Plug and Play Autoencoders for Conditional Text Generation

Florian Mai[†♠] **Nikolaos Pappas**[♣] **Ivan Montero**[♣]
Noah A. Smith[♣◇] **James Henderson**[†]

[†]Idiap Research Institute, Martigny, Switzerland
[♠]EPFL, Lausanne, Switzerland
[♣]University of Washington, Seattle, WA, USA
[◇]Allen Institute for Artificial Intelligence, Seattle, WA, USA
{florian.mai,james.henderson@idiap.ch}
{npappas,ivamon,nasmith@cs.washington.edu}

Abstract

Text autoencoders are commonly used for conditional generation tasks such as style transfer. We propose methods which are plug and play, where any pretrained autoencoder can be used, and only require learning a mapping within the autoencoder's embedding space, training embedding-to-embedding (*Emb2Emb*). This reduces the need for labeled training data for the task and makes the training procedure more efficient. Crucial to the success of this method is a loss term for keeping the mapped embedding on the manifold of the autoencoder and a mapping which is trained to navigate the manifold by learning offset vectors. Evaluations on style transfer tasks both with and without sequence-to-sequence supervision show that our method performs better than or comparable to strong baselines while being up to four times faster.

1 Introduction

Conditional text generation[1] encompasses a large number of natural language processing tasks such as text simplification (Nisioi et al., 2017; Zhang and Lapata, 2017), summarization (Rush et al., 2015; Nallapati et al., 2016), machine translation (Bahdanau et al., 2015; Kumar and Tsvetkov, 2019) and style transfer (Shen et al., 2017; Fu et al., 2018). When training data is available, the state of the art includes encoder-decoder models with an attention mechanism (Bahdanau et al., 2015; Vaswani et al., 2017) which are both extensions of the original sequence-to-sequence framework with a fixed bottleneck introduced by Sutskever et al. (2014). Despite their success, these models are costly to train and require a large amount of parallel data.

Yet parallel data is scarce for conditional text generation problems, necessitating unsupervised solutions. Text autoencoders (Bowman et al., 2016) have proven useful for a particular subclass of unsupervised problems that can be broadly defined as style transfer, i.e., changing the style of a text in such a way that the content of the input is preserved. Examples include sentiment transfer (Shen et al., 2017), sentence compression (Fevry and Phang, 2018), and neural machine translation (Artetxe et al., 2018). Most existing methods specialize autoencoders to the task by conditioning the decoder on the style attribute of interest (Lample et al., 2019; Logeswaran et al., 2018), assuming the presence of labels during training of the autoencoder. The main drawback of this approach is that it cannot leverage pretraining on **unlabeled data**, which is probably the most important factor for widespread progress in supervised NLP models in recent years in text analysis (Peters et al., 2018; Radford et al., 2019; Devlin et al., 2019) and generation tasks (Song et al., 2019; Variš and Bojar, 2019). There are no style transfer methods, to the best of our knowledge, that were designed to leverage autoencoder pretraining, and only few can be used in this way (Shen et al., 2020; Wang et al.,

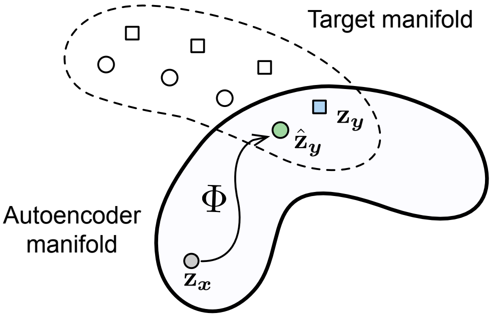

Figure 1: The *manifold* of a text autoencoder is the low-dimensional region of the high-dimensional embedding space where texts are actually embedded. The example shows the mapping of a source sequence $\boldsymbol{x}$ with embedding $\mathbf{z}_{\boldsymbol{x}}$ to $\mathbf{z}_{\boldsymbol{y}}$, which is the embedding of target sequence $\boldsymbol{y}$ such that it reflects the target manifold.

[1] We use this term to refer to text generation conditioned on *textual* input.

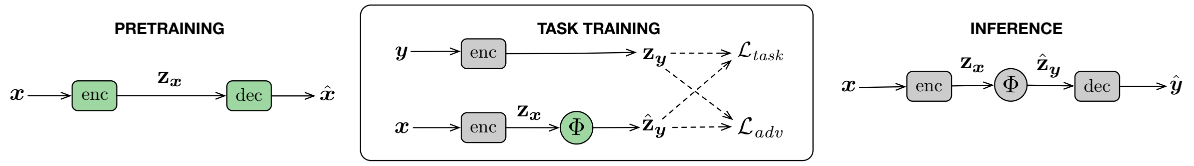

Figure 2: High-level view of the supervised variant of our framework *Emb2Emb*. *Left*: we pretrain an autoencoder on (unannotated) text, which transforms an input sentence $\boldsymbol{x}$ into an embedding $\mathbf{z}_{\boldsymbol{x}}$ and uses it to predict a reconstruction $\hat{\boldsymbol{x}}$ of the input sentence. *Center*: using the (frozen, hence depicted in gray) autoencoder, we learn a mapping Φ (trained, hence depicted in green) from the autoencoder's embedding of an input $\mathbf{z}_{\boldsymbol{x}}$ to the embedding $\mathbf{z}_{\boldsymbol{y}}$ of the output sentence $\boldsymbol{y}$. The training objective consists of two losses: $\mathcal{L}_{task}$ enforces the predicted output embedding to be close to the true output embedding, and $\mathcal{L}_{adv}$ is an adversarial loss term that enforces the output embedding to be on the manifold of the autoencoder. *Right*: at inference time, Φ is composed between the autoencoder's encoder and decoder to transform input sentence $\boldsymbol{x}$ to output sentence $\hat{\boldsymbol{y}}$. Not shown: the unsupervised variant where only $\boldsymbol{x}$ (not $\boldsymbol{y}$) sequences are available in task training (Section 9).

2019).

In this paper, we propose an autoencoder-based framework that is *plug and play*,[2] meaning it can be used with any pretrained autoencoder, and thus can benefit from pretraining. Instead of learning conditional text generation in the discrete, high-dimensional space where texts are actually located, our method, called **Emb2Emb**, does all learning in the low-dimensional continuous embedding space, on the *manifold* of a pretrained text autoencoder (see Figure 1). The result of learning is simply a mapping from input embedding to output embedding. Two crucial model choices enable effective learning of this mapping. First, an adversarial loss term encourages the output of the mapping to remain on the manifold of the autoencoder, to ensure effective generation with its decoder. Second, our neural mapping architecture is designed to learn offset vectors that are added to the input embedding, enabling the model to make small adjustments to the input to solve the task. Lastly, we propose two conditional generation models based on our framework, one for supervised style transfer (Section 2.3) and the other for unsupervised style transfer (Section 2.4) that implement the criteria of content preservation and attribute transfer directly on the autoencoder manifold.

We evaluate on two style transfer tasks for English. On text simplification (Section 3.1), where supervision is available, we find that our approach outperforms conventional end-to-end training of models with a fixed-size "bottleneck" embedding (like an autoencoder) while being about four times faster. On unsupervised sentiment transfer (Section 3.2), where no parallel sentence pairs are available to supervise learning, and where models with a fixed-size bottleneck are a common choice, Emb2Emb preserves the content of the input sentence better than a state-of-the-art method while achieving comparable transfer performance. Experimentally, we find that our method, due to being plug and play, achieves performances close to the full model when only 10% of the labeled examples are used, demonstrating the importance of pretraining for this task.

Our contributions can be summarized as follows:

- Our proposed framework Emb2Emb reduces conditional text generation tasks to mapping between continuous vectors in an autoencoder's embedding space.

- We propose a neural architecture and an adversarial loss term that facilitate learning this mapping.

- We evaluate two new conditional generation models for generation tasks with and without parallel examples as supervision.

- We demonstrate that our model benefits substantially from pretraining on large amounts of unlabeled data, reducing the need for large labeled corpora.

2 Proposed Framework

The key idea of our framework is to reduce discrete sequence-to-sequence tasks to a continuous embedding-to-embedding regression problem. Our Emb2Emb framework for conditional generation based on pretrained autoencoders (Figure 2) encompasses learning sequence-to-sequence tasks both

[2] This term is borrowed from studies on unconditional text generation with a specific attribute (Duan et al., 2020).

where parallel input/output sentence pairs are available ("supervised") and where they are not ("unsupervised"). Given a pretrained autoencoder (left of Figure 2, Section 2.1) we use its encoder to encode both input and, during supervised training, the output sequence (Section 2.3).[3] We then learn a continuous mapping Φ from input sequence embeddings to output sequence embeddings (center of Figure 2). In the unsupervised case (Section 2.4), the task loss reflects the objectives of the task, in our experiments consisting of two terms, one to encourage content preservation and the other to encourage style transfer. In both supervised and unsupervised cases, the task-specific loss $\mathcal{L}_{task}$ is combined with an adversarial term $\mathcal{L}_{adv}$ that encourages the output vector to stay on the autoencoder's manifold (see Figure 1; Section 2.5), so that the complete loss function is:

$$\mathcal{L} = \mathcal{L}_{task} + \lambda_{adv}\mathcal{L}_{adv}. \tag{1}$$

At inference time (right of Figure 2; Section 2.4), the decoder from the pretrained autoencoder's decoding function is composed with Φ and the encoder to generate a discrete output sentence $\hat{\boldsymbol{y}}$ conditioned on an input sentence $\boldsymbol{x}$:

$$\hat{\boldsymbol{y}} = (\mathrm{dec} \circ \Phi \circ \mathrm{enc})(\boldsymbol{x}) \tag{2}$$

2.1 Text Autoencoders

The starting point for our approach is an autoencoder $\mathcal{A} = \mathrm{dec} \circ \mathrm{enc}$ trained to map an input sentence to itself, i.e., $\mathcal{A}(\boldsymbol{x}) = \boldsymbol{x}$. Letting $\mathcal{X}$ denote the (discrete) space of text sequences, the encoder $\mathrm{enc} : \mathcal{X} \to \mathbb{R}^d$ produces an intermediate continuous vector representation (embedding), which is turned back into a sequence by the decoder $\mathrm{dec} : \mathbb{R}^d \to \mathcal{X}$. Note that an autoencoder can, in principle, be pretrained on a very large dataset, because it does not require any task-related supervision.

While our framework is compatible with any type of autoencoder, in practice, learning the mapping Φ will be easier if the embedding space is smooth. How to train smooth text autoencoders is subject to ongoing research (Bowman et al., 2016; Shen et al., 2020). In this work, we will focus on denoising recurrent neural network autoencoders (Vincent et al., 2010; Shen et al., 2020; see Appendix A). However, any advancement in this research direction will directly benefit our framework.

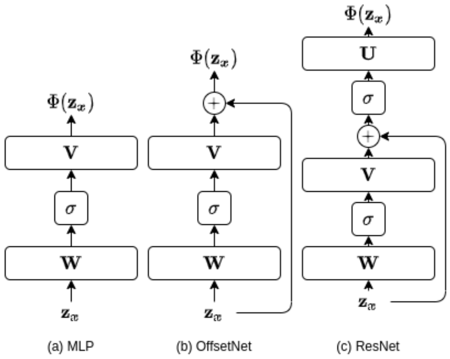

Figure 3: Illustration of the three neural architectures (1 layer) considered in this study. OffsetNet (b) differs from ResNet (c) in that there is no non-linear activation after the skip-connection (+), satisfying the notion of computing an offset vector that is added to the input.

2.2 Mapping Function Φ

A common choice for the mapping Φ to learn a regression task would be a k-layer MLP (Rumelhart et al., 1986), which transforms the input $\mathbf{z}_{\boldsymbol{x}} \in \mathbb{R}^d$ as:

$$\mathbf{y}^{(0)} = \mathbf{z}_{\boldsymbol{x}} \tag{3}$$

$$\forall j \in \{1, \ldots, k\}, \quad \mathbf{y}^{(j)} = \sigma(\mathbf{W}^{(j)}\mathbf{y}^{(j-1)}) \tag{4}$$

$$\Phi(\mathbf{z}_{\boldsymbol{x}}) = \mathbf{W}^{(k)}\mathbf{y}^{(k)}, \tag{5}$$

where $\mathbf{W}^{(j)} \in \mathbb{R}^{d\times d}$ are linear transformations and σ denotes a non-linear activation function. The linear transformation at the output layer allows Φ to match the unbounded range of the regression task. Note that we have suppressed the bias terms of the transformations for clarity. In past work (Shen et al., 2020), a mapping function was chosen with a specific form,

$$\phi(\mathbf{z}) = \mathbf{z} - \mathbf{v}_1 + \mathbf{v}_2, \tag{6}$$

where the "offset" vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ correspond to encodings of the input style and the output style, computed as the average of sentences with the respective style. Because dimensions not relating to style information cancel each other out, the output remains close to the input. However, this model lacks generality because the offset vectors are independent of the input.

We propose a mapping which incorporates the notion of an offset vector, but is conditioned on the input. Each layer of the Φ network moves through the embedding space by an input-specific offset, computed using a "skip" connection at each layer:

$$\mathbf{y}^{(j)} = \mathbf{y}^{(j-1)} + \underbrace{\mathbf{V}^{(j)}\sigma(\mathbf{W}^{(j)}\mathbf{y}^{(j-1)})}_{\text{offset } j}. \tag{7}$$

[3] Our code is available at: https://github.com/florianmai/emb2emb

$\mathbf{V}^{(j)}, \mathbf{W}^{(j)} \in \mathbb{R}^{d \times d}$ again denote linear transformations. Unlike the MLP, the skip connections bias the output to be close to the input. We refer to this architecture as **OffsetNet**.

Note that a residual network (He et al., 2016) corresponds to Equation 7 but with an additional non-linear transformation (typically of bounded range) applied to the output of Equation 7, again necessitating a linear transformation at the output layer. However, transforming the output all together would defeat the purpose of navigating the manifold with learned offsets. Figure 3 illustrates the differences. Our experiments (Section 3.1.2) validate the design choice for OffsetNet.

2.3 Supervised Task Loss $\mathcal{L}_{task}$

For supervised tasks, a collection of parallel sentence pairs $\langle(\boldsymbol{x}_i, \boldsymbol{y}_i)\rangle_{i=1}^{N}$ is available as supervision for the conditional generation task. After pretraining the autoencoder, enc is available to transform the training data into pairs of vectors $(\mathbf{z}_{\boldsymbol{x}_i}{=}\text{enc}(\boldsymbol{x}_i), \mathbf{z}_{\boldsymbol{y}_i}{=}\text{enc}(\boldsymbol{y}_i))$, giving us:

$$\mathcal{L}_{task} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{emb}(\Phi(\mathbf{z}_{\boldsymbol{x}_i}; \boldsymbol{\theta}), \mathbf{z}_{\boldsymbol{y}_i}). \quad (8)$$

The multivariate regression in Equation 8 requires that we specify a loss function, $\mathcal{L}_{emb}$, which should reflect semantic relatedness in the autoencoder's embedding space. For sentence and word embeddings, past work has concluded that cosine distance is preferable to Euclidean distance (i.e., mean squared error) in such settings (Xing et al., 2015; Bhat et al., 2019), which agreed with our preliminary experiments; hence, we adopt cosine distance for the task-specific loss $\mathcal{L}_{emb}$.

Kumar and Tsvetkov (2019) showed that another alternative, the Von Mises-Fisher loss, was preferable in learning to generate continuous word vector outputs. Their loss is not applicable in our setting, because the embedding space of an autoencoder is not unit-normalized like word vectors typically are. Therefore, we employ cosine loss and leave the exploration of other regression losses to future work.

2.4 Unsupervised Task Loss $\mathcal{L}_{task}$

In the unsupervised case, we do not have access to parallel sentence pairs $(\boldsymbol{x}, \boldsymbol{y})$. Instead, we have a collection of sentences labeled with their style attribute (e.g., sentiment), here denoted $\langle(\boldsymbol{x}_i, a_i)\rangle_{i=1}^{N}$. The goal of the task is twofold (Logeswaran et al., 2018): preserve the content of the input and match the desired value for the style attribute. We view this as a tradeoff, defining $\mathcal{L}_{task}$ as an interpolation between loss terms for each. With $\hat{\mathbf{z}}_{\boldsymbol{x}_i}{=}\Phi(\mathbf{z}_{\boldsymbol{x}_i}; \boldsymbol{\theta})$, for the unsupervised case we have:

$$\mathcal{L}_{task} = \lambda_{sty} \mathcal{L}_{sty}(\hat{\mathbf{z}}_{\boldsymbol{x}_i}) + (1 - \lambda_{sty}) \mathcal{L}_{cont}(\hat{\mathbf{z}}_{\boldsymbol{x}_i}, \mathbf{z}_{\boldsymbol{x}_i}). \quad (9)$$

where $\mathcal{L}_{cont}$ and $\mathcal{L}_{sty}$ are described in the following. Lastly, we describe an inference-time method that can improve the loss after applying the mapping even further.

Content preservation. We encourage the output to stay close to the input, on the assumption that embeddings are primarily about semantic content. To this end, we choose $\mathcal{L}_{cont}(\Phi(\mathbf{z}_{\boldsymbol{x}_i}; \boldsymbol{\theta}), \mathbf{z}_{\boldsymbol{x}_i})$ to be cosine distance.

Style. Following previous approaches (Engel et al., 2018; Liu et al., 2020; Wang et al., 2019), our style objective requires that we pretrain a (probabilistic) classifier that predicts the style attribute value from the (fixed) autoencoder's embedding. The classifier is then frozen (like the autoencoder) and our minimizing objective requires the output of our method to be classified as the target style. Formally, in a preliminary step, we train a style classifier $\mathbf{c} : \mathbb{R}^d \rightarrow \{0, 1\}$ on the embeddings of the autoencoder to predict one of the attributes (labeled as 0 and 1, respectively). We then freeze the classifier's parameters, and encourage Φ to produce outputs of the target attribute ($y{=}1$) via a negative log-likelihood loss:

$$\mathcal{L}_{sty}(\Phi(\mathbf{z}_{\boldsymbol{x}_i}; \boldsymbol{\theta}), \mathbf{z}_{\boldsymbol{x}_i}) = -\log(\mathbf{c}(\Phi(\mathbf{z}_{\boldsymbol{x}_i}; \boldsymbol{\theta}))).$$

Inference Time. The mapping Φ is trained to try to optimize the objective (1). In the unsupervised case, we can actually verify at test time whether it has succeeded, since nothing is known at training time that is not also known at test time (i.e., no labeled output). We propose a second stage of the mapping for the unsupervised case which corrects any suboptimality. We apply fast gradient iterative modification (FGIM; Wang et al., 2019) to improve the predicted embeddings further. Formally, we modify the predicted embedding $\hat{\mathbf{z}}_{\boldsymbol{x}_i}{=}\Phi(\mathbf{z}_{\boldsymbol{x}_i}; \boldsymbol{\theta})$ as

$$\hat{\hat{\mathbf{z}}}_{\boldsymbol{x}_i} = \hat{\mathbf{z}}_{\boldsymbol{x}_i} + \omega \nabla_{\hat{\mathbf{z}}_{\boldsymbol{x}_i}} \mathcal{L}(\hat{\mathbf{z}}_{\boldsymbol{x}_i}, \mathbf{z}_{\boldsymbol{x}_\mathbf{i}}),$$

where ω is the stepsize hyperparameter. This step is repeated for a fixed number of steps or until

$\mathbf{c}(\hat{\hat{\mathbf{z}}}_{\boldsymbol{x}_i}) > t$, where, $\mathbf{c}$ is the style classifier from above, and $t \in [0, 1]$ denotes some threshold.

Wang et al. (2019) use this method to modify the input embedding $\mathbf{z}_{\boldsymbol{x}_\mathbf{i}}$ by only following the gradient of the classifier $\mathbf{c}$, i.e., $\hat{\hat{\mathbf{z}}}_{\boldsymbol{x}_i} = \mathbf{z}_{\boldsymbol{x}_\mathbf{i}} + \omega \nabla_{\mathbf{z}_{\boldsymbol{x}_\mathbf{i}}} - \log \mathbf{c}(\mathbf{z}_{\boldsymbol{x}_\mathbf{i}})$. In contrast, our variant takes the entire loss term into account, including the adversarial term that encourages embeddings that lie on the manifold of the autoencoder, which we explain next.

2.5 Adversarial Loss $\mathcal{L}_{adv}$

Recall that at test time the output of Φ is the input to the pretrained decoding function dec. Even for supervised training, we do not expect to obtain zero loss during training (or to generalize perfectly out of sample), so there is a concern that the output of Φ will be quite different from the vectors dec was trained on (during pretraining). In other words, there is no guarantee that Φ will map onto the manifold of the autoencoder.

To address this issue, we propose an adversarial objective that encourages the output of Φ to remain on the manifold. Our method is similar to the "realism" constraint of Engel et al. (2018), who train a discriminator to distinguish between latent codes drawn from a prior distribution (e.g., a multivariate Gaussian) and the latent codes actually produced by the encoder. Instead of discriminating against a prior (whose existence we do not assume), we discriminate against the embeddings produced by Φ. We build on the adversarial learning framework of Goodfellow et al. (2014) to encourage the transformation Φ to generate output embeddings indistinguishable from the embeddings produced by the encoder enc.

Formally, let disc be a (probabilistic) binary classifier responsible for deciding whether a given embedding was generated by enc or Φ. The discriminator is trained to distinguish between embeddings produced by enc and embeddings produced by Φ:

$$\max_{\mathrm{disc}} \sum_{i=1}^{N} \log(\mathrm{disc}(\mathbf{z}_{\tilde{\boldsymbol{y}}_i})) + \log(\overline{\mathrm{disc}}(\Phi(\mathbf{z}_{\boldsymbol{x}_i})) \tag{10}$$

where $\mathrm{disc}(\mathbf{z})$ denotes the probability of vector $\mathbf{z}$ being produced by enc and $\overline{\mathrm{disc}}(\mathbf{z}) = 1 - \mathrm{disc}(\mathbf{z})$. The mapping Φ is trained to "fool" the discriminator:

$$\mathcal{L}_{adv}(\Phi(\mathbf{z}_{\boldsymbol{x}_i}); \boldsymbol{\theta}) = -\log(\mathrm{disc}(\Phi(\mathbf{z}_{\boldsymbol{x}_i}); \theta)) \tag{11}$$

Training the discriminator requires encoding negatively sampled sentences, $\mathbf{z}_{\tilde{\boldsymbol{y}}_i} = \mathrm{enc}(\tilde{y}_i)$, where we want these sentences to contrast with the output of the mapping $\Phi(\mathbf{z}_{\boldsymbol{x}_i})$. For the supervised case, we achieve this by taking the negative samples from the target sentences of the training data, $\tilde{y}_i = y_i$. In the unsupervised case, $\tilde{y}_i$ are sampled randomly from the data.

The mapping Φ is trained according to the objective in (1), in which $\mathcal{L}_{adv}$ depends on training the discriminator disc according to (10). In practice, we alternate between batch updates to Φ and disc. Our experiments in Section 3 will explore sensitivity to λ_{adv}, finding that it has a large effect. In practical applications, it should therefore be treated as a hyperparameter.

2.6 Summary

Our framework is *plug and play*, since it is usable with any pretrained autoencoder. Unlike previous methods by Shen et al. (2020) and Wang et al. (2019), which are specific to style transfer and do not learn a function (like Φ in Emb2Emb), ours can, in principle, be used to learn a mapping from any sort of input data to text, as long as the desired attributes of the generated text can be expressed as a loss function that is tied to the autoencoder manifold. In this study, we apply it to supervised and unsupervised text style transfer. The key component is the mapping function Φ, which is trained via a regression loss (plus auxiliary losses) to map from the embedding of the input sequence to the embedding of the output sequence. Learning the function is facilitated through the proposed **OffsetNet** and an adversarial loss term that forces the outputs of the mapping to stay on the manifold of the autoencoder.

3 Experiments

We conduct controlled experiments to measure the benefits of the various aspects of our approach. First, we consider a supervised sentence simplification task and compare our approach to others that use a fixed-size representational bottleneck, considering also the model's sensitivity to the strength of the adversarial loss and the use of OffsetNet (Section 3.1). We then turn to an unsupervised sentiment transfer task, first comparing our approach to other methods that can be considered "plug and play" and then investigating the effect of plug and play when only a little labeled data is available

(Section 3.2). We note that the current state of the art is based on transformers (Vaswani et al., 2017); since our aim is to develop a general-purpose framework and our computational budget is limited, we focus on controlled testing of components rather than achieving state-of-the-art performance.

Autoencoder. In all our experiments, we use a one-layer LSTM as encoder and decoder, respectively. We pretrain it on the text data of the target task as a denoising autoencoder (DAE; Vincent et al., 2010) with the noise function from Shen et al. (2020). Additional training and model details can be found in Appendix A.

3.1 Sentence Simplification

Sentence simplification provides a useful testbed for the supervised variant of our approach. The training data contains pairs of input and output sentences $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, where $\boldsymbol{x}_i$ denotes the input sentence in English and $\boldsymbol{y}_i$ denotes the output sentence in simple English. We evaluate on the English WikiLarge corpus introduced by Zhang and Lapata (2017), which consists of 296,402 training pairs, and development and test datasets adopted from Xu et al. (2016). Following convention, we report two scores: BLEU (Papineni et al., 2002), which correlates with grammaticality (Xu et al., 2016), and SARI (Xu et al., 2016), found to correlate well with human judgements of simplicity. We also compare training runtimes.

3.1.1 Comparison to Sequence-to-Sequence

Our first comparisons focus on models that, like ours, use a fixed-size encoding of the input. Keeping the autoencoder architecture fixed (i.e., the same as our model), we consider variants of the sequence-to-sequence model of Sutskever et al. (2014).[4] All of these models are trained "end-to-end," minimizing token-level cross-entropy loss. The variants are:

- **S2S-Scratch**: trains the model from scratch.

- **S2S-Pretrain**: uses a pretrained DAE and finetunes it.

Model	BLEU	SARI	Time
S2S-Scratch	3.6	15.6	3.7×
S2S-Pretrain	5.4	16.2	3.7×
S2S-MLP	10.5	17.7	3.7×
S2S-Freeze	23.3	22.4	2.2×
Emb2Emb	**34.7**	**25.4**	1.0×

Table 1: Text simplification performance of model variants of end2end training on the test set. "Time" is wall time of one training epoch, relative to our model, Emb2Emb.

- **S2S-MLP**: further adds the trainable mapping Φ used in our approach.

- **S2S-Freeze**: freezes the pretrained autoencoder parameters, which we expect may help with the vanishing gradient problem arising in the rather deep S2S-MLP variant.

For all the models, we tuned the learning rate hyperparameter in a comparable way and trained with the ADAM optimizer by Kingma and Ba (2015) (more details in the Appendix A.4).

Results. Table 1 shows test-set performance and the runtime of one training epoch relative to our model (Emb2Emb). First, note that the end-to-end models are considerably more time-consuming to train. S2S-Freeze is not only more than two times slower per epoch than Emb2Emb, but we find it to also require 14 epochs to converge (in terms of validation performance), compared to 9 for our model. Turning to accuracy, as expected, adding pretraining and the MLP to S2S-Scratch does improve its performance, but freezing the autoencoder (S2S-Freeze) has an outsized benefit. This observation may seem counter to the widely seen success of finetuning across other NLP scenarios, in particular with pretrained transformer models like BERT (Devlin et al., 2019). However, finetuning does not always lead to better performance. For instance, Peters et al. (2019) not only find the LSTM-based ELMo (Peters et al., 2018) difficult to configure for finetuning in the first place, but also observe performances that are often far lower than when just freezing the parameters. Hence, our results are not entirely unexpected. To further eliminate the possibility that the finetuned model underperformed merely because of improper training, we verified that the training loss of S2S-Pretrain is indeed lower than that of S2S-Freeze. Moreover,

[4] On this task, much stronger performance than any we report has been achieved using models without this constraint (Mathews et al., 2018; Zhang and Lapata, 2017). Our aim is not to demonstrate superiority to those methods; the fixed-size encoding constraint is of general interest because (i) it is assumed in other tasks such as unsupervised style transfer and (ii) it is computationally cheaper.

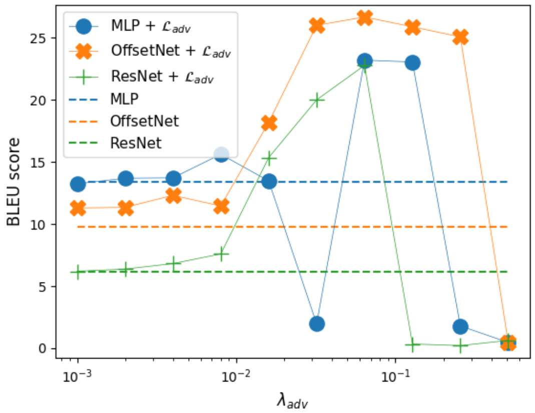

Figure 4: Performance on WikiLarge in terms of BLEU score on the development set (higher is better) by weight for the adversarial term λ_{adv}. Note that the x-axis is on a log scale.

the poor performance is also unlikely to be a problem of overfitting, because we mitigate this via early stopping. This suggests that the differences are largely due to the generalization abilities coming from the pretraining, which is partly forgotten when finetuning the entire model on the target task. Our results thus support the hypothesis of Mathews et al. (2018) that fixed-size bottlenecks and deeper networks make end-to-end learning harder. In contrast, training Emb2Emb even outperforms the best end-to-end model, S2S-Freeze.

From these results, we conclude that, when pretraining a fixed-size-representation autoencoder for plug and play text generation, learning text transformations entirely in continuous space may be easier and more efficient than using conventional sequence-to-sequence models.

3.1.2 Sensitivity Analysis

We next explore two of the novel aspects of our model, the adversarial loss and the use of OffsetNet in the mapping function Φ. We vary the tradeoff parameter λ_{adv} and consider variants of our approach using an MLP, ResNet, and OffsetNet at each value. All other hyperparameters are kept fixed to default values reported in Appendix A.

Results. Figure 4 plots the BLEU scores, with $\lambda_{adv} = 0$ as horizontal dashed lines. Each model's BLEU score benefits, in some λ_{adv} range, from the use of the adversarial loss. Gains are also seen for SARI (see Appendix A.4.3). OffsetNet is also consistently better than ResNet and, when using the adversarial loss, the MLP. From this we conclude that OffsetNet's approach of starting close to the input's embedding (and hence on/near the manifold), facilitates (adversarial) training compared to the MLP and ResNet, which, at the beginning of training, map to an arbitrary point in the embedding space due to the randomly initialized projection at the last layer.

3.2 Sentiment Transfer

We next evaluate our model on an unsupervised style transfer task. For this task, the training data is given pairs of input sentences and sentiment attributes $(\boldsymbol{x}_i, a_i)$, where $\boldsymbol{x}_i$ denotes the input sentence in English and a_i denotes its target sentiment, a binary value. For training, we use the Yelp dataset preprocessed following Shen et al. (2017). At inference time, we follow common evaluation practices in this task (Hu et al., 2017; Shen et al., 2017; Lample et al., 2019) and evaluate the model on its ability to "flip" the sentiment (measured as the accuracy of a DistilBERT classifier trained on the Yelp training set, achieving 97.8% on held-out data; Sanh et al., 2019),[5] and "self-BLEU," which computes the BLEU score between input and output to measure content preservation. There is typically a tradeoff between these two goals, so it is useful to visualize performance as a curve (accuracy at different self-BLEU values). We conduct three experiments. First, we compare to two other unsupervised sentiment transfer models that can be considered "plug and play" (Section 3.2.1). Second, we conduct controlled experiments with variants of our model to establish the effect of pretraining (Section 3.2.2). Third, we confirm the effectiveness of OffsetNet and the adversarial loss term for the sentiment transfer (Appendix A.5.4).

3.2.1 Comparison to Plug and Play Methods

To the best of our knowledge, there are only two other autoencoder-based methods that can be used in a plug and play fashion, i.e., training the autoencoder and sentiment transfer tasks in succession. These are the method of Shen et al. (2020), which is based on addition and subtraction of mean vectors of the respective attribute corpora (see Section 4), and FGIM (see Section 2.4); both of them are inference-time methods and do not learn a function (like Φ in Emb2Emb). Even though these methods are not specifically introduced with pretraining plug and play in mind, we can consider them in this way as alternatives to our model. Note that Wang

[5] Due to budget constraints, we evaluate only on transforming sentiment from negative to positive.

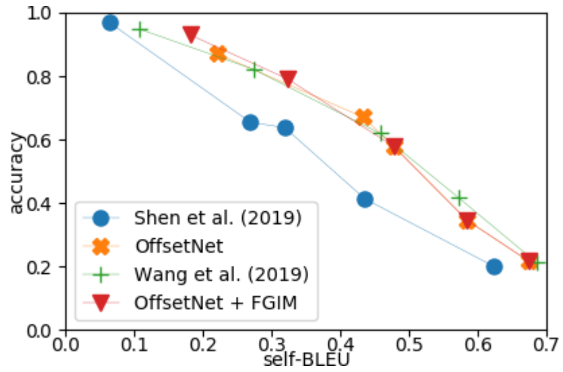

Figure 5: Comparison of plug and play methods for unsupervised style transfer on the Yelp sentiment transfer task's test set. Up and right is better

et al. (2019) achieve state-of-the-art on unsupervised sentiment transfer using FGIM, but applied to the latent space of a powerful transformer autoencoder. Since we want to conduct controlled experiments to find the best plug and play method, we integrated FGIM into our framework rather than directly comparing to their results. We treat the method by Shen et al. (2020) analogously.

For our learning-based model, we tune λ_{adv} on the development set from Yelp. After finding the best λ_{adv}, we inspect the behavior of the models at different levels of transfer by varying λ_{sty} ($\{0.1, 0.5, 0.9, 0.95, 0.99\}$), giving a tradeoff curve (more details in Appendix A.5.4). Analogously, we vary the multiplier for Shen et al. (2020) and the thresholds t for Wang et al. (2019) to obtain different tradeoffs between accuracy and self-BLEU. We also report the computational overhead each method incurs in addition to encoding and decoding.

Results. Figure 5 plots the tradeoff curves for the existing models, and ours with and without FGIM at inference time. We report accuracy and computational overhead in Table 2, for the most accurate points. Note that our model clearly outperforms that of Shen et al. (2020), confirming that learning the offset vectors in the autoencoder manifold is indeed beneficial.[6] Our model's performance is close to that of Wang et al. (2019), even without FGIM at inference time. Consequently, our model has a much lower computational overhead. With FGIM, our model shows an advantage at the high-accuracy end of the curve (top), increasing content preservation by 68% while reaching 98% of FGIM's transfer accuracy, though this is computationally expensive. This confirms that our training framework, while being very flexible (see Section 2.6), is a strong alternative not only in the supervised, but also in the unsupervised case.

Model	Acc.	s-BLEU	+Time
Shen et al.	96.8	6.5	0.5×
FGIM	94.9	10.8	70.0×
Emb2Emb + FGIM	93.1	18.1	2820.0×
Emb2Emb	87.1	22.1	1.0×

Table 2: Self-BLEU ("s-BLEU") on the Yelp sentiment transfer test set for the configurations in Figure 5 with highest transfer accuracy ("Acc."). "+Time" reports the inference-time slowdown factor due to each model's additional computation (relative to our method).

3.2.2 Pretraining

We have argued for a plug and play use of autoencoders because it allows generation tasks to benefit from independent research on autoencoders and potentially large datasets for pretraining. Here we measure the benefit of pretraining directly by simulating low-resource scenarios with limited style supervision. We consider three pretraining scenarios:

- **Upper bound**: We pretrain on all of the texts and labels; this serves as an upper bound for low-resource scenarios.
- **Plug and play**: A conventional plug and play scenario, where all of texts are available for pretraining, but only 10% of them are labeled (chosen at random) for use in training Φ.
- **Non plug and play**: A matched scenario with no pretraining ("non plug and play"), with only the reduced (10%) labeled data.

Results. Figure 6 shows the tradeoff curves in the same style as the last experiment. The benefit of pretraining in the low-resource setting is very clear, with the gap compared to the plug and play approach widening at lower transfer accuracy levels. The plug and play model's curve comes close to the "upper bound" (which uses ten times as much labeled data), highlighting the potential for pretraining an autoencoder for plug and play use in text generation tasks with relatively little labeled data.

[6] In Appendix B, we analyze the differences between these models' outputs qualitatively.

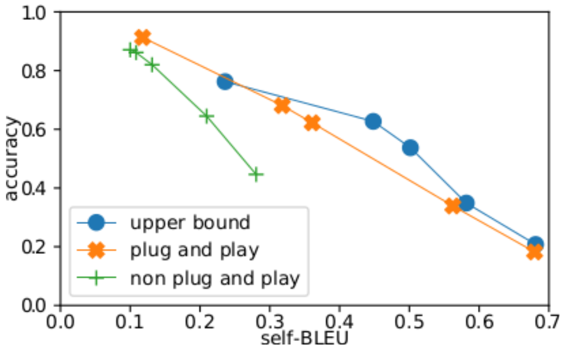

Figure 6: Sentiment transfer results for different model scenarios. Up and right is better.

4 Related Work

Text Style Transfer The most common approach to text style transfer is to learn a disentangled shared latent space that is agnostic to the style of the input. Style transfer is then achieved by training the decoder conditioned on the desired style attribute, (Hu et al., 2017; Shen et al., 2017; Fu et al., 2018; Zhao et al., 2018; Lample et al., 2019; Li et al., 2018; Logeswaran et al., 2018; Yang et al., 2018; Li et al., 2019), which hinders their employment in a plug and play fashion. Most methods either rely on adversarial objectives (Shen et al., 2017; Hu et al., 2017; Fu et al., 2018), retrieval (Li et al., 2018), or backtranslation (Lample et al., 2019; Logeswaran et al., 2018) to make the latent codes independent of the style attribute. Notable exceptions are Transformer-based (Dai et al., 2019; Sudhakar et al., 2019), use reinforcement learning for backtranslating through the discrete space (Liu and Liu, 2019), build pseudo-parallel corpora (Kruengkrai, 2019; Jin et al., 2019), or modify the latent-variable at inference time by following the gradient of a style classifier (Wang et al., 2019; Liu et al., 2020). Similar to our motivation, Li et al. (2019) aim at improving in-domain performance by incorporating out-of-domain data into training. However, because their model again conditions on the target data, they have to train the autoencoder jointly with the target corpus, defeating the purpose of large-scale pretraining.

In contrast to previous methods, Emb2Emb can be combined with any pretrained autoencoder even if it was not trained with target attributes in mind. It is therefore very close in spirit to plug and play language models by Dathathri et al. (2020) who showed how to use pretrained language models for controlled generation without any attribute conditioning (hence, the name). It is also similar to pretrain-and-plugin variational autoencoders (Duan et al., 2020), who learn small adapters with few parameters for a pretrained VAE to generate latent codes that decode into text with a specific attribute. However, these models cannot be conditioned on input text, and are thus not applicable to style transfer.

Textual Autoencoders Autoencoders are a very active field of research, leading to constant progress through denoising (Vincent et al., 2010), variational (Kingma and Welling, 2014; Higgins et al., 2017; Dai and Wipf, 2019), adversarial (Makhzani et al., 2016; Zhao et al., 2018), and, more recently, regularized (Ghosh et al., 2020) autoencoders, to name a few. Ever since Bowman et al. (2016) adopted variational autoencoders for sentences by employing a recurrent sequence-to-sequence model, improving both the architecture (Semeniuta et al., 2017; Prato et al., 2019; Liu and Liu, 2019; Gagnon-Marchand et al., 2019) and the training objective (Zhao et al., 2018; Shen et al., 2020) have received considerable attention. The goal is typically to improve both the reconstruction and generation performance (Cífka et al., 2018).

Our framework is completely agnostic to the type of autoencoder that is used, as long as it is trained to reconstruct the input. Hence, our framework directly benefits from any kind of modelling advancement in autoencoder research.

5 Conclusion

In this paper, we present *Emb2Emb*, a framework that reduces conditional text generation tasks to learning in the embedding space of a pretrained autoencoder. We propose an adversarial method and a neural architecture that are crucial for our method's success by making learning stay on the manifold of the autoencoder. Since our framework can be used with any pretrained autoencoder, it will benefit from large-scale pretraining in future research.

Acknowledgments

The authors thank Maarten Sap, Elizabeth Clark, and the anonymous reviewers for their helpful feedback. Florian Mai was supported by the Swiss National Science Foundation under the project LAOS, grant number "FNS-30216". Nikolaos Pappas was supported by the Swiss National Science Foundation under the project UNISON, grant number "P400P2_183911". This work was also partially supported by US NSF grant 1562364.

References

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2018. Unsupervised neural machine translation. In *International Conference on Learning Representations*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Gayatri Bhat, Sachin Kumar, and Yulia Tsvetkov. 2019. A margin-based loss with synthetic negative samples for continuous-output machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 199–205.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany. Association for Computational Linguistics.

Ondřej Cífka, Aliaksei Severyn, Enrique Alfonseca, and Katja Filippova. 2018. Eval all, trust a few, do wrong to none: Comparing sentence generation models. *arXiv preprint arXiv:1804.07972*.

Bin Dai and David Wipf. 2019. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations*.

Ning Dai, Jianze Liang, Xipeng Qiu, and Xuanjing Huang. 2019. Style transformer: Unpaired text style transfer without disentangled latent representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5997–6007, Florence, Italy. Association for Computational Linguistics.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yu Duan, Canwen Xu, Jiaxin Pei, Jialong Han, and Chenliang Li. 2020. Pre-train and plug-in: Flexible conditional text generation with variational autoencoders. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 253–262, Online. Association for Computational Linguistics.

Jesse Engel, Matthew Hoffman, and Adam Roberts. 2018. Latent constraints: Learning to generate conditionally from unconditional generative models. In *International Conference on Learning Representations*.

Thibault Fevry and Jason Phang. 2018. Unsupervised sentence compression using denoising autoencoders. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 413–422, Brussels, Belgium. Association for Computational Linguistics.

Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. 2018. Style transfer in text: Exploration and evaluation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Jules Gagnon-Marchand, Hamed Sadeghi, Md Akmal Haidar, and Mehdi Rezagholizadeh. 2019. Salsa-text: self attentive latent space based adversarial text generation. In *Canadian Conference on Artificial Intelligence*, pages 119–131. Springer.

Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Scholkopf. 2020. From variational to deterministic autoencoders. In *International Conference on Learning Representations*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew M Botvinick, Shakir Mohamed, and Alexander Lerchner. 2017. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org.

Zhijing Jin, Di Jin, Jonas Mueller, Nicholas Matthews, and Enrico Santus. 2019. IMaT: Unsupervised text attribute transfer via iterative matching and translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3097–3109, Hong Kong, China. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *International Conference on Learning Representations*.

Canasai Kruengkrai. 2019. Learning to flip the sentiment of reviews from non-parallel corpora. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6312–6317.

Sachin Kumar and Yulia Tsvetkov. 2019. Von Mises-Fisher loss for training sequence to sequence models with continuous outputs. In *International Conference on Learning Representations*.

Guillaume Lample, Sandeep Subramanian, Eric Smith, Ludovic Denoyer, Marc'Aurelio Ranzato, and Y-Lan Boureau. 2019. Multiple-attribute text rewriting. In *International Conference on Learning Representations*.

Dianqi Li, Yizhe Zhang, Zhe Gan, Yu Cheng, Chris Brockett, Bill Dolan, and Ming-Ting Sun. 2019. Domain adaptive text style transfer. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3302–3311. Association for Computational Linguistics.

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1865–1874. Association for Computational Linguistics.

Danyang Liu and Gongshen Liu. 2019. A transformer-based variational autoencoder for sentence generation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE.

Dayiheng Liu, Jie Fu, Yidan Zhang, Chris Pal, and Jiancheng Lv. 2020. Revision in continuous space: Unsupervised text style transfer without adversarial learning. In *AAAI*.

Lajanugen Logeswaran, Honglak Lee, and Samy Bengio. 2018. Content preserving text generation with attribute controls. In *Advances in Neural Information Processing Systems*, pages 5103–5113.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. 2016. Adversarial autoencoders. In *International Conference on Learning Representations*.

Alexander Mathews, Lexing Xie, and Xuming He. 2018. Simplifying sentences with sequence to sequence models. *arXiv preprint arXiv:1805.05557*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar GuÌ‡lçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Sergiu Nisioi, Sanja Štajner, Simone Paolo Ponzetto, and Liviu P. Dinu. 2017. Exploring neural text simplification models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 85–91, Vancouver, Canada. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? Adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Gabriele Prato, Mathieu Duchesneau, Sarath Chandar, and Alain Tapp. 2019. Towards lossless encoding of sentences. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1577–1583, Florence, Italy. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2017. A hybrid convolutional variational autoencoder for text generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 627–637, Copenhagen, Denmark. Association for Computational Linguistics.

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.

Tianxiao Shen, Jonas Mueller, Regina Barzilay, and Tommi Jaakkola. 2020. Educating text autoencoders: Latent representation guidance via denoising. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Online. PMLR.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: Masked sequence to sequence pre-training for language generation. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936, Long Beach, California, USA. PMLR.

Akhilesh Sudhakar, Bhargav Upadhyay, and Arjun Maheswaran. 2019. “Transforming” delete, retrieve, generate approach for controlled text style transfer. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3269–3279, Hong Kong, China. Association for Computational Linguistics.

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.

Dušan Variš and Ondřej Bojar. 2019. Unsupervised pretraining for neural machine translation using elastic weight consolidation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 130–135, Florence, Italy. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408.

Ke Wang, Hang Hua, and Xiaojun Wan. 2019. Controllable unsupervised text attribute transfer via editing entangled latent representation. In *Advances in Neural Information Processing Systems*, pages 11034–11044.

Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. 2015. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1006–1011.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Zichao Yang, Zhiting Hu, Chris Dyer, Eric P Xing, and Taylor Berg-Kirkpatrick. 2018. Unsupervised text style transfer using language models as discriminators. In *Advances in Neural Information Processing Systems*, pages 7287–7298.

Xingxing Zhang and Mirella Lapata. 2017. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 584–594, Copenhagen, Denmark. Association for Computational Linguistics.

Junbo Jake Zhao, Yoon Kim, Kelly Zhang, Alexander M Rush, Yann LeCun, et al. 2018. Adversarially regularized autoencoders. In *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR.

A Experimental Details

We first describe the experimental details that are common to the experiments on both datasets. Dataset-specific choices are listed in their respective subsections.

A.1 Preprocessing and Tokenization

We do not apply any further preprocessing to the datasets that we obtain. We use BPE for tokenization, and restrict the vocabulary to 30,000. We truncate all inputs to 100 tokens at maximum.

A.2 Experimental Setup

Computing Infrastructure. For all of our experiments, we relied on a computation cluster with a variety of different GPUs with at minimum 12GB GPU memory and 50GB RAM. For the text simplification experiments where we measure training speed, we ran all experiments on the same machine (with a GeForce GTX 1080 Ti) in succession to ensure a fair comparison.

Implementation. We used Python 3.7 with PyTorch 1.4 for all our experiments. Our open-source implementation is available at https://github.com/florianmai/emb2emb.

Adversarial Training. We employ a 2-layer MLP with 300 hidden units and ReLU activation as discriminator, and train it using Adam with a learning rate of 0.00001 (the remaining parameters are left at their PyTorch defaults). We train it in alternating fashion with the generator Φ, in batches of size 64.

A.3 Neural Architectures

Encoder For encoding, we employ a one-layer bidirectional LSTM as implemented in PyTorch. To obtain the fixed-size bottleneck, we average the last hidden state of both directions. The input size (and token embedding size) is 300.

Decoder For decoding, we initialize the hidden state of a one-layer LSTM decoder as implemented in PyTorch with the fixed size embedding. During training, we apply teacher forcing with a probability of 0.5. The input size is 300. We use greedy decoding at inference time.

Transformation Φ. We train all neural network architectures with one layer. The hidden size is set to the same as the input size, which in turn is determined by the size of the autoencoder bottleneck. Hence, the MLP and OffsetNet have the same number of parameters. Due to its extra weight matrix at the output-layer, the ResNet has 50% more parameters than the other models. All networks use the SELU activation function. All training runs with our model were performed with the Adam optimizer.

A.4 Text Simplification

A.4.1 Dataset Details

We evaluate on the WikiLarge dataset by Zhang and Lapata (2017), which consists of sentence pairs extracted from Wikipedia, where the input is in English and the output is in simple English. It contains of 296,402 training pairs, 2,000 development pairs, and 359 pairs for testing. The 2,359 development and test pairs each come with 8 human-written reference sentences to compute the BLEU and SARI overlap with. The dataset can be downloaded from https://github.com/XingxingZhang/dress.

A.4.2 Experimental Details

Training our model. We use a fixed learning rate of 0.0001 to train our model for 10 epochs. We evaluate the validation set performance in terms of BLEU after every epoch and save the iteration with the best validation loss performance.

Training S2S models. For all S2S models we compare against in Section 3.1.1, we select the best performing run on the validation set among the learning rates $\{0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000005\}$, and also assess the validation set performance after each of the 20 epochs. Training is performed with the Adam optimizer.

Encoder hyperparameters We use a 1-layer bidirectional LSTM with a memory size of 1024 and an input size of 300.

Number of Parameters All models share the same encoder and decoder architecture, consisting of 34,281,600 parameters in total. The mappings MLP, OffsetNet, and ResNet have 2,097,132, 2,097,132, and 3,145,708 parameters, respectively. We report total numbers for the models used in the experimental details below.

Evaluation metrics. For computing BLEU, we use the Python NLTK 3.5 library.[7]

[7] https://www.nltk.org/api/nltk.translate.html#module-nltk.translate.bleu_score

For computing SARI score, we use the implementation provided by (Xu et al., 2016) at https://github.com/cocoxu/simplification/blob/master/SARI.py.

A.4.3 SARI Score by λ_{adv}

Experimental details. We measure the performance of our model on the development set of WikiLarge in terms of SARI score. These results are for the same training run for which we reported the BLEU score, hence, the stopping criterion for early stopping was BLEU, and we report the results for all 10 exponentially increasing values of λ_{adv}. The best value when using BLEU score as stopping criterion is $\lambda_{adv} = 0.032$.

Results. The results in Figure 7 show the same pattern as for the BLEU score, although with a smaller relative gain of 23% when using the adversarial term.

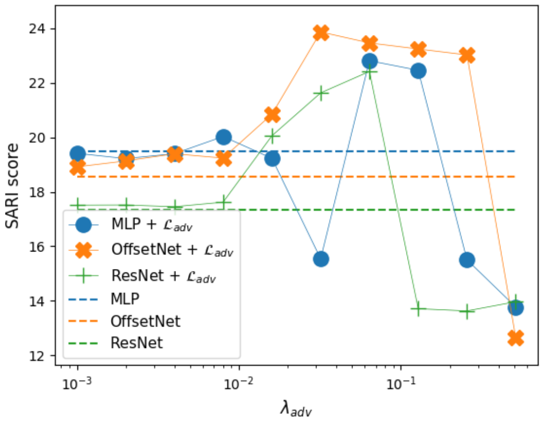

Figure 7: Performance on WikiLarge in terms of SARI score (higher is better) by weight for the adversarial term λ_{adv}.

A.4.4 Development Set Results for Comparison to S2S Models

In Table 3, we report the development set performances corresponding to the experiments reported in Section 3.1.1. For each model, we also specify the best learning rate, if applicable, and the number of parameters in the model

A.5 Sentiment Transfer

A.5.1 Dataset Details

We evaluate on the Yelp dataset as preprocessed by (Shen et al., 2017), which consists of sentences with positive or negative sentiment extracted from restaurant reviews. The training set consists of 176,787 negative and 267,314 positive examples. The development set has 25,278 negative and 38,205 positive examples, and the test set has 50,278 negative and 76,392 positive examples. The dataset can be downloaded from https://github.com/shentianxiao/language-style-transfer/tree/master/data/yelp.

Model	BLEU	SARI	LR	$\|\Theta\|$
S2S-Scratch	3.2	14.3	0.0001	34.3m
S2S-Pretrain	5.9	15.1	0.0005	34.3m
S2S-MLP	8.6	16.0	0.0001	36.4m
S2S-Freeze	17.4	20.1	0.00005	36.4m
Ours	**26.7**	**23.5**	-	36.4m

Table 3: Text simplification performance of model variants of seq2seq training on the development set. $|\Theta|$ denotes the number of parameters for each model.

Training our models. We use a fixed learning rate of 0.00005 to train our model for 10 epochs (for the ablations) or 20 epochs (for the final model). We evaluate the validation set performance in terms of self-BLEU plus transfer accuracy after every epoch and save the iteration with the best validation loss performance.

For all models involving training the mapping Φ (including the ablation below), we perform a search of λ_{adv} among the values $\{0.008, 0.016, 0.032, 0.0640.128\}$. We select them based on the following metric: $\sum_{i=1}^{5}(BLEU(\lambda_{adv}, \lambda^i_{sty}) + accuracy(\lambda_{adv}, \lambda^i_{sty})$, where λ^i_{sty} corresponds to the i-th value of λ_{sty} that we have used to obtain the BLEU-accuracy tradeoff curve. By $BLEU(\lambda_{adv}, \lambda^i_{sty})$ and $accuracy(\lambda_{adv}, \lambda^i_{sty})$, respectively, we mean the score resulting from training with the given parameters.

Encoder hyperparameters We use a 1-layer bidirectional LSTM with a memory size of 512.

Number of Parameters All models again share the same encoder and decoder architecture, consisting of 22,995,072 parameters in total. The mappings MLP, OffsetNet, and ResNet have 524,288, 524,288, and 786,432 parameters, respectively. Hence, the total number of parameters for our models is 23.5m, whereas the variants we report as Shen et al. and FGIM have 23m parameters.

Sentiment classifier on autoencoder manifold. For binary classification, we train a 1-layer MLP with a hidden size of 512 with Adam using a learning rate of 0.0001. For regularization, we use dropout with $p = 0.5$ at the hidden and input layer, and also add isotropic Gaussian noise with a standard deviation of 0.5 to the input features.

BERT classifier. The DistilBERT classifier is trained using the HuggingFace transformers library.[8] We train it for 30 epochs with a batch size of 64 and a learning rate of 0.00002 for Adam, with a linear warm-up period over the first 3000 update steps. We evaluate the validation set performance every 5000 steps and save the best model.

A.5.2 Implementation of Wang et al. Baseline

We reimplemented the Fast Gradient Iterative Modification method by (Wang et al., 2019) to either i) follow the gradient of the sentiment classifier from the input, or ii) from the output of Φ, follow the gradient of the complete loss function of training Φ.

Following the implementation by (Wang et al., 2019), in all runs, we repeat the computation for weights $\omega \in \{1, 10, 100, 1000\}$ and stop at the first weight that leads to the classification probability exceeding a threshold t. For each weight, we make 30 gradient steps at maximum.

The Wang et al. (2019) baseline is generated from choosing $t = \{0.5, 0.9, 0.99, 0.999, 0.9999\}$, i.e., we choose lower thresholds to stop the gradient descent from changing the input too much towards the target attribute, leading to lower transfer accuracy performances.

When we apply FGIM to the output of Φ in our model (with the more sophisticated loss function, where we set $\lambda_{sty} = 0.5$), we apply the same thresholds.

A.5.3 Development Set Result for Comparison of Plug and Play

In Figure 8, we report the development set result corresponding to the test set results of the experiments presented in Section 3.2.1. These results are shown for $\lambda_{adv} = 0.008$, which performed the best in terms of the development score metric introduced in the training details.

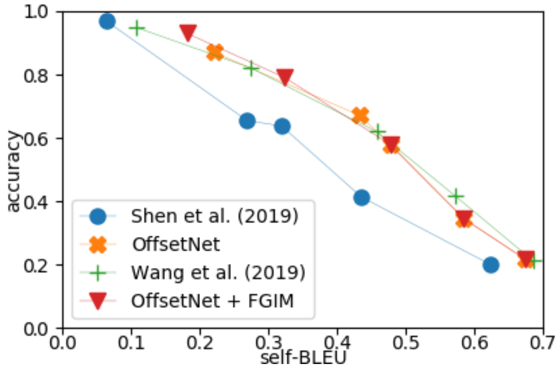

Figure 8: Comparison of plug and play methods for unsupervised style transfer on the Yelp sentiment transfer task's development set. Up and right is better

A.5.4 Model Analysis

Experimental Setup We investigate the effect of OffsetNet and the adversarial training term on our unsupervised style transfer model by measuring the self-BLEU score with the input sentence and the accuracy of a separately trained BERT classifier (achieving 97.8% classification accuracy) on the Yelp development set. We again report the best performance among 6 exponentially increasing λ_{adv} values for each model. To inspect the behavior of the models at varying levels of transfer, we trained and plotted one model each for $\lambda_{sty} \in \{0.1, 0.5, 0.9, 0.95, 0.99\}$.

Results. The results in Figure 9 show that OffsetNet reaches better transfer accuracy than the MLP at comparable self-BLEU scores. The performance drops significantly if the adversarial term is not used. This confirms the importance of our design decisions.

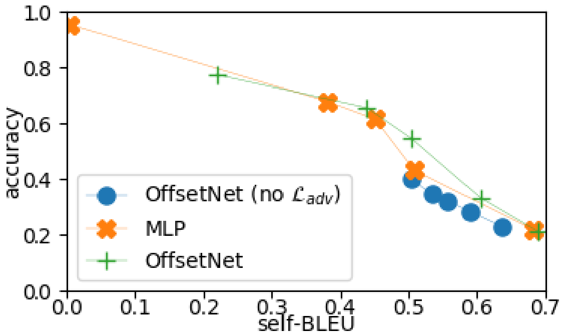

Figure 9: Ablation of our model components on the Yelp sentiment transfer tasks. Up and right is better.

[8] Specifically, we use the run_glue.py script in from https://github.com/huggingface/transformers and only replace the SST-2 dataset with the Yelp dataset. We used the commit "11c3257a18c4b5e1a3c1746eefd96f180358397b" for training our model.

B Qualitative Analysis

We provide several example outputs of our method in comparison to the outputs of the baseline by Shen et al. (2020) in Tables 4, 5, 6, and 7. Moreover, we show how the output evolves as the multiplier and λ_{sty} (i.e., the level of transfer accuracy) increases.

In our qualitative analysis we generally observe that both models generate similar outputs when the inputs are short and can be transferred by only changing or deleting single words (e.g., Table 4). We observe that grammaticality degrades in both methods for higher transfer levels. However, our method is more often able to preserve the content of the input as the transfer accuracy increases: At a multiplier of 3.0, the method by Shen et al. (2020) outputs rather general positive statements that are mostly disconnected from the input, whereas our method is able to stay on the topic of the input statement. This observation matches the quantitative results from Section 3.2.1, where our method attains substantially higher self-BLEU scores at comparable levels of transfer accuracy.

However, it is clear that both models mostly rely on exchanging single words in order to change the sentiment classification. In the example from Table 5, our model changes the input "the cash register area was empty and no one was watching the store front ." to the rather unnatural sentence "the cash area was great and was wonderful with watching the front desk ." instead of the more natural, but lexically distant reference sentence "the store front was well attended ". We think that this is best explained by the fact that we use a denoising autoencoder with a simple noise function (deleting random words) for these experiments, which encourages sentences within a small edit-distance to be close to each other in the embedding space (Shen et al., 2020). Denoising autoencoders with a more sophisticated noise functions focused on semantics could possibly mitigate this, but is out of scope for this study.

multiplier / λ_{sty}	Shen et al. (2019)	Ours
1.5 / 0.5	i will be back .	i will be back .
2.0 / 0.9	i will be back back	i will definitely be back .
2.5 / 0.95	i will definitely be back .	i will definitely be back
3.0 / 0.99	i love this place !	i will be back !

Table 4: **Input**: i will never be back .

multiplier / λ_{sty}	Shen et al. (2019)	Ours
1.5 / 0.5	the cash area was great and the the best staff	the cash area was great and was wonderful one watching the front desk .
2.0 / 0.9	the cash register area was empty and no one was watching the store front .	the cash area was great and was wonderful with watching the front desk .
2.5 / 0.95	the cash bar area was great and no one was the friendly staff .	the cash area was great and was wonderful with watching the front desk .
3.0 / 0.99	the great noda area and great and wonderful staff .	the cash area was great and her and the staff is awesome !

Table 5: **Input**: the cash register area was empty and no one was watching the store front . **Reference**: the store front was well attended

multiplier / λ_{sty}	Shen et al. (2019)	Ours
1.5 / 0.5	we sit down and we got some really slow and lazy service .	we sit down and we got some really slow and lazy service .
2.0 / 0.9	we sit down and we got really awesome and speedy service .	we sit down and we got some really slow and lazy service .
2.5 / 0.95	we sit down and we we grab the casual and and service .	we sit down and we got some really great and and awesome service .
3.0 / 0.99	we sit great and and some really great and awesome atmosphere .	we sit down and we got some really comfortable and and service .

Table 6: **Input**: the cash register area was empty and no one was watching the store front . **Reference**: the service was quick and responsive

multiplier / λ_{sty}	Shen et al. (2019)	Ours
1.5 / 0.5	definitely disappointed that i ’m not my birthday !	definitely disappointed that i could not use my birthday gift !
2.0 / 0.9	definitely disappointed that i have a great !	definitely not disappointed that i could use my birthday gift !
2.5 / 0.95	definitely super disappointed and i ’ll definitely have a great gift !	definitely disappointed that i could use my birthday gift !
3.0 / 0.99	definitely delicious and i love the !	definitely disappointed that i could use my birthday gift !

Table 7: **Input**: definitely disappointed that i could not use my birthday gift ! **Reference**: definitely not disappointed that i could use my birthday gift !