Deep Text Style Transfer

Oron Ashual, Tom Jurgenson, Daniel Grinberg
School of Computer Science
Tel Aviv University
{oronashual,tomjurge,grinberg}@mail.tau.ac.il

September 24, 2017

Abstract

We propose to transfer the content of a text written in a certain style to an alternative text written in a different style, while maintaining as much as possible of the original meaning. Our work is inspired by recent progress of applying style transfer to images, as well as attempts to replicate the results to text. Our model is a deep neural network based on Generative Adversarial Networks (GAN). Our novelty is replacing the discrete next-word prediction with prediction in the embedding space, which provides two benefits (1) train the GAN without using gradient approximations and (2) provide semantically related results even for failure cases.

1 Introduction and Related Work

In this work we propose a method to transfer the content of a sentence from a source domain of one writing style to a target domain of another style. This objective is challenging because the model should capture the style of both the source and target domains, understand the content of a sentence, and strip the sentence of the source style while applying the target style. Furthermore, the model is unsupervised, meaning it does not require aligned sentences between the source and target domain, since parallel text is not usually present for real world datasets.

Recent years have brought the rise of new techniques for implementing deep neural networks, specifically variants of generative adversarial networks (GANs). GANs are powerful generative models and have already shown promising results for image generation ([13], [9]), as well as applying style transfer for images, videos and text ([15] [14]). Images (and videos), have the benefit of being more continuous due to their resolution. In text, however, pixels are analogous to words, hence the media is very discrete, making the objective function of the model harder to train. Next-word prediction is a discrete action that, when used in a text generating model, causes the gradient descent training process to rely on methods to approximate the gradient. In contrast, when doing prediction in the continuous space gradients can be calculated analytically and this problem is abolished. Several approaches for this problem were offered ([10], [17], [2], [5]), for example [14] has shown how to transition from the discrete prediction to continuous prediction by working with probability vectors instead of one-hot vectors. We propose to use the word embedding space as the continuous space for the prediction. This provides the following benefits:

1. The embedding space contains semantic information, meaning, that even if the predicted word is not correct, it is probably in the semantic proximity of the expected word. For models which output word probabilities, these relations would have to be learned between all pairs of words.
2. The embedding space will be the input and output of the model, and since it is usually much smaller than the size of the vocabulary, the task should be relatively easier. Also, most textual models use an embedding as the initial step in the model in order to benefit from the

semantic relation captured by the space. Thus, as opposed to other models, the input and the output of our model is naturally aligned.

Finally, our model is not required to perform two-way transfer - a sentence is transferred from the source to the target and not vice versa, making the model simpler.

2 Background

Recurrent Neural Networks: As opposed to classical feed-forward neural networks, RNNs cell connections may form directed cycles. This allows the network to evolve over time by using its internal memory to process arbitrary length sequences of inputs.

LSTM and GRU cells: Long Short-Term Memory (LSTM) [8, 6] and Gated Recurrent Units (GRU) [3], are two kinds of RNN cells which allow the network to retain important information and drop irrelevant information from the input and/or from the previous time steps. They are specifically built to allow gradients to better flow between time steps and thus overcome the vanishing gradient problem [6] that usually occurs in RNNs.

Generative Adversarial Networks: A GAN [7] is a class of unsupervised learning mechanism that is designed to create a generative model. It is comprised of two components: a generator network G and a discriminator network D which compete with each other through a mutual loss computation (eq. 1). G tries to generate samples similar to the real sample distribution $\mathbb{P}_r$, while D attempts to distinguish between generated and real samples ($\mathbb{P}_G$ denotes the output distribution of G).

$$\min_G \max_D \mathbb{E}_{x\sim\mathbb{P}_r}\left[\log(D(x))\right] + \mathbb{E}_{\tilde{x}\sim\mathbb{P}_G}\left[\log(1-D(\tilde{x}))\right] \tag{1}$$

Professor Forcing: Traditional RNNs use teacher forcing - they are trained by supplying a known sequence of values as inputs during training. However, in unsupervised domains, where the correct input at each time step is not known in advance, the last step prediction is the input to the RNN - which causes the network to be less stable since errors are accumulated over time. Professor forcing [10] addresses this problem by using the GAN framework to train a discriminator to discern between two classes of an RNN's hidden states. The first, is the hidden states for teacher-forced inputs and the second class is the hidden states when feeding the last step's output to the RNN.

Wasserstein Loss: Traditional GANs have a sparse learning signal, especially when there is no overlap between the generated distribution $\mathbb{P}_G$ and the true data distribution $\mathbb{P}_r$. To remedy the situation, WGAN [1] suggests an alternative loss function for the discriminator (eq. 2), under the condition that D has to be Lipschitz (enforced by clipping D's weights ω). This alternative formulation improves learning stability and enables better debugging and parameter settings by providing meaningful learning curves.

$$\max_{\omega\in W} \mathbb{E}_{x\sim\mathbb{P}_r}\left[D(x;\omega)\right] - \mathbb{E}_{\tilde{x}\sim\mathbb{P}_G}\left[D(G(\tilde{x});\omega)\right] \tag{2}$$

3 Model

Let $s \in S, t \in T$ be sentences in the source and target domains respectively. We transfer the content of s to a sentence following the style of domain T, and apply the GAN framework for this purpose: an adversary D tries to distinguish between original sentences in T and ones transferred from S to T by a generator G. As mentioned before, unlike supervised tasks such as translation, the transferred result is not known in advance, but the generator learns to generate them.

The generator G is comprised of an encoder-decoder pair [4]; The encoder e takes as input a sentence of domains either S or T, and outputs a single context vector z. The decoder d unrolls z into a sentence in T (regardless of its origin domain). In other words, our generator is $G = d \circ e$.

Next, we address the issue of training GANs with discrete sequences: in our model selection of the next-word (which is a discrete action) is replaced by predicting an embedding for the next word (which is continuous). Since we work in the embedding space, we can use differentiable losses (see below) and not rely on methods such as policy gradient (that are less stable than training smooth objectives). Put differently, the input to our model is a sentence translated to sequences of embedding vectors (one embedding per word) that is fed to the encoder e. Unlike other models, the output of

our generator (more specifically - d) is an vector representation of the next word. Since that vector is highly unlikely to represent a word from the corpus, we find the vector embedding of minimal distance from the output vector, and the prediction is of the word associated with it.

In order to encourage the encoder-decoder to generate meaningful sentences we apply a reconstruction loss between sentences $t \in T$ and the reconstructed $\hat{t} = d(e(t))$. Let λ be a margin parameter, k a parameter to control the number of random words, w_i the embedding of word i in t, $\hat{w}_i$ the word embedding at index i of $\hat{t}$, and r_{ij} the embedding of a random word in the vocabulary. The reconstruction loss for t is shown in eq. 3. The goal of the reconstruction is to make $\hat{w}_i$ closer to w_i than the random word embeddings r_{ij} by a margin of λ. Other models that directly predict vocabulary words usually use negative log-likelihood for this purpose. Another benefit of using embeddings as outputs is that even when the model predicts the wrong word, usually the prediction is still in close proximity to the original word, resulting in a sentence that makes sense. In models that directly predict the next word this relation needs to be learned (e.g when probability of the word "is" is high, also set the probability for the word "was" to high).

$$L_{reco}(t, \hat{t}) = \sum_{w_i \in t} \frac{1}{k} \sum_{j \in [k]} \max\left(0, \lambda + |w_i - \hat{w}_i|_2^2 - |r_{ij} - \hat{w}_i|_2^2\right) \tag{3}$$

As for the adversary D, we tested the following architectures (also depicted in figures 3, 4 and 5). We ended up using the latter one for being the most stable of the tree:

1. **Content vector adversary** that takes as input content vectors (z) and is comprised of fully connected layers.
2. **Embedding adversary** that takes as input the sequence of generated embeddings and processes them using an RNN, then applies the fully connected layers on the last state of the RNN. Here we used Professor Forcing [10] - i.e. for t the decoder is teacher forced (takes as inputs the true embeddings) whereas for s the decoder is fed its last prediction.
3. **Combined adversary** takes both the content vector and the sequence of generated embeddings. It then processes them using combination of both architectures as shown in figure 5.

For the discriminator loss we started with the original GAN loss (eq. 1), but found that Wasserstein loss (eq. 2) gives a stronger learning signal for the discriminator (for all three architectures). Denoting B_s and B_t the source and target sentences in the batch respectively, the discriminator loss L_D is then defined as:

$$L_D(B_s, B_t) = \frac{1}{\|B_s\|} \sum_{s \in B_s} D(G(s), e(s)) - \frac{1}{\|B_t\|} \sum_{t \in B_t} D(G(t), e(t)) \tag{4}$$

Note, that in contrast to the traditional GAN formulation, when using Professor Forcing, D is focused on aligning the generator G outputs between the source examples and the target examples. This is why it appears on both terms of L_D. Also, since we are using the WGAN, the weights of D have to be clipped.

In the generator training step, G has to succeed in two objectives - disrupt D and successfully reconstruct B_t. Let $\alpha > 0$ be a hyper parameter that sets the importance of the latter objective compared to the former one. Since G should interact with D only if D is a good predictor, we consider L_D only if D has high accuracy. Let $Acc_D(B_s, B_t)$ be the prediction accuracy of D, and ϵ be a margin between D and random. The loss in the generator step L_G is:

$$L_G(B_s, B_t) = -\mathbb{I}_{Acc_D(B_s, B_t) \geq 0.5 + \epsilon} \cdot L_D(B_s, B_t) + \alpha \sum_{t \in B_t} L_{reco}(t, G(t)) \tag{5}$$

See figure 1 for a complete depiction of the system.

Training process - We managed to improve the system stability by applying a few modifications:

1. **Pre-trained word embeddings:** the following word vector configurations were used: trainable and non-trainable vectors with random initialization, non-trainable general-purpose GloVe vectors and skip-gram vectors trained specifically for our dataset. The reported results use the skip-gram vectors configuration since it produced the best results.

2. **Pre-training G:** the training process starts by only training G on L_{reco} (forcing $L_D = 0$). This allows the game between D an G to start when G already retains some information about correct reconstruction instead of a complete random behavior.
3. **Training the GAN:** continue training both G and D according to the losses described in eq. 4, 5.

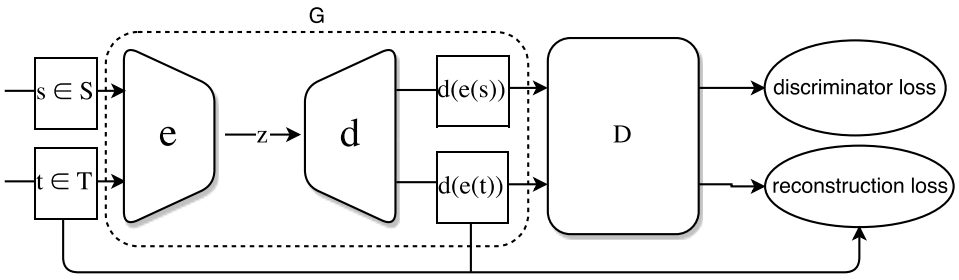

Figure 1: Main components of the network, see section 3 for complete description

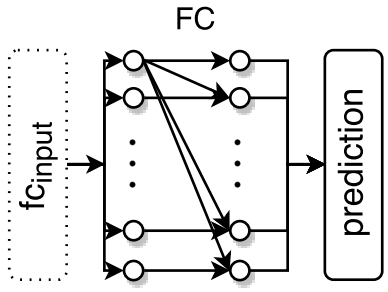

Figure 2: Descriminator architecture: inputs (depending on the discriminator type) are fed to a sequence of fully connected layer followed by a single value prediction.

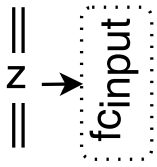

Figure 3: The content vector z as an input

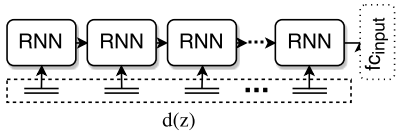

Figure 4: Input built of the RNN receiving $d(z)$ by (1) teacher forcing for the target domain and (2) by previous timestep prediction for the source domain

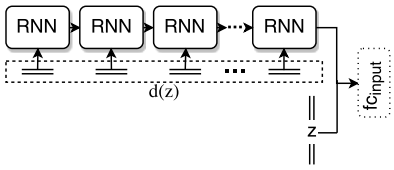

Figure 5: Combined input made of z and $d(z)$

4 Experiment Description

The corpus we used was **Yelp's dataset** [16], containing positive and negative local businesses reviews followed by a 1-5 stars ranking. It is fairly large and suitable for transfer of sentiment, and was used previously for style transfer [14] which allows us to compare results.

We further modified the dataset to ease later phases by making it more distinct: reviews with a ranking of 3 stars (neutral sentiment) were removed. Each review was then split into sentences using the NLTK sentence tokenizer [11], labeling each sentence as positive or negative based on the review ranking. We then trained a sentiment analysis model based on the those sentences and kept only

reviews whose sentiment analysis predicted correctly with high certainty. We ended up with 1,039K positive sentences and 329K negative ones.

The corpus was then processed into vector embeddings of size 200. Rare words (of single appearance) were replaced with a special token, and we ended with 53,708 unique words. We trained the embeddings according to the skip-gram model [12], with noise-contrastive estimation loss and stochastic gradient descent.

When running the model we randomly chose 300K sentences from each of the corpora for our train set and 1K different sentences for the validation set.

In all our expirements the source was the negative sentiment corups and the target was the positive sentiment corpus, which mean that our goal was to transform the negative sentences to positive ones.

5 Main Results

Our model achieved an accuracy of 93.3% (sentences which our trained classifier classified as positive), outperforming the cross-aligned auto encoder [14] which got 78%. Note that our results were achieved by executing on 6 words sentences due to limitations of computational resources, while the cross-aligned model was execute on 15 words sentences. Using 15 words would probably decrease our system's accuracy, however this may be overcome by using curriculum learning. In addition, our classifier is 100% accurate on our test and train sets (see section 4) while [14] is only 85.4% accurate. Table 1 shows a few samples of sentiment transfer generated by the model, including both successful and less successful examples. The unsuccessful ones may be generally divided into grammar mistakes (rare), content mistakes, or general vagueness. Although most of the sentences were converted into the correct sentiment, some of them did not preserve the content of the original sentence, which was counted as failure. However we were not successful at coming up with a metric to quantify the content preservation.

For the experiments to converge, we had to use high α (e.g stronger generator loss then discriminator loss), otherwise the discriminator gained too much power and disrupted the results.

Successful attempts
will not return . will definitely go back .
I would give zero stars but I highly recommend this place .
worst pedi I have ever had best facial I have ever had
the fried rice was horrible . the fried rice was perfect .
Incorrect grammar
the macchiato i had today was the vanilla burger has been was
Unpreserved content
did n't want to honor my always a great experience for an
my 2nd order was denied my new favorite is fantastic on
Vague
completely screwed up my order giving super delish with my favorite found
do n't waste your time . great food , great food .

Table 1: Successful and unsuccessful examples: pairs of original sentence (negative domain), followed by the same sentence transferred (positive domain). We can see the advantage of our pre-trained embedding (e.g. pedi transformed to facial). We can also notice that even in the failed transformations, the sentences are overall positive.

6 Project Overview

In general we have followed execution plan. There are several modification we did to differentiate our work from [14] that has a similar motivation and a similar model (a paper that was published after we have started working on our project):

1. The model was changed from having a sequence of discrete actions to a model which outputs a continuous space prediction. An important distinction is that we are using the embedding space instead of next word soft-max (for advantages of using embedding see section 3).
2. We redesigned our evaluation process. Originally, we wanted to take Yodish (language of Yoda) and transfer it to English, and again with Shakespearean English to modern English. The intended evaluation metric would have been cross entropy reconstruction loss. However, this metric is better suited for translation and less suited for transfer of content. Since style transfer is less restrictive, we adopted the task of transferring negative Yelp reviews to positive ones. The evaluation metric here is fooling a pretrained sentiment analyzer to think that the sentiment of the transferred sentence is the same as of the target domain.

Along the way, we also stabilized the system by changing the adversarial loss to Wasserstein loss, and by pre-training the word vectors and the generator. There are other features that we examined that did not seem to work well:

1. Other word embedding architectures (described earlier 3)
2. Other discriminator architectures (described earlier 3)
3. Other RNN cell types, we tested both LSTM and GRU and found no concrete difference. We decided to stick to LSTM where we had more intuition about hyper-parameter tuning (due to a large number of tests).
4. Feeding the domain to the encoder both as a single fixed integer, or as a projected vector (as in [14]). We found that this caused the system to become unstable. We hypothesize that this is similar to why trainable word embeddings did not converge - the system became too complex and therefore harder to learn.
5. Addition of a loss term to L_G: the L_2 distance between $e(s)$ (encoded source) and $e(d(e(s)))$. The motivation here is to encourage the encoder to only keep content related information. We hypothesize that it was not effective because encouraging all the encoded vectors to look the same, e centered all the results around a single point regardless of the input. A solution to this issue is to use a margin-like loss, however we did not have time to test this resolution and its effectiveness.

7 Conclusion

In this work we proposed an embedding-based generative model to transfer sentences from one domain to another. Our model is trained using the GAN framework, however unlike most text generation methods in other projects, we managed to avoid using gradient approximations as a result of using an embedding vector as output. Our experiments show promising results, that make good use of the embedding space as well.
Our implementation may be accessed at `https://github.com/ashual/style-transfer`.

References

[1] Martın Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *CoRR* abs/1701.07875 (2017). URL: `http://arxiv.org/abs/1701.07875`.

[2] T. Che et al. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks". In: *ArXiv e-prints* (Feb. 2017). arXiv: `1702.07983 [cs.AI]`.

[3] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). URL: `http://arxiv.org/abs/1406.1078`.

[4] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). URL: http://arxiv.org/abs/1406.1078.

[5] R Devon Hjelm et al. “Boundary-Seeking Generative Adversarial Networks”. In: *ArXiv e-prints* (Feb. 2017). arXiv: 1702.08431 [stat.ML].

[6] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to Forget: Continual Prediction with LSTM”. In: *Neural Computation* 12 (1999), pp. 2451–2471.

[7] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *CoRR* abs/1406.2661 (2014). URL: http://arxiv.org/abs/1406.2661.

[8] Sepp Hochreiter and JÃŒrgen Schmidhuber. “Long Short-term Memory”. In: 9 (Dec. 1997), pp. 1735–80.

[9] T. Kim et al. “Learning to Discover Cross-Domain Relations with Generative Adversarial Networks”. In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.05192 [cs.CV].

[10] Alex Lamb et al. “Professor Forcing: A New Algorithm for Training Recurrent Networks”. In: *CoRR* abs/1610.09038 (2016). URL: http://arxiv.org/abs/1610.09038.

[11] Edward Loper and Steven Bird. “NLTK: The Natural Language Toolkit”. In: *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*. ETMTNLP ’02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 63–70. DOI: 10.3115/1118108.1118117. URL: https://doi.org/10.3115/1118108.1118117.

[12] T. Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *ArXiv e-prints* (Oct. 2013). arXiv: 1310.4546 [cs.CL].

[13] M. Mirza and S. Osindero. “Conditional Generative Adversarial Nets”. In: *ArXiv e-prints* (Nov. 2014). arXiv: 1411.1784 [cs.LG].

[14] Tianxiao Shen et al. “Style Transfer from Non-Parallel Text by Cross-Alignment”. In: *CoRR* abs/1705.09655 (2017). URL: http://arxiv.org/abs/1705.09655.

[15] Yaniv Taigman, Adam Polyak, and Lior Wolf. “Unsupervised Cross-Domain Image Generation”. In: *CoRR* abs/1611.02200 (2016). URL: http://arxiv.org/abs/1611.02200.

[16] Yelp. *Yelp dataset challenge*. URL: https://www.yelp.com/dataset/challenge.

[17] L. Yu et al. “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient”. In: *ArXiv e-prints* (Sept. 2016). arXiv: 1609.05473 [cs.LG].