A Hybrid Convolutional Variational Autoencoder for Text Generation

Stanislau Semeniuta[1] **Aliaksei Severyn**[2] **Erhardt Barth**[1]
[1]Universität zu Lübeck, Institut für Neuro- und Bioinformatik
{stas,barth}@inb.uni-luebeck.de
[2]Google Research Europe
severyn@google.com

Abstract

In this paper we explore the effect of architectural choices on learning a variational autoencoder (VAE) for text generation. In contrast to the previously introduced VAE model for text where both the encoder and decoder are RNNs, we propose a novel hybrid architecture that blends fully feed-forward convolutional and deconvolutional components with a recurrent language model. Our architecture exhibits several attractive properties such as faster run time and convergence, ability to better handle long sequences and, more importantly, it helps to avoid the issue of the VAE collapsing to a deterministic model.

1 Introduction

Generative models of texts are currently at the cornerstone of natural language understanding enabling recent breakthroughs in machine translation (Bahdanau et al., 2014; Wu et al., 2016), dialogue modelling (Serban et al., 2016), abstractive summarization (Rush et al., 2015), etc.

Currently, RNN-based generative models hold state-of-the-art results in both unconditional (Józefowicz et al., 2016; Ha et al., 2016) and conditional (Vinyals et al., 2014) text generation. At a high level, these models represent a class of autoregressive models that work by generating outputs sequentially one step at a time where the next predicted element is conditioned on the history of elements generated thus far.

Variational autoencoders (VAE), recently introduced by (Kingma and Welling, 2013; Rezende et al., 2014), offer a different approach to generative modeling by integrating stochastic latent variables into the conventional autoencoder architecture. The primary purpose of learning VAE-based generative models is to be able to generate realistic examples as if they were drawn from the input data distribution by simply feeding noise vectors through the decoder. Additionally, the latent representations obtained by applying the encoder to input examples give a fine-grained control over the generation process that is harder to achieve with more conventional autoregressive models. Similar to compelling examples from image generation, where it is possible to condition generated human faces on various attributes such as hair, skin color and style (Yan et al., 2015; Larsen et al., 2015), in text generation it should be possible to also control various attributes of the generated sentences, such as, for example, sentiment or writing style.

While training VAE-based models seems to pose little difficulty when applied to the tasks of generating natural images (Bachman, 2016; Gulrajani et al., 2016) and speech (Fraccaro et al., 2016), their application to natural text generation requires additional care (Bowman et al., 2016; Miao et al., 2015). As discussed by Bowman et al. (2016), the core difficulty of training VAE models is the collapse of the latent loss (represented by the KL divergence term) to zero. In this case the generator tends to completely ignore latent representations and reduces to a standard language model. This is largely due to the high modeling power of the RNN-based decoders which with sufficiently small history can achieve low reconstruction errors while not relying on the latent vector provided by the encoder.

In this paper, we propose a novel VAE model for texts that is more effective at forcing the decoder to make use of latent vectors. Contrary to existing work, where both encoder and decoder layers are LSTMs, the core of our model is a feed-forward architecture composed of one-dimensional convolutional and deconvolutional (Zeiler et al., 2010) layers. This choice of architecture helps to gain

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 627–637
Copenhagen, Denmark, September 7–11, 2017.

more control over the KL term, which is crucial for training a VAE model. Given the difficulty of generating long sequences in a fully feed-forward manner, we augment our network with an RNN language model layer. To the best of our knowledge, this paper is the first work that successfully applies deconvolutions in the decoder of a latent variable generative model of natural text. We empirically verify that our model is easier to train than its fully recurrent alternative, which, in our experiments, fails to converge on longer texts. To better understand why training VAEs for texts is difficult we carry out detailed experiments, discuss optimization difficulties, and propose effective ways to address them. Finally, we demonstrate that sampling from our model yields realistic texts.

2 Related Work

So far, the majority of neural generative models of text are built on the autoregressive assumption (Larochelle and Murray, 2011; van den Oord et al., 2016). Such models assume that the current data element can be accurately predicted given sufficient history of elements generated thus far. The conventional RNN based language models fall into this category and currently dominate the language modeling and generation problem in NLP. Neural architectures based on recurrent (Józefowicz et al., 2016; Zoph and Le, 2016; Ha et al., 2016) or convolutional decoders (Kalchbrenner et al., 2016; Dauphin et al., 2016) provide an effective solution to this problem.

A recent work by Bowman et al. (2016) tackles language generation problem within the VAE framework (Kingma and Welling, 2013; Rezende et al., 2014). The authors demonstrate that with some care it is possible to successfully learn a latent variable generative model of text. Although their model is slightly outperformed by a traditional LSTM (Hochreiter and Schmidhuber, 1997) language model, their model achieves a similar effect as in computer vision where one can (i) sample realistic sentences by feeding randomly generated novel latent vectors through the decoder and (ii) linearly interpolate between two points in the latent space. Miao et al. (2015) apply VAE to bag-of-words representations of documents and the answer selection problem achieving good results on both tasks. Yang et al. (2017) discuss a VAE consisting of RNN encoder and CNN decoder so that the decoder's receptive field is limited. They demonstrate that this allows for a better control of KL and reconstruction terms. Hu et al. (2017) build a VAE for text generation and design a cost function that encourages interpretability of the latent variables. Zhang et al. (2016), Serban et al. (2016) and Zhao et al. (2017) apply VAE to sequence-to-sequence problems, improving over deterministic alternatives. Chen et al. (2016) propose a hybrid model combining autoregressive convolutional layers with the VAE. The authors make an argument based on the Bit-Back coding (Hinton and van Camp, 1993) that when the decoder is powerful enough the best thing for the encoder to do is to make the posterior distribution equivalent to the prior. While they experiment on images, this argument is very relevant to the textual data. A recent work by Bousquet et al. (2017) approaches VAEs and GANs from the optimal transport point of view. The authors show that commonly known blurriness of samples from VAEs trained on image data are a necessary property of the model. While the implications of their argument to models combining latent variables and autoregressive layers trained on non-image data are still unclear, the argument supports the hypothesis of Chen et al. (2016) that difficulty of training a hybrid model is not caused by a simple optimization difficulty but rather may be a more principled issue.

Various techniques to improve training of VAE models where the total cost represents a trade-off between the reconstruction cost and KL term have been used so far: KL-term annealing and input dropout (Bowman et al., 2016; Sønderby et al., 2016), imposing structured sparsity on latent variables (Yeung et al., 2016) and more expressive formulations of the posterior distribution (Rezende and Mohamed, 2015; Kingma et al., 2016). A work by (Mescheder et al., 2017) follows the same motivation and combines GANs and VAEs allowing a model to use arbitrary complex formulations of both prior and posterior distributions. In Section 3.4 we propose another efficient technique to control the trade-off between KL and reconstruction terms.

3 Model

In this section we first briefly explain the VAE framework of Kingma and Welling (2013), then describe our hybrid architecture where the feed-

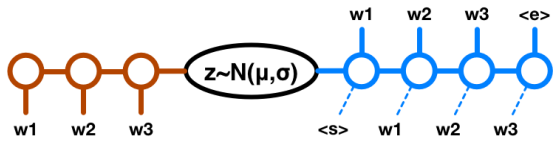


Figure 1: LSTM VAE model of (Bowman et al., 2016)

forward part is composed of a fully convolutional encoder and a decoder that combines deconvolutional layers and a conventional RNN. Finally, we discuss optimization recipes that help VAE to respect latent variables, which is critical training a model with a meaningful latent space and being able to sample realistic sentences.

3.1 Variational Autoencoder

The VAE is a recently introduced latent variable generative model, which combines variational inference with deep learning. It modifies the conventional autoencoder framework in two key ways. Firstly, a deterministic internal representation $\mathbf{z}$ (provided by the encoder) of an input $\mathbf{x}$ is replaced with a posterior distribution $q(\mathbf{z}|\mathbf{x})$. Inputs are then reconstructed by sampling $\mathbf{z}$ from this posterior and passing them through a decoder. To make sampling easy, the posterior distribution is usually parametrized by a Gaussian with its mean and variance predicted by the encoder. Secondly, to ensure that we can sample from any point of the latent space and still generate valid and diverse outputs, the posterior $q(\mathbf{z}|\mathbf{x})$ is regularized with its KL divergence from a prior distribution $p(\mathbf{z})$. The prior is typically chosen to be also a Gaussian with zero mean and unit variance, such that the KL term between posterior and prior can be computed in closed form (Kingma and Welling, 2013). The total VAE cost is composed of the reconstruction term, i.e., negative log-likelihood of the data, and the KL regularizer:

$$\begin{aligned} J_{vae} &= KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &\quad -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log\ p(\mathbf{x}|\mathbf{z})] \end{aligned} \tag{1}$$

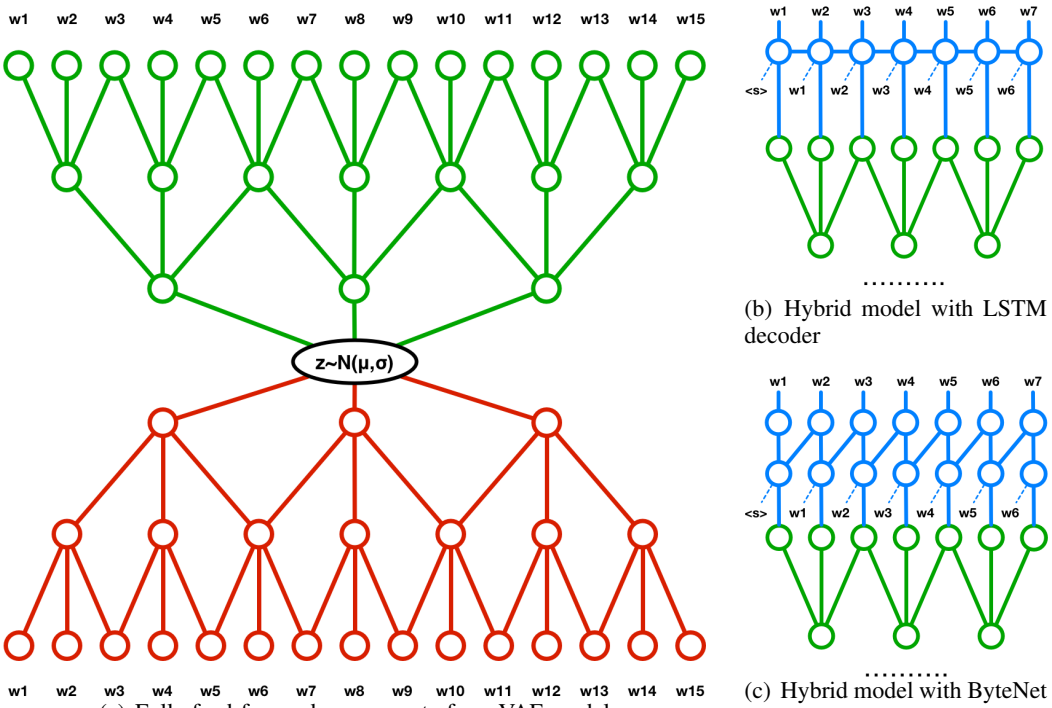
Kingma and Welling (2013) show that the loss function from Eq (1) can be derived from the probabilistic model perspective and it is an upper bound on the true negative likelihood of the data.

One can view a VAE as a traditional Autoencoder with some restrictions imposed on the internal representation space. Specifically, using a sample from the $q(\mathbf{z}|\mathbf{x})$ to reconstruct the input instead of a deterministic $\mathbf{z}$, forces the model to map an input to a region of the space rather than to a single point. The most straight-forward way to achieve a good reconstruction error in this case is to predict a very sharp probability distribution effectively corresponding to a single point in the latent space (Raiko et al., 2014). The additional KL term in Eq (1) prevents this behavior and forces the model to find a solution with, on one hand, low reconstruction error and, on the other, predicted posterior distributions close to the prior. Thus, the decoder part of the VAE is capable of reconstructing a sensible data sample from every point in the latent space that has non-zero probability under the prior. This allows for straightforward generation of novel samples and linear operations on the latent codes. Bowman et al. (2016) demonstrate that this does not work in the fully deterministic Autoencoder framework . In addition to regularizing the latent space, KL term indicates how much information the VAE stores in the latent vector.

Bowman et al. (2016) propose a VAE model for text generation where both encoder and decoder are LSTM networks (Figure 1). We will refer to this model as LSTM VAE in the remainder of the paper. The authors show that adapting VAEs to text generation is more challenging as the decoder tends to ignore the latent vector (KL term is close to zero) and falls back to a language model. Two training tricks are required to mitigate this issue: (i) KL-term annealing where its weight in Eq (1) gradually increases from 0 to 1 during the training; and (ii) applying dropout to the inputs of the decoder to limit its expressiveness and thereby forcing the model to rely more on the latent variables. We will discuss these tricks in more detail in Section 3.4. Next we describe a deconvolutional layer, which is the core element of the decoder in our VAE model.

3.2 Deconvolutional Networks

A deconvolutional layer (also referred to as transposed convolutions (Gulrajani et al., 2016) and fractionally strided convolutions (Radford et al., 2015)) performs spatial up-sampling of its inputs and is an integral part of latent variable generative models of images (Radford et al., 2015; Gulrajani et al., 2016) and semantic segmentation algorithms (Noh et al., 2015). Its goal is to perform an "inverse" convolution operation and increase spatial size of the input while decreasing the number of feature maps. This operation can be viewed as



(a) Fully feed-forward component of our VAE model

(b) Hybrid model with LSTM decoder

(c) Hybrid model with ByteNet decoder

Figure 2: Illustrations of our proposed models.

a backward pass of a convolutional layer and can be implemented by simply switching the forward and backward passes of the convolution operation. In the context of generative modeling based on global representations, the deconvolutions are typically used as follows: the global representation is first linearly mapped to another representation with small spatial resolution and large number of feature maps. A stack of deconvolutional layers is then applied to this representation, each layer progressively increasing spatial resolution and decreasing the amount of feature channels. The output of the last layer is an image or, in our case, a text fragment. A notable example of such a model is the deep network of (Radford et al., 2015) trained with adversarial objective. Our model uses a similar approach but is instead trained with the VAE objective.

There are two primary motivations for choosing deconvolutional layers instead of the dominantly used recurrent ones: firstly, such layers have extremely efficient GPU implementations due to their fully parallel structure. Secondly, feed-forward architectures are typically easier to optimize than their recurrent counterparts, as the number of back-propagation steps is constant and potentially much smaller than in RNNs. Both points become significant as the length of the generated text increases. Next, we describe our VAE architecture that blends deconvolutional and RNN layers in the decoder to allow for better control over the KL-term.

3.3 Hybrid Convolutional-Recurrent VAE

Our model is composed of two relatively independent modules. The first component is a standard VAE where the encoder and decoder modules are parametrized by convolutional and deconvolutional layers respectively (see Figure 2(a)). This architecture is attractive for its computational efficiency and simplicity of training.

The other component is a recurrent language model consuming activations from the deconvolutional decoder concatenated with the previous output characters. We consider two flavors of recurrent functions: a conventional LSTM network (Figure 2(b)) and a stack of masked convolutions also known as the ByteNet decoder from Kalchbrenner et al. (2016) (Figure 2(c)). The primary reason for having a recurrent component in the decoder is to capture dependencies between elements of the text sequences – a hard task for a fully feed-forward architecture. Indeed, the conditional distribution $P(\mathbf{x}|\mathbf{z}) = P(x_1, \ldots, x_n|\mathbf{z})$ of generated sentences cannot be richly represented with a feed-forward network. Instead, it factor-

izes as: $P(x_1, \ldots, x_n|\mathbf{z}) = \prod_i P(x_i|\mathbf{z})$ where components are independent of each other and are conditioned only on $\mathbf{z}$. To minimize the reconstruction cost the model is forced to encode every detail of a text fragment. A recurrent language model instead models the full joint distribution of output sequences without having to make independence assumptions $P(x_1, \ldots, x_n|\mathbf{z}) = \prod_i P(x_i|x_{i-1}, \ldots, x_1, \mathbf{z})$. Thus, adding a recurrent layer on top of our fully feed-forward encoder-decoder architecture relieves it from encoding every aspect of a text fragment into the latent vector and allows it to instead focus on more high-level semantic and stylistic features.

Note that the feed-forward part of our model is different from the existing fully convolutional approaches of Dauphin et al. (2016) and Kalchbrenner et al. (2016) in two respects: firstly, while being fully parallelizable during training, these models still require predictions from previous time steps during inference and thus behave as a variant of recurrent networks. In contrast, expansion of the z vector is fully parallel in our model (except for the recurrent component). Secondly, our model down- and up-samples a text fragment during processing while the existing fully convolutional decoders do not. Preserving spatial resolution can be beneficial to the overall result, but comes at a higher computational cost. Lastly, we note that our model imposes an upper bound on the size of text samples it is able to generate. While it is possible to model short texts by adding special padding characters at the end of a sample, generating texts longer than certain thresholds is not possible by design. This is not an unavoidable restriction, since the model can be extended to generate variable sized text fragments by, for example, variable sized latent codes. These extensions however are out of scope of this work.

3.4 Optimization Difficulties

The addition of the recurrent component results in optimization difficulties that are similar to those described by Bowman et al. (2016). In most cases the model converges to a solution with a vanishingly small KL term, thus effectively falling back to a conventional language model. Bowman et al. (2016) have proposed to use input dropout and KL term annealing to encourage their model to encode meaningful representations into the $\mathbf{z}$ vector. We found that these techniques also help our model to achieve solutions with non-zero KL term.

KL term annealing can be viewed as a gradual transition from conventional deterministic Autoencoder to a full VAE. In this work we use linear annealing from 0 to 1. We have experimented with other schedules but did not find them to have a significant impact on the final result. As long as the KL term weight starts to grow sufficiently slowly, the exact shape and speed of its growth does not seem to affect the overall result. We have found the following heuristic to work well: we first run a model with KL weight fixed to 0 to find the number of iterations it needs to converge. We then configure the annealing schedule to start after the unregularized model has converged and last for no less than 20% of that amount.

While helping to regularize the latent vector, input dropout tends to slow down convergence. We propose an alternative technique to encourage the model to compress information into the latent vector: in addition to the reconstruction cost computed on the outputs of the recurrent language model, we also add an auxiliary reconstruction term computed from the activations of the last deconvolutional layer:

$$\begin{aligned} J_{aux} &= -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log\ p(\mathbf{x}|\mathbf{z})] \\ &= -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\sum_t \log\ p(x_t|\mathbf{z})]. \end{aligned} \tag{2}$$

Since at this layer the model does not have access to previous output elements it needs to rely on the $\mathbf{z}$ vector to produce a meaningful reconstruction. The final cost minimized by our model is:

$$J_{hybrid} = J_{vae} + \alpha J_{aux} \tag{3}$$

where α is a hyperparameter, J_{aux} is the intermediate reconstruction term and J_{vae} is the bound from Eq (1). Expanding the two terms from Eq (3) gives:

$$\begin{aligned} J_{hybrid} &= KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &-\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\sum_t \log\ p(x_t|\mathbf{z}, x_{<t})] \\ &-\alpha\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\sum_t \log p(x_t|\mathbf{z})]. \end{aligned} \tag{4}$$

The objective function from Eq (4) puts a mild constraint on the latent vector to produce features useful for historyless reconstruction. Since the autoregressive part reuses these features, it also

improves the main reconstruction term. We are thus able to encode information in the latent vector without hurting expressiveness of the decoder.

One can view the objective function in Eq 4 as a joint objective for two VAEs: one only feed-forward, as in Figure 2(a), and the other combining feed-forward and recurrent parts, as in Figures 2(b) and 2(c), that partially share parameters. Since the feed-forward VAE is incapable of producing reasonable reconstructions without making use of the latent vector, the full architecture also gains access to the latent vector through shared parameters. We note that this trick comes at a cost of worse result on the density estimation task, since part of the parameters of the full model are trained to optimize an objective that does not capture all the dependencies that exist in the textual data. However, the gap between purely deterministic LM and our model is small and easily controllable by the α hyperparameter. We refer the reader to Figure 4 for quantitative results regarding the effect of α on the performance of our model on the LM task.

4 Experiments

We use KL term annealing and input dropout when training the LSTM VAE models from Bowman et al. (2016) and KL term annealing and regularized objective function from Eq (3) when training our models. All models were trained with the Adam optimization algorithm (Kingma and Ba, 2014) with decaying learning rate. We use Layer Normalization (Ba et al., 2016) in LSTM layers and Batch Normalization (Ioffe and Szegedy, 2015) in convolutional and deconvolutional layers. To make our results easy to reproduce we have released the source code of all our experiments[1].

Data. Our first task is character-level language generation performed on the standard Penn Treebank dataset (Marcus et al., 1993). One of the goals is to test the ability of the models to successfully learn the representations of long sequences. For training, fixed-size data samples are selected from random positions in the standard training and validation sets.

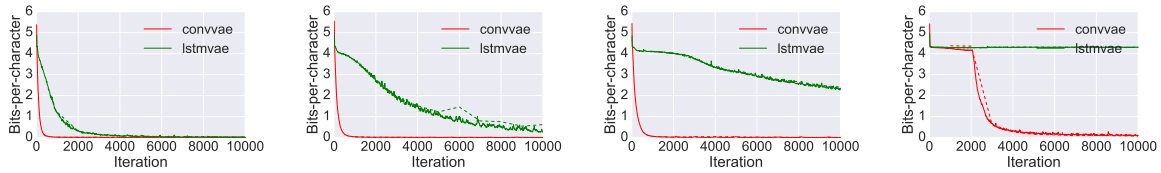
4.1 Comparison with LSTM VAE

Historyless decoding. We start with an experiment where the decoder is forced to ignore the history and has to rely fully on the latent vector. By conditioning the decoder only on the latent vector $\mathbf{z}$ we can directly compare the expressiveness of the compared models. For the LSTM VAE model historyless decoding is achieved by using the dropout on the input elements with the dropout rate equal to 1 so that its decoder is only conditioned on the $\mathbf{z}$ vector and, implicitly, on the number tokens generated so far. We compare it to our fully-feedforward model without the recurrent layer in the decoder (Figure 2(a)). Both networks are parametrized to have comparable number of parameters.

To test how well both models can cope with the stochasticity of the latent vectors, we minimize only the reconstruction term from Eq. (1). This is equivalent to a pure autoencoder setting with stochastic internal representation and no regularization of the latent space. This experiment corresponds to an initial stage of training with KL term annealing when its weight is set to 0. We pursue two goals with this experiment: firstly, we investigate how do the two alternative encoders behave in the beginning of training and establish a lower bound on the quality of the reconstructions. Secondly, we attempt to put the Bit Back coding argument from Chen et al. (2016) in context. The authors assume the encoder to be powerful enough to produce a good representation of the data. One interpretation of this argument applied to textual data is that factorizing the joint probability as $p(\mathbf{x}) = \prod_t p(x_t|x_{<t})$ provides the model with a sufficiently powerful decoder that does not need the latent variables. However, our experimental results suggest that LSTM encoder may not be a sufficiently expressive encoder for VAEs for textual data, potentially making the argument inapplicable.

The results are presented in Figure 3. Note that when the length of input samples reaches 30 characters, the historyless LSTM autoencoder fails to fit the data well, while the convolutional architecture converges almost instantaneously. The results appear even worse for LSTMs on sequences of 50 characters. To make sure that this effect is not caused by optimization difficulties, i.e. exploding gradients (Pascanu et al., 2013), we have searched over learning rates, gradient clipping thresholds and sizes of LSTM layers but were only able to get results comparable to those shown in Figure 3. Note that LSTM networks make use of Layer Nor-

[1] https://github.com/stas-semeniuta/textvae



(a) 10 characters (b) 20 characters (c) 30 characters (d) 50 characters

Figure 3: Training curves of LSTM autoencoder and our model on samples of different length. Solid and dashed lines show training and validation curves respectively. Note that the model exhibits little to no overfitting since the validation curve follows the training one almost perfectly.

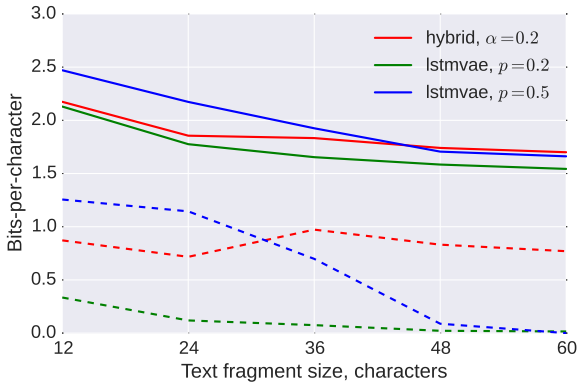


Figure 4: The full cost (solid lines) and its KL component (dashed lines) in bits-per-character of our Hybrid model trained with 0.2 α hyper-parameter vs. LSTM based VAE trained with 0.2 and 0.5 input dropout, measured on validation partition.

malization (Ba et al., 2016) which has been shown to make training of such networks easier. These results suggest that our model is easier to train than the LSTM-based model, especially for modeling longer pieces of text. Additionally, our model is computationally faster by a factor of roughly two, since we run only one recurrent network per sample and time complexity of the convolutional part is negligible in comparison.

Decoding with history. We now move to a case where the decoder is conditioned on both the latent vector and previous output elements. In these experiments we pursue two goals: firstly, we verify whether the results obtained on the historyless decoding task also generalize to a less restricted case. Secondly, we study how well the models cope with stochasticity introduced by the latent variables. Note that we do not attempt to improve the state-of-the-art result on the Language Modeling task but instead focus on providing an approach capable of generating long and diverse sequences. We experiment on the task to obtain a detailed picture of how are our model and LSTM VAE affected by various choices and compare the two models, focusing on how effective is the encoder at producing meaningful latent vector. However, we note that our model performs fairly well on the LM task and is only slightly worse than purely deterministic Language Model, trained in the same environment, and is comparable to the one of Bowman et al. (2016) in this regard.

We fix input dropout rates at 0.2 and 0.5 for LSTM VAE and use auxiliary reconstruction loss (Section 3.4) with 0.2 weight in our Hybrid model. The bits-per-character scores on differently sized text samples are presented in Figure 4. As discussed in Section 3.1, the KL term value indicates how much information the network stores in the latent vector. We observe that the amount of information stored in the latent vector by our model and the LSTM VAE is comparable when we train on short samples and largely depends on hyper-parameters α and p. When the length of a text fragment increases, LSTM VAE is able to put less information into the latent vector (i.e., the KL component is small) and for texts longer than 48 characters, the KL term drops to almost zero while for our model the ratio between KL and reconstruction terms stays roughly constant. This suggests that our model is better at encoding latent representations of long texts since the amount of information in the latent vector does not decrease as the length of a text fragment grows. In contrast, there is a steady decline of the KL term of the LSTM VAE model. This result is consistent with our findings from the historyless decoding experiment. Note that in both of these experiments the LSTM VAE model fails to produce meaningful latent vectors with inputs over 50 characters long. This further suggests that our Hybrid model en-

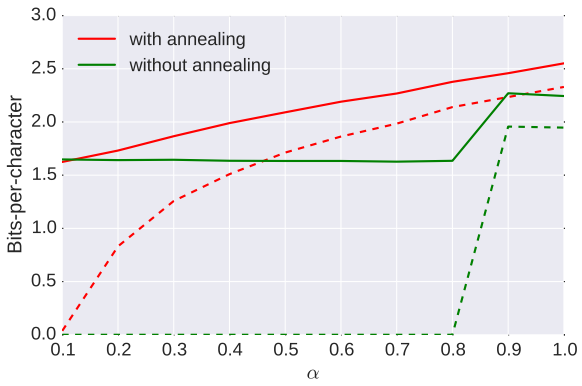


Figure 5: The full cost (solid line) and the KL component (dashed line) of our Hybrid model with LSTM decoder trained with various α, with and without KL term weight annealing, measured on the validation partition.

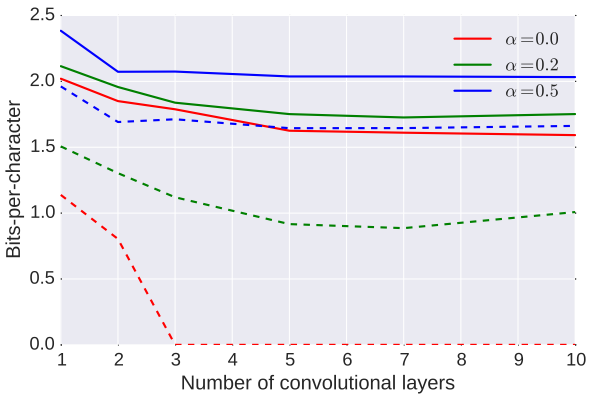


Figure 6: The full cost (solid line) and the KL component (dashed line) of our Hybrid model with ByteNet decoder trained with various number of convolutional layers, measured on the validation partition

codes long texts better than the LSTM VAE.

4.2 Controlling the KL term

We study the effect of various training techniques that help control the KL term which is crucial for training a generative VAE model.

Aux cost weight. First, we provide a detailed view of how optimization tricks discussed in Section 3.4 affect the performance of our Hybrid model. Figure 5 presents results of our model trained with different values of α from Eq. (3). Note that the inclusion of the auxiliary reconstruction loss slightly harms the bound on the likelihood of the data but helps the model to rely more on the latent vector as α grows. A similar effect on model's bound was observed by Bowman et al. (2016): increased input dropout rates force their model to put more information into the $\mathbf{z}$ vector but at the cost of increased final loss values. This is a trade-off that allows for sampling outputs in the VAE framework. Note that our model can find a solution with non-trivial latent vectors when trained with the full VAE loss provided that the α hyper-parameter is large enough. Combining it with KL term annealing helps to find non-zero KL term solutions at smaller α values.

Receptive field. The goal of this experiment is to study the relationship between the KL term values and the expressiveness of the decoder. Without KL term annealing and input dropout, the RNN decoder in LSTM VAE tends to completely ignore information stored in the latent vector and essentially falls back to an RNN language model. To have a full control over the receptive field size of the recurrent component in our decoder, we experiment with masked convolutions (Figure 2(c)), which is similar to the decoder in ByteNet model from Kalchbrenner et al. (2016). We fix the size of the convolutional kernels to 2 and do not use dilated convolutions and skip connections as in the original ByteNet.

The resulting receptive field size of the recurrent layer in our decoder is equal to $N + 1$ characters, where N is the number of convolutional layers. We vary the number of layers to find the amount of preceding characters that our model can consume without collapsing the KL term to zero.

Results of these experiments are presented in Figure 6. Interestingly, with the receptive field size larger than 3 and without the auxiliary reconstruction term from Eq. (3) ($\alpha = 0$) the KL term collapses to zero and the model falls back to a pure language model. This suggests that the training signal received from the previous characters is much stronger than that from the input to be reconstructed. Using the auxiliary reconstruction term, however, helps to find solutions with non-zero KL term component irrespective of receptive field size. Note that increasing the value of α results in stronger values of KL component. This is consistent with the results obtained with LSTM decoder in Figure 5.

@userid @userid @userid @userid @userid ...
I want to see you so much @userid #FollowMeCam ...
@userid @userid @userid @userid @userid ...
Why do I start the day today?
@userid thanks for the follow back
no matter what I'm doing with my friends they are so cute
@userid Hello How are you doing
I wanna go to the UK tomorrow!! #feelinggood #selfie #instago
@userid @userid I'll come to the same time and it was a good day too xx

Table 1: Random sample tweets generated by LSTM VAE (top) and our Hybrid model (bottom).

	Rec	KL
LSTM, $p = 0.2$	67.4	1.0
LSTM, $p = 0.5$	77.1	2.1
LSTM, $p = 0.8$	93.7	3.8
Hybrid, $\alpha = 0.2$	58.5	12.5

Table 2: Breakdown into KL and reconstruction terms for char-level tweet generation. p refers to input dropout rate.

4.3 Generating Tweets

In this section we present qualitative results on the task of generating tweets.

Data. We use 1M tweets[2] to train our model and test it on a held out dataset of 10k samples. We minimally preprocess tweets by only replacing user ids and urls with "@userid" and "url".

Setup. We use 5 convolutional layers with the ReLU non-linearity, kernel size 3 and stride 2 in the encoder. The number of feature maps is [128, 256, 512, 512, 512] for each layer respectively. The decoder is configured equivalently but with the amount of feature maps decreasing in each consecutive layer. The top layer is an LSTM with 1000 units. We have not observed significant overfitting. The baseline LSTM VAE model contained two distinct LSTMs both with 1000 cells. The models have comparable number of parameters: 10.5M for the LSTM VAE model and 10.8M for our hybrid model.

Results. Both VAE models are trained on the character-level generation. The breakdown of total cost into KL and reconstruction terms is given in Table 2. Note that while the total cost values are comparable, our model puts more information into the latent vector, further supporting our observations from Section 4.1. This is reflected in the random samples from both models, presented in Table 1. We perform greedy decoding during generation so any variation in samples is only due to the latent vector. LSTM VAE produces very limited range of tweets and tends to repeat "@userid" sequence, while our model produces much more diverse samples.

[2] a random sample collected using the Twitter API

5 Conclusions

We have introduced a novel generative model of natural texts based on the VAE framework. Its core components are a convolutional encoder and a deconvolutional decoder combined with a recurrent layer. We have shown that the feed-forward part of our model architecture makes it easier to train a VAE and avoid the problem of KL-term collapsing to zero, where the decoder falls back to a standard language model thus inhibiting the sampling ability of VAE. Additionally, we propose an efficient way to encourage the model to rely on the latent vector by introducing an additional cost term in the training objective. We observe that it works well on long sequences which is hard to achieve with purely RNN-based VAEs using the previously proposed tricks such as KL-term annealing and input dropout. Finally, we have extensively evaluated the trade-off between the KL-term and the reconstruction loss. In particular, we investigated the effect of the receptive field size on the ability of the model to respect the latent vector which is crucial for being able to generate realistic and diverse samples. In future work we plan to apply our VAE model to semi-supervised NLP tasks and experiment with conditioning generation on text attributes such as sentiment and writing style.

Acknowledgments

We thank Enrique Alfonseca, Katja Filippova, Sylvain Gelly and Jason Lee for their useful feedback while preparing this paper. This project has received funding from the European Union's Framework Programme for Research and Innovation HORIZON 2020 (2014-2020) under the Marie Skodowska-Curie Agreement No. 641805. Stanislau Semeniuta thanks the support from Pattern Recognition Company GmbH. We thank the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

References

Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Philip Bachman. 2016. An architecture for deep, hierarchical generative models. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *NIPS*, pages 4826–4834.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. 2017. From optimal transport to generative modeling: the vegan cookbook. *CoRR*, abs/1705.07642.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *CONLL*, pages 10–21.

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. Variational lossy autoencoder. *CoRR*, abs/1611.02731.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083.

Marco Fraccaro, Søren Kaae Sø nderby, Ulrich Paquet, and Ole Winther. 2016. Sequential neural models with stochastic layers. In *NIPS*, pages 2199–2207.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vázquez, and Aaron C. Courville. 2016. Pixelvae: A latent variable model for natural images. *CoRR*, abs/1611.05013.

David Ha, Andrew M. Dai, and Quoc V. Le. 2016. Hypernetworks. *CoRR*, abs/1609.09106.

Geoffrey E. Hinton and Drew van Camp. 1993. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993.*, pages 5–13.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, pages 1735–1780.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Controllable text generation. *CoRR*, abs/1703.00955.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *CoRR*, abs/1610.10099.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Diederik P. Kingma, Tim Salimans, and Max Welling. 2016. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934.

Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

Hugo Larochelle and Iain Murray. 2011. The neural autoregressive distribution estimator. In *AISTATS*, pages 29–37.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. 2015. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Lars M. Mescheder, Sebastian Nowozin, and Andreas Geiger. 2017. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *CoRR*, abs/1701.04722.

Yishu Miao, Lei Yu, and Phil Blunsom. 2015. Neural variational inference for text processing. *CoRR*, abs/1511.06038.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. 2015. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016. Pixel recurrent neural networks. *CoRR*, abs/1601.06759.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434.

Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. 2014. Techniques for learning binary stochastic feedforward neural networks. *CoRR*, abs/1406.2989.

Danilo Jimenez Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1530–1538.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2016. A hierarchical latent variable encoder-decoder model for generating dialogues. *CoRR*, abs/1605.06069.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. 2016. Ladder variational autoencoders. *CoRR*, abs/1602.02282.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2014. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. 2015. Attribute2image: Conditional image generation from visual attributes. *CoRR*, abs/1512.00570.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved variational autoencoders for text modeling using dilated convolutions. *CoRR*, abs/1702.08139.

Serena Yeung, Anitha Kannan, Yann Dauphin, and Li Fei-Fei. 2016. Epitomic variational autoencoder. *In submission to ICLR 2017*.

Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Robert Fergus. 2010. Deconvolutional networks. In *CVPR*, pages 2528–2535.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2016. Variational neural machine translation. *CoRR*, abs/1605.07869.

Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. 2017. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *CoRR*, abs/1703.10960.

Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578.