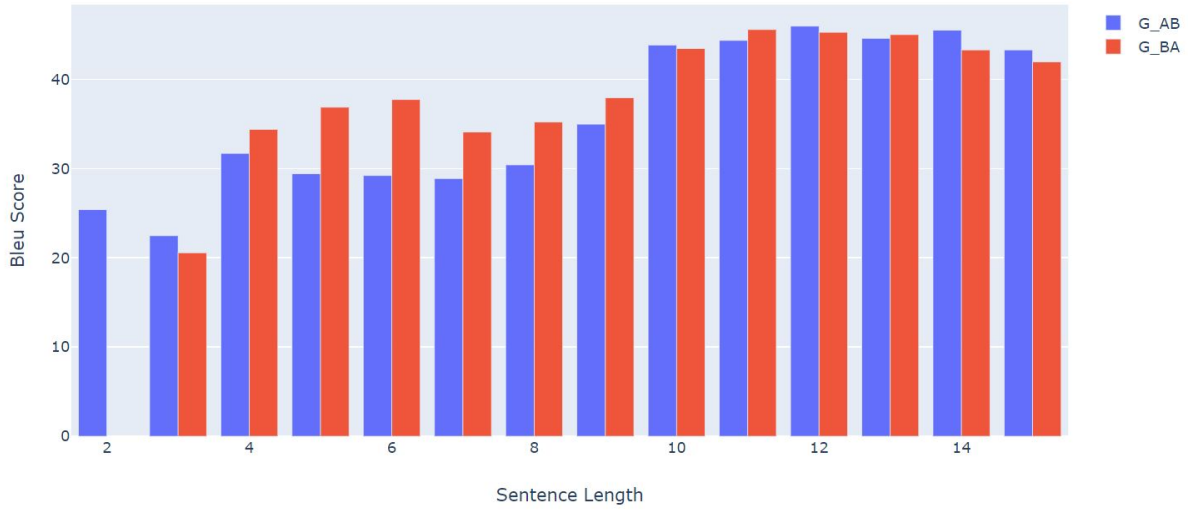

Figure 8: Performance of our system as a function of sentence length, measured by BLEU score. The x-axis corresponds to the test sentences sorted by their length and is marked by the actual sequence length

but keep it simple and not computation expensive. (d) Try to learn more sophisticated ways to train GAN. GAN and Cycle-GAN are not easy deep learning architectures to train. Therefore we can try to implement several improvements to the training process, and maybe it will affect the conversion of the models. In our work, we apply some optimization. The optimization includes pre-train the generator, keeping dropout also in the validation phase, construct different mini-batches for real and fake sentences, and use the ADAM optimizer.

References

Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49.

Ning Dai, Jianze Liang, Xipeng Qiu, and Xuanjing Huang. 2019. Style transformer: Unpaired text style transfer without disentangled latent representation. *arXiv preprint arXiv:1905.05621*.

Ian Goodfellow. 2016. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

NLPprogress. 2019. *Models Evaluation on YELP dataset*.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in neural information processing systems*, pages 6830–6841.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232.