## 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 $\Phi$, 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 $\Phi$.** 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 $\lambda_{adv}$

**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 $\lambda_{adv}$. 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.

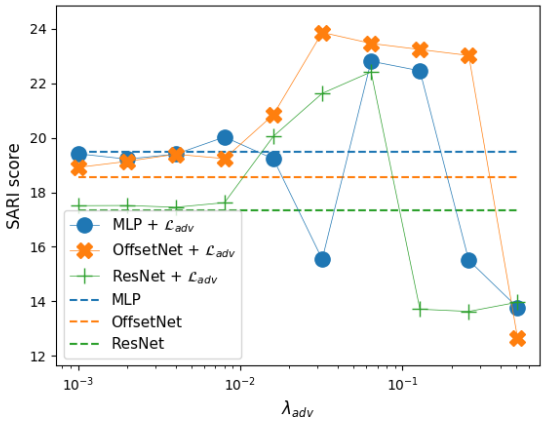


Figure 7: Performance on WikiLarge in terms of SARI score (higher is better) by weight for the adversarial term $\lambda_{adv}$.

### 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

| Model | BLEU | SARI | LR | $\lvert\Theta\rvert$ |
|---|---|---|---|---|
| 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.

## 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.

**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 $\Phi$ (including the ablation below), we perform a search of $\lambda_{adv}$ 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 $\lambda^i_{sty}$ corresponds to the $i$-th value of $\lambda_{sty}$ 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 $\Phi$, follow the gradient of the complete loss function of training $\Phi$.

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 $\Phi$ 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.

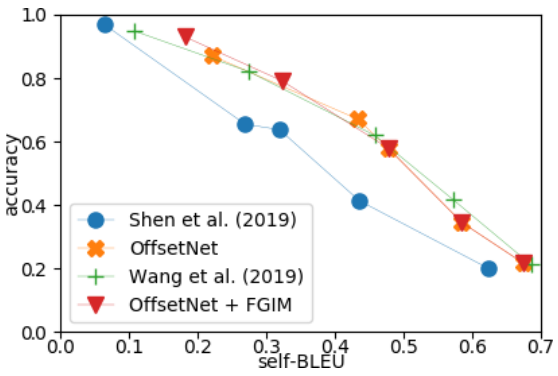


Figure 8: Comparison of plug and play methods for unsupervised style transfer on the Yelp sentiment transfer task's development set. Up and right is better

### 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 $\lambda_{adv}$ 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.

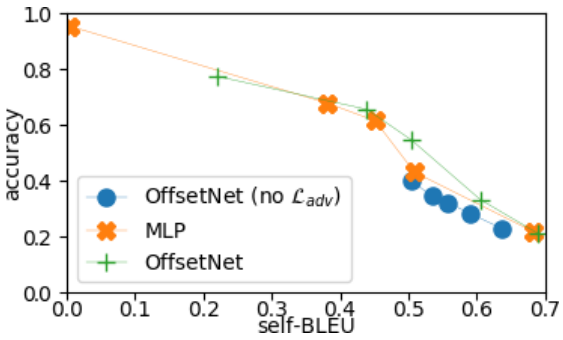


Figure 9: Ablation of our model components on the Yelp sentiment transfer tasks. Up and right is better.

[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 $\lambda_{sty}$ (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 / $\lambda_{sty}$ | 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 / $\lambda_{sty}$ | 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 / $\lambda_{sty}$ | 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 / $\lambda_{sty}$ | 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 !