Notes on "Unsupervised Text Style Transfer using Language Models as Discriminators"

Xavier Garcia

June 19, 2019

The purpose of these notes is to explain the paper"Unsupervised Text Style Transfer using Language Models as Discriminators".

1 Theoretical Framework

For the purposes of these notes, we have a collections of texts $t_1, ..., t_N$, which are viewed being generated by a random variable T. We make the structural assumption that there are two latent random variables which play a critical role in the distribution of T: A **content** random variable Z, which contains the factual information of the text, as well as a **style** random variable S, which represents the style of text e.g. whether it's positive or negative, written by a male or female, the age of the writer, etc. For these notes, we'll assume that S is a Bernouilli random variable, and that the dataset $t_1, ..., t_N$ decomposes into two datasets $x_1, ..., x_{N_0}$ and $y_1, ..., y_{N_1}$ which are generated by $X := T|(S = 0)$ and $Y := T|(S = 1)$ respectively. We also assume that we know these labels i.e. we know whether each t_i has style 0 or 1.

We define the **style transfer task** to be the problem of understanding the conditional distribution $Y|X = x$. The first step towards tackling this problem is to use the latent content variable z:

$$p_{Y|X}(y|x) = \int p_{Z|X}(z|x)p_{Y|X,Z}(y|x, z)dz \tag{1}$$

where for a random variable V, p_V denotes the density. A priori this may not even make sense, much less exist, but we will ignore these concerns since further assumptions will solve all these problems. To further simplify this expression, this paper makes the assumption of **disentanglement**: Z and S are independent of each other. This assumption yields the following equality:

$$p_{Y|X,Z}(y|x, z) = p_{Y|Z}(y|z)$$

which allows us to write equation (1) as follows:

$$p_{Y|X}(y|x) = \int p_{Z|X}(z|x)p_{Y|Z}(y|z)dz \tag{2}$$

We can also write out the densities in this expression in terms of T and S directly, as is done in the paper:

$$p_{Y|Z}(y|z) = p_{T|Z,S}(y|z,1)$$
$$p_{Z|X}(x|z) = p_{Z|T,S}(z|x,0)$$

We can interpret equation (2) as detailing the following generative process: Sample a content vector z from the posterior distribution of $p_{Z|X}(z|x)$ then sample a new text y from the posterior distribution of $p_{Y|Z}(y|z)$. More informally, we can think of the posterior $p_{Z|X}(z|x)$ as extracting the latent code z from x, and the posterior $p_{Y|X}(y|z)$ as decoding the latent code to generate y, under the assumption that the generated text should have the style $S = 1$.

This suggests that we should use the encoder-decoder structure. The encoder E will take a text t and style s and produce a content vector $z := E(t, s)$ while the decoder G will take the latent vector z and a style s' and produce a new text $t' = G(z, s')$. In particular, we can think of decoder $G(z, s')$ as the distribution of $T|Z = z, S = s'$.

2 Loss functions

From the last section, we've reduced the problem to training the encoder-decoder pair (E, G). For concreteness, we will denote the parameters of the pair by θ_E and θ_G respectively. Given the lack of parallel data, it's not clear what the training objective should be. The first obvious candidate is to use the encoder-decoder as an auto-encoder.

Definition 1. *Suppose we have some dataset $t_1, ..., t_N$ with style labels $s_1, ..., s_N$. Suppose that we also have some encoder and decoder parameters θ_E and θ_G respectively, with the notation $z_i = E_{\theta_E}(t_i, s_i)$ and $p_{\theta_G}(t|z, s) = p_{T|Z,S}(t|z, s, \theta_G)$. We define the* ***reconstruction loss*** *$\mathcal{L}_{rec}$ by the following formnula:*

$$\mathcal{L}_{rec}(\theta_E, \theta_G) = \sum_{i=1}^{N} -\log p_{\theta_G}(t_i|z_i, s_i)$$

The problem with this loss is that it doesn't ensure that the Z and S remain independent. In the ideal case, we should expect that the random variables $E(X, 0)$ and $E(Y, 1)$ should have the same distribution, since they correspond to $Z|S = 0$ and $Z|S = 1$ respectively. To verify this, we could use an adversarial training method by using an auxilliary discriminator D to help distinguish between the two. In theory, if the hidden representations are similar, then this should force the desired disentanglement.

However, our real goal is not disentanglement, but high quality generation. Mathematically speaking, our goal is not to force $E(X, 0)$ and $E(Y, 1)$ to have the same distribution. Instead, our goal should be to have Y and $G(E(X, 0), 1)$ share the distribution, similarly for X and $G(E(Y, 1), 0)$. In English, this says that if we sample a sentence from X, recover its content $E(X, 0)$ then generate

a sentence with the style of Y, then it should be similar to sampling a sentence directly from Y.

One could do this by using professor forcing on the hidden states of the generator G. Instead, the paper proposes to use language models to approximate the probability mass functions p_X and p_Y of X and Y directly, which we can use to evaluate the quality of our samples. We pretrain these language models directly and assume without loss of generality that we have the exact mass functions and hence fix these parameters. We'll find it useful to define the **transfer distributions**:

$$\tilde{X} = G_X(Y) := G(E(Y, 1), 0)$$
$$\tilde{Y} = G_Y(X) := G(E(X, 0), 1)$$

This allows us to introduce the cross-alignment loss.

Definition 2. *Let p_X and p_Y denote the probability mass functions for the random variables X and Y. We define the **cross-alignment loss** or language model loss $\mathcal{L}_{\text{LM}}$ by the following formula:*

$$\mathcal{L}_{\text{LM}}(\theta_E, \theta_G) = \mathbb{E}_{\tilde{y}\sim\tilde{Y}}[-\log p_Y(\tilde{y})] + \mathbb{E}_{\tilde{x}\sim\tilde{X}}[-\log p_X(\tilde{x})] \tag{3}$$

While the loss $\mathcal{L}_{\text{LM}}$ seems reasonable, it possess one nasty feature: it's non differentiable. We can obviate this problem by using REINFORCE to compute the gradients i.e.

$$\nabla_{\theta_G}\mathcal{L}_{\text{LM}} = -\mathbb{E}_{\tilde{y}\sim\tilde{Y}}[\log p_Y(\tilde{y})\nabla_{\theta_G}\log p_{\tilde{Y}}(\tilde{y})] - \mathbb{E}_{\tilde{x}\sim\tilde{X}}[\log p_X(\tilde{x})\nabla_{\theta_G}\log p_{\tilde{X}}(\tilde{x})]$$

which can then be estimated by Monte Carlo methods. While unbiased, the estimator ends up being quite noisy, exhibiting high variance. To see why, we will only focus on the first term of equation (3), since manipulations on the second term are analogous. We can identify the categorical random variable $\tilde{Y}$ with its one-hot encoded representation i.e. if we denote the tth value of $\tilde{Y}$ by $\tilde{Y}_t$, then

$$\tilde{Y}_t = k \leftrightarrow \tilde{Y}_t = e_k$$

where e_k is the vector which is 1 at the kth index and 0 elsewhere. Let $\mathcal{T}$ be the random variable which denotes the length of $\tilde{Y}$. We also introduce the notation

$$p_{Y_t}(\tilde{y}) := p_Y(\cdot|\tilde{y}_1, ..., \tilde{y}_{t-1})$$

i.e. $p_{\tilde{Y}_t}$ is the conditional distribution of the tth input of $\tilde{Y}$. We can then rewrite the cross-align loss as follows:

$$\mathcal{L}_{\text{LM}}(\theta_E, \theta_G) = \mathbb{E}_{\tilde{y}\sim\tilde{Y}}\left[-\sum_{t=1}^{\mathcal{T}} \tilde{y}_t \cdot \log p_{Y_t}(\tilde{y})\right] \tag{4}$$

where the $\cdot$ denotes inner-product. This formula reveals the reason for the high variance: one-hot encoding makes it so that two different samples of $\tilde{Y}$ could lead to widely different log probabilities. To reduce the variance, we need to soften the one-hot encoding into something slight more flexible.

3 Gumbel-Softmax

In this section, we present the Gumbel-Softmax distribution as a relaxation of the distribution induced by the one-hot encoding representation. To introduce this concept, we will need two auxilliary lemmas:

Lemma 3.1. *Let $\lambda_1, ..., \lambda_n$ denote some positive parameters, and suppose that we had i.i.d. exponential random variables $W_i \sim \text{Exp}(\lambda_i)$. Then, the minimum of these random variables is still an exponential random variable:*

$$\min_{1 \leq i \leq n} W_i \sim \text{Exp}\left(\sum_{i=1}^{n} \lambda_i\right)$$

Proof. The result will follow from the fact that the minimum $\min_{1 \leq i \leq n} W_i$ has the same tails as $\text{Exp}\left(\sum_{i=1}^{n} \lambda_i\right)$. We prove this explicitly in the following equations:

$$\begin{aligned}
\mathbb{P}\left(\min_{1 \leq i \leq n} W_i \geq x\right) &= \prod_{i=1}^{n} \mathbb{P}(W_i \geq x) \\
&= \prod_{i=1}^{n} \exp\left(-\lambda_i x\right) \\
&= \exp\left(-\sum_{i=1}^{n} \lambda_i x\right) \\
&= \mathbb{P}\left(\text{Exp}\left(\sum_{i=1}^{n} \lambda_i\right) \geq x\right)
\end{aligned}$$

□

Lemma 3.2. *Retain the assumptions from the previous lemma. Then, we can compute the distribution of* $\text{argmin}_{1 \leq i \leq n} W_i$*:*

$$\mathbb{P}\left(\text{argmin}_{1 \leq i \leq n} W_i = j\right) = \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i}$$

Proof. We prove the equality directly by invoking the previous lemma as follows:

$$\begin{aligned}
\mathbb{P}\left(\text{argmin}_{1 \leq i \leq n} W_i = j\right) &= \mathbb{P}\left(\min_{1 \leq i \leq n, i \neq j} W_i \geq W_j\right) \\
&= \lambda_j \int_0^\infty \mathbb{P}\left(\min_{1 \leq i \leq n, i \neq j} W_i \geq x\right) e^{-\lambda_j x} dx \\
&= \lambda_j \int_0^\infty e^{-\sum_{i \neq j} \lambda_i x} e^{-\lambda_j x} dx \\
&= \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i}
\end{aligned}$$

□

We can use these two lemmas to prove a version of the reparametrization trick for discrete random variables:

Lemma 3.3. *Suppose V is a discrete random variable with K different values $v_1, ..., v_k$ and*

$$\mathbb{P}(V = v_k) \propto p_k$$

for some known values p_k. Suppose $U_1, ..., U_K$ are i.i.d. uniformly distributed random variables valued in $[0, 1]$. Then we have the following result:

$$V \sim \text{argmax}_{1 \leq k \leq K} \log p_k - \log(-\log U_k). \tag{5}$$

Proof. The key idea is that $-\log U_k \sim \text{Exp}(1)$ which forces the following computation:

$$\begin{aligned} \log p_k - \log(-\log U_k) &= \log p_k - \log \text{Exp}(1) \\ &= -\log\left(\frac{1}{p_k}\text{Exp}(1)\right) \\ &= -\log \text{Exp}(p_k) \end{aligned}$$

We can take argmax to both sides, and replace the argmax on the right with an argmin by removing the negative sign in front of the log. We then apply Lemma 3.2 to furnish the result. □

Notice that the we've separated the parameters on the right hand side of equation (5) from the distribution, which allows us to use this for various generative models, such as variational autoencoders. Traditionally, the random variable $\mathcal{G} = -\log(\log U)$ is called the **Gumbel** distribution.

We can identify the categorical random variable V with its one-hot encoded representative, where the only non-zero is the one corresponding to the argmax. We can then relax this argmax by replacing it with a softmax. In particular, if $l = (l_1, ..., l_K)$ are the logits of the categorical distribution V, then we can approximate V with an approximation p, given by

$$p = \text{Gumbel-Softmax}(l, \tau) := \text{softmax}\left(\frac{l + g}{\tau}\right)$$

where $g = (g_1, ..., g_K)$ consists of K i.i.d. Gumbel distributions, and $\tau > 0$ is a temperature parameter. As $\tau \to 0$, the distribution converges to the one given by the argmax. As $\tau \to \infty$, the distribution converges to the uniform distribution, completely independent of the logits.

Returning to our original setting, we can use the Gumbel-Softmax during training. We describe the generative process:

1. Start with $\langle \texttt{SOS} \rangle$ token and content vector z.
2. Pass the token and content vector through the generator and through the language model for Y to produce some logits $\tilde{l}_1$ and $\hat{l}_1$, as well as probabilities $\tilde{p}_1 = \text{Gumbel-Softmax}(l_1, \tau)$ and $\hat{p}_1 = \text{Gumbel-Softmax}(l_1, \tau)$.

3. Use $\tilde{p}_1$ as weights to compute soft prediction of next word: $\tilde{y}_1(\tau) = \sum_{i=1}^{|\mathcal{V}|} w_i(\tilde{p}_1)_i$ where $|\mathcal{V}|$ is the size of the vocabulary and w_i refer to the embeddings of the words.

4. Use $\tilde{y}_1(\tau)$ as the next word for both the language model and prediction and continue.

At the end of this process, we'll end up with two sequence of probability vectors: the one predicted by the generator $\tilde{p}_1, ..., \tilde{p}_{\mathcal{T}}$ and the ones predicted by the language model $\hat{p}_1, ..., \hat{p}_{\mathcal{T}}$. These probability vectors are completely determined by $\tilde{y}_1, ..., \tilde{y}_{\mathcal{T}}$, which we think of as a sample of a single random variable $\tilde{Y}(\tau)$. By definition, we set $\tilde{Y}(0) := \tilde{Y}$. This notation is justified by considering the dynamics of these equations as $\tau \to 0$:

$$\begin{aligned} \lim_{\tau \to 0} \tilde{p}_t &= \tilde{y}_t \\ \lim_{\tau \to 0} \hat{p}_t &= p_{Y_t}(\tilde{y}) \\ \lim_{\tau \to 0} \tilde{Y}(\tau) &= w_{\tilde{Y}} \end{aligned}$$

These limits imply the following limit:

$$\lim_{\tau \to 0} \tilde{p}_t \log \hat{p}_t = \tilde{y}_t \cdot \log p_{Y_t}(\tilde{y})$$

Finally, we recall that $\tilde{Y}$ can be reparametrized as function of X, which is useful since X does not depend on the parameters. In particular, we could write $\tilde{y} := \tilde{y}(x)$ and $\tilde{p} := \tilde{p}(x)$.

These limits suggest that we should approximate the cross-align loss $\mathcal{L}_{\text{LM}}$ in equation (4) as follows:

$$\begin{aligned} \mathcal{L}_{\text{LM}}(\theta_E, \theta_G) &= \mathbb{E}_{\tilde{y} \sim \tilde{Y}} \left[-\sum_{t=1}^{\mathcal{T}} \tilde{y}_t \cdot \log p_{Y_t}(\tilde{y}) \right] \\ &= \mathbb{E}_{x \sim X} \left[-\sum_{t=1}^{\mathcal{T}} \tilde{y}(x)_t \cdot \log p_{Y_t}(\tilde{y}(x)) \right] \\ &\approx \mathbb{E}_{x \sim X} \left[-\sum_{t=1}^{\mathcal{T}} \tilde{p}_t(x) \cdot \log \hat{p}_t(x) \right] \end{aligned}$$

The final objective on the right is nice for two reasons: 1) the distribution for which we are expectating against does not depend on the parameters and 2) the functions inside the expectation are smooth with respect to the generator's parameters. This allows us to differentiate the right hand side, and move the gradient inside the expectationg, yielding the following equation:

$$\nabla_{\theta_G} \mathcal{L}_{\text{LM}} \approx \mathbb{E}_{x \sim X} \left[-\nabla_{\theta_G} \sum_{t=1}^{T} \tilde{p}_t(x) \cdot \log \hat{p}_t(x) \right]$$