Statistical NLP - Final Project
Language Style Transfer

David Martuscello
Courant Institute of Mathematical Sciences
New York University
dm4350@nyu.edu

Fang Han Cabrera
Courant Institute of Mathematical Sciences
New York University
fh643@nyu.edu

Jiyuan Lu
Courant Institute of Mathematical Sciences
New York University
jl11046@nyu.edu

Abstract

The task to transfer styles of texts is proved to be challenging because the style and content interact in subtle ways and there's a lack of parallel data and reliable evaluation metrics. In order to better understand the complexity of this problem, we decided to re-implement the model described in Zhou et al. (2018).

1 Problem Setting

Our project was to re-implement the model discussed in the paper *Language Style Transfer from Sentences with Arbitrary Unknown Styles* (Zhou et al. (2018)). The code from this paper is not publicly available, so our implementation was strictly based on the research paper provided.

Language style transfer is the task of generating a new sentence that preserves the content of the source sentence while emulating the style of a target domain. It is an important component of natural language generation that facilitates many NLP applications, such as automatic conversion of paper title to news title, which reduces the human effort in academic news report. For tasks like poetry generation (Yan et al. 2013; Yan 2016; Ghazvininejad et al.2016), style transfer can be applied to generate poetry in different styles.

Style and Content A major challenge of style transfer is to separate style from the content. In computer vision, Li et al. (2017) proposes an expression to distinguish style and content of a picture. However, this is under-explored in the NLP community. How to separate content from style in text remains an open research problem in text style transfer. One effective approach proposed in Shen et al. (2017) maps between sentences and continuous latent representations and then decompose the latent representations into style and content. The generative assumption adopted is as shown below, which we'll use in our implement.

- `latent style variable` $y \sim p(y)$
- `latent content variable` $z \sim p(z)$
- `sentence` $x \sim p(x|y, z)$

Evaluation is also a key challenge in style transfer. In machine translation and summarization, researchers use BLEU (Papineni et al. 2002) and ROUGE (Lin 2004) to compute the similarity between model outputs and the ground truth. However, we lack parallel data for style transfer to

provide ground truth references for evaluation. The same problem also exists in style transfer in computer vision. To solve this problem, we adopted style discrepancy loss to measure similarity between the style encodings of the target sentence and source sentence, as well as a cycle consistency loss that encourages the generated sentence to preserve the content of the source. Both these metrics will be discussed in detail in 4.

2 Related Work

The task of style transfer has been predominantly explored by transferring style between images. However, many of the approaches used for image style transfer can be adapted to work with language. The model in Zhao et al. (2018) adapts the ideas of generative adversarial networks (GANs) as proposed in Goodfellow et al.(2014) and cycle consistency loss as proposed in Zhu et al. (2017).

Most of the work in this area has made the assumption that the source domain has one consistent style and the target domain has its own consistent style. The paper we are focused on (Zhao et al. (2018)) loosens these assumptions to accept any source data, regardless of style.

One example of related work that takes a different approach is Subramanian et al (2019). This paper achieves state-of-the-art results on this problem and they enable the ability to manipulate multiple attribute of the generated style. This is done by replacing the adversarial training approach with a back-translation approach. Back-translation has been demonstrated to work well for unsupervised machine translation and is now being applied to language style transfer. The framework is based on mixing a denoising auto-encoding loss with an online back-translation technique. We did not pursue this approach because extensive human labeling evaluation metrics are used and this would not be possible given our resources.

3 Baselines

We will use the model discussed in Shen et al. (2017) as our baseline model. This model was treated as a baseline in our source paper (Zhao et al. (2018)) which provides accuracy values that we can compare against. The code and data from Shen et al. (2017) is publicly available which would allow us to run this model on other datasets as well, making this baseline suitable for future work.

4 Proposed Approach

We plan to implement the model as described in Zhao et al. (2018). Once the model is successfully implemented we may replace the RNN components with Transformer models and compare the change in performance with these different architectures. We will also compare these results with the baseline model as discussed above.

Model Architecture Now we will discuss the model as proposed in Zhao et al. (2018). The model uses an encoder-decoder framework to disentangle the style (y) from the content(z) of a sentence. There is a style encoder $(E_y(x))$ and a content encoder $(E_z(x))$. The two encodings are fed into a decoder that is trained to recreate the sentence. In order to perform style transfer on a source sentence x_s, the decoder will use the content encoding of the source sentence, $z_{source} = E_z(x_{source})$, along with the style encoding of the target domain y^*. This style encoding is a fixed vector that has been trained to represent the overall style of the target domain. The overall layout of the model is shown in Figure 1.

There are four components to the overall loss function including reconstruction loss, adversarial loss, style discrepancy loss and cycle consistency loss. These components each serve to help the model make useful latent representations and generate sentences with transferred style. These four losses are combined into a single objective with parameters to tune the contribution of each component.

The reconstruction loss encourages the model to reconstruct the input. This loss function measures how well the source sentence can be recreated from only its style and content representations. The purpose of this loss is to generate accurate representations of the style and content.

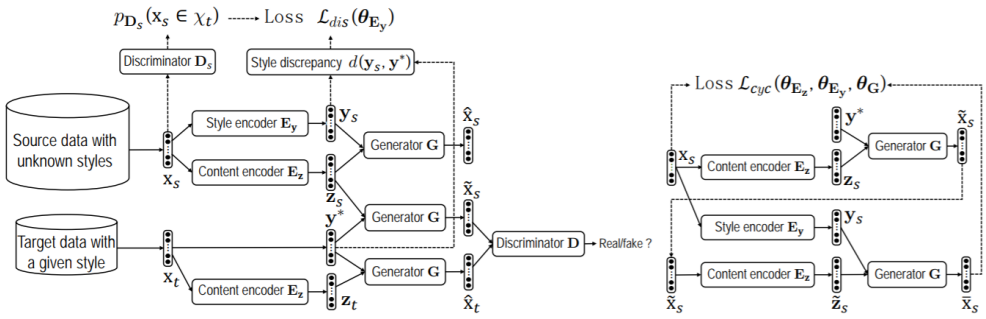

Figure 1: Model Diagram

The adversarial loss encourages the generated sentence to have the same style as the target domain. A discriminator is a classifier trained to distinguish between the real and fake sentences. The real sentence is that which was generated directly from z_target and $y*$. The fake sentence is that which was generated from z_ource and $y*$. The generator (decoder) is trained to fool the discriminator by generating sentences that are indistinguishable from the target domain.

A typical encoder-decoder structure would require a reconstruction loss and an adversarial loss to generate sentences that are similar to the target domain. However, the problem is under-constrained in two respects. First, the representations will not necessarily represent style and content unless they are explicitly told to do so. Second, the content may vary from the source sentence to the generated sentence. The following two losses address these two issues.

The style discrepancy loss tries to relate the style encoding of the target sentences with the style encoding of the source sentences. The style discrepancy is measured by taking the l_2 norm of y_s and y^*. If this measure . In order to link this discrepancy to the actual style property of the data it need to be combined with a classifier that can actually detect differences in style. This classifier is a discriminator D_s that is trained on a portion of the training data to predict the probability that a given sentence has the target language style. This probability is then multiplied by the style discrepancy to produce the final loss. This has the effect of making sure that the style representation corresponds to the actual style of the sentence in such a way that the target style can be distinguished from some arbitrary unknown style.

The cycle consistency loss encourages the generated sentence to preserve the content of the source sentence. This is a concept borrowed from Zhu et al. (2017) where the model is shown to recover the original sentence in a cyclic manner. As can be seen on the right side of Figure 1, the generated sentence, $\tilde{X}_s$, is fed back into the encoder-decoder setup to see how well it can recreate the source sentence that it was generated from. This is effectively undoing the generation process to make sure that the content of the sentence is preserved.

5 Yelp Dataset

In order to directly compare our evaluation result with Zhao et al. (2018), we plan to use the same English dataset which contains Yelp reviews provided in the Yelp Challenge Round 10.

The raw data is composed of reviews from 1 star to 5 stars where we consider reviews of 4 or 5 stars as positive, 1 or 2 stars as negative, and 3 stars as neutral. Zhao et al. (2018) constructed two datasets from the raw data as shown below:

- Positive+Negative: positive reviews are added into the negative so that the source domain contains data with both positive and negative sentiments.
- Neutral+Negative: neutral and negative data are combined together as the source data.

However, for the purpose of our experiments, we only used the Positive+Negative data, because the Neutral+Negative dataset is considered harder to learn.

5.1 Data Preprocessing

The actual data we used was preprocessed and released by Shen et al. (2017), which contains 250k negative sentences and 350k positive sentences.

5.2 Padding

All input sentences are padded to have an uniformed length 20.

6 Implementation

The original paper was implemented in TensorFlow, however the code was not publicly available. We decided to make our implementation in pytorch because we were more familiar with it, and it is more widely used in the academic community, making it more likely to be useful for other researchers.

We built the code from scratch including the data pipeline and the modules themselves. We also had to implement a training loop with 4 different optimizers to update the parameters for different parts of the model. This was a great way to get a feel for the different styles of loss functions and various ways to train a model. For example, one section of the model is set up like an autoencoder with a reconstruction loss. Another section is setup as a Generative Adversarial Network(GAN) with a generator and discriminator.

The model implementation is described in the following sections.

6.1 Preprocessing

Embedding Layer We used the GloVe pretrained word embeddings as in the original paper. The first step was to create a pytorch dataset that loads in the yelp data one sentence at a time. This includes a "get item" function which allows the sentences and their labels to be retrieved in batches from a training loader. Once the batch is retrieved, the words are converted to indices in a lookup table containing all the available GloVe word vectors. These indices can then be passed to a word embedding layer where they are converted into the actual word vectors. If the word in the input sentence does not appear in the GloVe vocabulary it is replaced by a UNKNOWN vector which is an average of all the word vectors in the GloVe dataset.

Initially, we attempted to use a transform to convert each word to its corresponding word embedding. However, this did not allow us to train the embeddings once they were loaded. Our final approach, while slightly more complicated, gives us the option to train the embeddings further if necessary.

Data Loader We created three primary data loaders. First, the pretrain loader takes a subset of the data aside to be used for training the pretrained discriminator for the style discrepancy loss. The other two data loaders hold the source and target datasets. These loaders allow the data to be distributed in batches to the model.

6.2 Modules

Style Encoder The style encoder E_y is a CNN that takes the input sentence and converts it to a single vector. We followed the architecture specified in Kim(2014) for the implementation of CNN and used filter sizes of $200 \times \{1, 2, 3, 4, 5\}$ with 100 feature maps each.

Content Encoder The content encoder E_z is implemented using an RNN where the last hidden state is used as the content encoding.

GRU In both the style and content encoder, GRU is used as the encoder cells. Dropout is applied to the GRUs in both the encoder and the decoder with probability set to 0.5.

Generator The style encoding and content encoding are then concatenated to create a single representation of the sentence. This can then be passed to the Decoder(Generator) as the initial hidden state. The Generator module for this model is also the Generator for the adversarial loss. Generation cells in the decoder are implemented as GRU as well.

Discriminator Discriminator D_s is pre-trained with an implementation similar to the style encoder E_y except that the filter sizes are $200 \times \{2, 3, 4, 5\}$ with 250 feature maps each.

Optimizer The model is trained using the Adam optimizer with an learning rate of 10^{-4}.

6.3 Loss Implementations

Adversarial Loss The discriminator and generator are setup with an adversarial loss. The discriminator is trained with two paths. First, the discriminator is trained to output 1 for sentences that were generated from target data. The style encoding is a fixed vector in the style latent space, while the content encoding is generated from the target sentences content encoder. Then, the discriminator is trained to output 0 for sentences were generated from source data. These sentences combine the fixed target style discussed above, with the content encoding generated from source sentences passing through the content encoder. The same generator is used for both of these processes. The encoders and generators are detached from the graph to limit the computation in the backward call. The optimizer applied to this loss only trains the parameters that belong to the discriminator.

To train the generator and encoders, three loss functions are used. The reconstruction loss, the cycle consistency loss and the discriminator loss.

Reconstruction Loss The reconstruction loss compares the output of the generator with the input from the embedding layer to determine if the sentence can be reconstructed from the encoding. This treats the encoder-decoder setup as an autoencoder where the generator(playing the role of decoder) tries to reconstruct the input sentence from the encoding. This loss is only done with the source sentences. This loss is implemented with a mean squared error loss.

Cycle Consistency Loss The cycle consistency loss takes the input from the generator and passes it back through the content encoder in a cycle. It basically swaps the style of the sentence and then swaps it back to ensure that model is reversible and does not degrade the representation too far.

Discriminator Loss The discriminator loss trains the generator and encoders to produce good sentence representations. This is the purpose of the discriminator to make sure that the generated sentences are convincing. This loss is implemented with a binary cross entropy loss.

Style Discrepancy Loss Finally, the style discrepancy loss first has a pretraining step that trains a discriminator to detect if the source sentence is positive or negative. This independent classification is used to tell style encoder how far the sentence should deviate from the target style, given this information. The same discriminator model and loss was used to train this model. This loss had its own optimizer because it is only used to train the style encoder. Unfortunately, this loss was not completed. The loss functions were implemented separately and the general training flow was laid out including pretraining, but the implementation was not finished.

7 Results

We were able to recreate the model according to the specifications provided in the original paper. In order to run the model we did modify some of the hyperparameters to attempt

8 Obstacles

We ran into two main problems through the course of this project. The first was incorporating the glove embeddings into the project. The second was dealing with limited resources to test our model.

Word Embeddings The inital attempt to use pretrained word embeddings was to use a pytorch Transform to convert each word into a GloVe embedding using a simple lookup. This however did not provide us the ability to train these embeddings. It also did not allow us to easily pad the vectors. For both of these reasons we decided to implement a word embedding layer. This change took a significant amount of time to get working due to lack of familiarity with pytorch.

Limited Resources Due the the fact that we used embeddings in high dimensions (as big as 1500 when the style and content representations are concatenated together), we constantly ran into `cuda out of memory` errors. We felt that it was hugely constraining not being able to fine-tune how pytorch manages gpu memories. When large tensors are passed onto the device, we couldn't control whether they're stored on the CUDA global memory or shared memories. It'd have been of great help if we were allowed to leverage the more recent CUDA unified memory to alleviate our insufficiencies.

9 Conclusion

We were able to complete our implementation of the model and training loop as planned. Unfortunately, we were unable to test our model due to running out of time and compute resources to do so. In the end, we learned a great deal about the process of adversarial training, autoencoders, and pytorch itself. The practice of implementing this model from scratch provided a view of the different components of a neural NLP project and some of the pitfalls that may appear along the way.

References

[2018] Zhao, Bi, Wei, Cai, Liu, Xiaojiang, Tu & Shi Language Style Transfer from Sentences with Arbitrary Unknown Styles. *arXiv:1808.04071v1*

[2014] Kim, Yoon; Convolutional neural networks for sentence classification. In *Proceddings of the Conference of Empirical Methods in Natural Language processing* pages 1746-1751.

[2013] Yan, R.; Jiang, H.; Lapata, M.; Lin, S.-D.; Lv, X.; & Li, X. 2013. i, poet: Automatic chinese poetry composition through a generative summarization framework under constrained optimization. *IJCAI, 2197 −2203.*

[2016] Yan, R. 2016. i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. *IJCAI, 2238 −2244*.

[2016] Ghazvininejad, M.; Shi, X.; Choi, Y.; and Knight, K. 2016. Generating topical poetry. *EMNLP, 1183−1191*

[2017] Shen, Tianxiao; Lei, Tao; Barzilay, Tianxiao $ Jaakkola, Tommi. Style Transfer from Non-Parallel Text by Cross-Alignment. *NIPS 2017. arXiv*

[2017] Fu, Zhenxin; Tan, Xiaoye; Peng, Nanyun; Zhao, Dongyan; Yan, Rui. Style Transfer in Text: Exploration and Evaluation *arXiv:1711.06861v2*

[2017] Zhu, Jun-Yan; Park, Taesung; Isola, Phillip. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the International Conference on Computer Vision*.

[2019] Sandeep Subramanian, Guillaume Lample, Eric Michael Smith, Ludovic Denoyer, Marc'Aurelio Ranzato, Y-Lan Boureau. Multiple-Attribute Text Style Transfer *arXiv:1811.00552v2*