Continuous Language Generative Flow

Zineng Tang **Shiyue Zhang** **Hyounghun Kim** **Mohit Bansal**
UNC Chapel Hill
{terran, shiyue, hyounghk, mbansal}@cs.unc.edu

Abstract

Recent years have witnessed various types of generative models for natural language generation (NLG), especially RNNs or transformer based sequence-to-sequence models, as well as variational autoencoder (VAE) and generative adversarial network (GAN) based models. However, flow-based generative models, which achieve strong performance in image generation due to their invertibility and exact density estimation properties, have been less explored for NLG. In this paper, we propose a flow-based language generation model by adapting previous flow generative models to language generation via continuous input embeddings, adapted affine coupling structures, and a novel architecture for autoregressive text generation. We also apply our framework to Sequence-to-Sequence generation, including text- and video-based Question Generation (QG) and Neural Machine Translation (NMT), and data augmentation for Question Answering (QA). We use our language flow model to provide extra input features for QG and NMT, which achieves improvements over the strong QG baselines on SQuAD and TVQA and NMT baseline on WMT16. We also augment QA data with new context by injecting noise to the latent features of the language flow and show this augmentation leads to a large performance improvement from strong baselines on SQuAD and TVQA.[1]

1 Introduction

Several generative models have been proposed for language generation, including sequence-to-sequence models based on RNNs (Luong et al., 2015) and transformers (Vaswani et al., 2017), as well as variational autoencoders (VAEs) to generate diverse texts (Bowman et al., 2016; Jain et al., 2017), plus generative adversarial networks (GANs) (Yu et al., 2017) to improve intended semantic fidelity. Another line of the generative model, normalizing flow (Rezende and Mohamed, 2015), is widely explored in computer vision and representation learning but less explored for NLG tasks. Flow models have been shown to be capable of improving probability density estimation, including variational inference (Rezende and Mohamed, 2015) and exact density estimation (Dinh et al., 2015). Generative flow is one type of flow model and first proposed by Dinh et al. (2015, 2017); Kingma and Dhariwal (2018). Taking advantage of its invertible structure, it can perform an exact density estimation of the input distribution. Thus, during generation, we can sample from its latent space and then generate new examples through its invertible decoder. Generative flow shows strong performance on image generation, attribute manipulation, and latent space inference (Kingma and Dhariwal, 2018). Considering these successful applications, we conjecture that the flow model should also have strong potential to be adapted for language generation tasks. Therefore, in this paper, we introduce a continuous language generative flow model that can deal with discrete language data in continuous latent space. We propose two variants, the non-autoregressive and autoregressive models, and show that they both can perform well on density estimation tasks.

We follow the architecture of one previous generative flow model, Glow (Kingma and Dhariwal, 2018), but make adaptions for language generation tasks. We first employ GloVe word embeddings (Pennington et al., 2014) to map the discrete token sequence to a continuous embedding matrix. Furthermore, we utilize two components: time-dimension permutation and affine coupling with RNN or Transformer non-linearity functions, which allow interaction between words in a se-

[1]Our code and models are available at: https://github.com/zinengtang/ContinuousFlowNLG

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 4609–4622
August 1–6, 2021.

quence and better contextualizes language semantics. Overall, these proposed components help generate texts in a non-autoregressive manner.

However, even though the non-autoregressive model has attracted a lot of research attention because of its fast generation speed, it still hardly surpasses the generation quality of autoregressive models (Ren et al., 2020). Therefore, to make our language flow model learn language generation in a stronger autoregressive manner, we change the flow model's affine coupling and permutation to a uni-directional structure, i.e., each timestep can only attend to previous timesteps. In this way, we enable our model to perform text generation autoregressively.

Some recent works have developed density estimation models targeted on character-level discrete data (DiscreteFlow (Tran et al., 2019)) and explored using the flow architecture as an extra data encoder that provides latent features to support non-autoregressive text generation (FlowSeq (Ma et al., 2019)). While our work shares some similar characteristics, we explore different directions: (1) DiscreteFlow develops a modulus calculation method to process discrete data. Instead, we use word embedding to transform the discrete input tokens to continuous features, which is simple yet effective. (2) FlowSeq essentially leverages the flow architecture in a typical encoder-decoder model to support non-autoregressive generation, whereas our models follow the standard generative flow framework and can directly generate texts via their invertible structure in both non-autoregressive or autoregressive manner. (3) Autoregressive flows were previously developed (Papamakarios et al., 2017; Huang et al., 2018) for stronger density estimation ability. However, the autoregressive language flow model we develop here aims for better text generation quality. For this, our model is autoregressive in both the forward stage (encoding an input to a latent feature) and inverse stage (decoding the latent feature to the input) with an uni-directional (i.e., the left-to-right direction) structure,

We evaluate the density estimation ability of our language flow models as well as their effectiveness for three downstream tasks: (1) sequence-to-sequence (Seq-to-Seq) generation that includes question generation (QG) and neural machine translation (NMT) and (2) data augmentation for Question Answering (QA). We test QG and QA data augmentation on two large-scale QA datasets: (a) SQuAD (Rajpurkar et al., 2016), a widely explored textual QA and QG dataset and (b) TVQA (Lei et al., 2018), a large-scale multimodal video-dialogue QA task. We test machine translation on WMT16 (Cettolo et al., 2012), a commonly used NMT dataset.

For density estimation, we compare the negative likelihoods of our models against a baseline LSTM model. For QG, we use the non-autoregressive flow model to provide extra input features for a standard encoder-decoder text generation model. We show that it can significantly improve a baseline QG model for both SQuAD and TVQA on both automatic and human evaluation metrics. Aided by our flow model, we achieve strong improvements over a transformer baseline in the neural machine translation experiment. In addition to improving language generation quality, we also use the proposed autoregressive flow model for data augmentation. For this, we focus on generating diverse textual contexts for QA tasks. In particular, we inject noise into the latent features of our flow models (encoded from ground-truth contexts) and then generate new contexts from the noise-injected features. Experiments show that the generated contexts can be either a varied expression of the same subject or paraphrasing the original context, but, mostly keep the answerability of the original question (see examples in Table 3). Combined with data augmentation strategies (data filtering and training schema), we achieve statistically significant improvements on both SQuAD and TVQA over strong baselines.

Overall, we have two contributions: (1) we propose two continuous language generative flow model variants that have better density estimation abilities than an LSTM baseline model, and can perform non-autoregressive and autoregressive generation respectively; (2) Our language flow model largely improves QG, NMT, and data augmentation for QA tasks.

2 Language Generative Flow

In this section, we first review the generative flow model proposed in previous works (Dinh et al., 2015; Kingma and Dhariwal, 2018). Then, following it, we propose two variants of our continuous language generative flow model.

2.1 Background: Generative Flow

Flow-based generative models transform simple latent distributions, $p(\mathbf{z})$, into a complex data dis-

tribution (language text in our case), $p(\mathbf{x})$, through a chain of invertible transformations.

We first designate a true data distribution $p(\mathbf{x})$ and a model $p_{\boldsymbol{\theta}}(\mathbf{x})$ with parameters $\boldsymbol{\theta}$ to parameterize the true distribution $p(\mathbf{x})$. The latent space inference is then defined as:

$$\mathbf{x}_i \sim p(\mathbf{x}) \tag{1}$$

$$\mathbf{z}_i = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i) \tag{2}$$

where $\mathbf{x}_i$ is a data point from the true data distribution and $\mathbf{z}_i$ the latent features. This encoding x to z procedure is usually referred as the *forward stage*.

The transformation $\mathbf{f}_{\boldsymbol{\theta}}$ is designed to be invertible and bijective. In previous flow-based generative models (Dinh et al., 2015, 2017; Kingma and Dhariwal, 2018), the generative process (or referred as the *inverse stage*) is defined as:

$$\mathbf{z}_i \sim p_{\boldsymbol{\theta}}(\mathbf{z}) \tag{3}$$

$$\mathbf{x}_i = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{z}_i) = \mathbf{f}_{\boldsymbol{\theta}}^{-1}(\mathbf{z}_i) \tag{4}$$

where $\mathbf{z}_i$ is a sample from the latent space distribution, such as a standard Gaussian distribution.

The flow mapping $\mathbf{f}_{\boldsymbol{\theta}}$ is composed of a chain of transformations: $\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ \cdots \circ \mathbf{f}_K$ with each representing one *flow step*. Then, the log-likelihood can be written as:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \sum_{j=1}^{K} \log \left| \det \left(\frac{d\mathbf{h}_j}{d\mathbf{h}_{j-1}} \right) \right| \tag{5}$$

where $\mathbf{h}_j$ is the output of each flow step. The value $\log|\det(d\mathbf{h}_j/d\mathbf{h}_{j-1})|$ is namely the log-determinant: the log of the absolute value of the determinant of the Jacobian matrix $(d\mathbf{h}_j/d\mathbf{h}_{j-1})$. This value is the change in log-density from $\mathbf{h}_{j-1}$ to $\mathbf{h}_j$ under transformation $\mathbf{f}_j$. This equation is namely the change of variable formula.

The objective for density estimation is formulated as:

$$\mathcal{L}(\mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} -\log p_{\boldsymbol{\theta}}\left(\tilde{\mathbf{x}}_i\right) - M \log d \tag{6}$$

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{u} \tag{7}$$

where $\mathbf{u}$ is usually sampled from a Gaussian distribution, N the number of samples in a batch, and d $(= 128)$ the discretization level of the data and M the dimension of $\mathbf{x}_i$.[2]

[2]The change of variable formula, Eq.5, treats the data space as unbounded. However, the data we use is usually within range -1.0 to 1.0 and parameter d (the discretization) can reduce the impact of boundary effects according to Dinh et al. (2017).

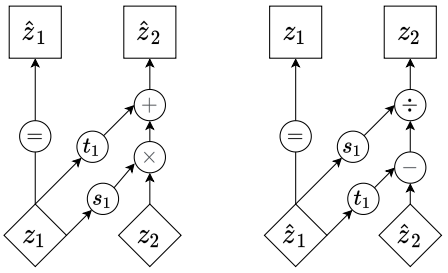

Figure 1: Affine Coupling illustration. The inputs $\mathbf{z}_0$ is split into two halves into z_1 and z_2 along hidden dimension and obtain the outputs $\hat{z}_1$ and $\hat{z}_2$ which will be concatenated. And it is similar for the reverse stage. Note that the $\times$ operation is element-wise product.

Each *flow step* in the generative flow model includes three parts: Normalization, Permutation, and Affine coupling.

(1) **Normalization** is designed to scale each output to stabilize training. We follow Glow (Kingma and Dhariwal, 2018) to use actnorm.

(2) **Permutation** makes sure after multiple flow steps, each channel can sufficiently affect other dimensions. The Glow model (Kingma and Dhariwal, 2018) proposes to use a (trainable) invertible 1×1 convolution. It is essentially a flexible generalization of a permutation operation. We follow Glow and also use its LU decomposition to reduce determinant computation cost. Different from all previous work, we apply 1×1 convolution on the time dimension rather than the hidden dimension. This is because language data is sequential and temporal. This change is crucial to the proposed flow model's performance, which will be shown in ablation studies (Table 4).

(3) **Affine coupling** is designed to incorporate complex nonlinear mapping but still keep invertibility (see Figure 1).

$$\mathbf{z}_1, \mathbf{z}_2 = \text{Split}(\mathbf{z}_0, \text{dim} : \text{time}) \tag{8}$$

$$\mathbf{s}, \mathbf{t} = \text{Split}(\text{NN}(\mathbf{z}_1), \text{dim} : \text{hidden}) \tag{9}$$

$$\hat{\mathbf{z}}_2 = \sigma(\mathbf{s} + \alpha) \odot (\mathbf{t} + \mathbf{z}_2) \tag{10}$$

where NN refers to nonlinear function, σ is sigmoid activation. α is a hyperparameter that prevents small value (around 0) from resulting in large negative value by log. Note that, in the first equation, Glow (Kingma and Dhariwal, 2018) splits along the hidden dimension. However, we split along time dimension (first introduced in FlowSeq (Ma et al., 2019)) which has the same motivation as the permutation module.

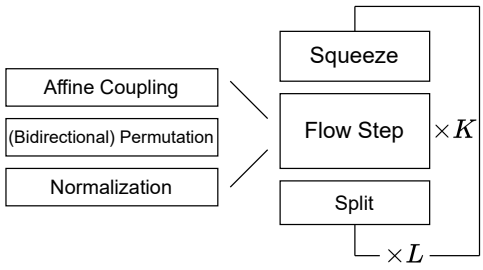

Figure 2: Non-autoregressive Language Flow model with Multi-Scale architecture.

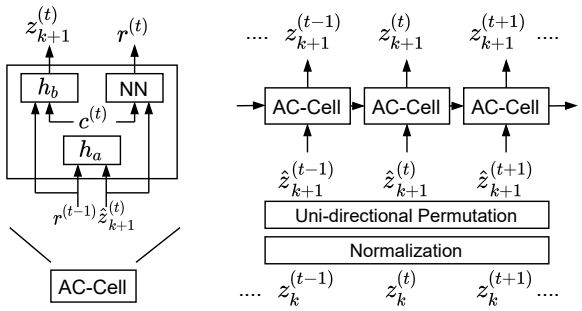

Figure 3: Autoregressive Language Generative Flow model. The whole autoregressive flow model contains multiple K steps. This figure illustrates one flow step from z_k to z_{k+1}.

2.2 Non-Autoregressive Language Flow

We first present our non-autoregressive language flow which is based on the architecture introduced above. Besides the permutation/affine coupling structures changes introduced above, we use RNNs or Transformer as the nonlinear mapping, propose to use continuous input embedding, and introduce multi-scale architecture.

Affine Coupling. We use a multihead self-attention module in transformer (Vaswani et al., 2017) or alternatively RNNs (a one-layer bidirectional LSTM (Schuster and Paliwal, 1997)) in the coupling layer by replacing the non-linear mapping of affine coupling, NN (see Eq.9).

Continuous Input Embedding. The language flow model we propose operates on continuous inputs, which means the inputs are not discrete tokens but continuous word embeddings. We implement it through GloVe embeddings (Pennington et al., 2014). Therefore, the density estimation is performed for the distribution $p(\mathbf{x})$, where $\mathbf{x}$ is the word embeddings of language tokens. Note that the word embeddings are frozen. In the inverse stage, we compute the cosine similarity between the embedding matrix and decoder output as the token generation probability distribution, so that all tokens can be generated in parallel, i.e., non-autoregressively.

Multi-Scale Architecture. Following Dinh et al. (2017), we use a multi-scale architecture (see Figure 2) that contains multiple blocks while each block containing several flow steps. In our work, we denote the number of flow steps as K, and the number of blocks as L that each contains K flow steps. We denote the input shape as (batch size b, sequence length s, hidden dimension h). At the start of each block, the tensor is reshaped from (b, s, h) to $(b, \frac{s}{2}, 2h)$, so the model can capture more local features; and at the end of each block (except the last block), the latent feature is split into halves via channel dimension with one as the output, $\mathbf{z}_l$, and the other as the input of the next block. If we have 3 blocks, we will have three latent outputs, $\mathbf{z}_l$. Past works (Dinh et al., 2017; Kingma and Dhariwal, 2018; Ma et al., 2019) reshape in this manner for all blocks. However, we do not reshape in the first block but apply the same for the following blocks, which allows the model to better process the original input text with intact sentence structure.

2.3 Autoregressive Language Flow

The model we developed in the previous subsection can properly operate on continuous word embeddings, have exact density estimation, and perform non-autoregressive generation, however, it lacks the autoregressive structure that is commonly used for text generation. Previous works have shown autoregressive generation usually performs better than non-autoregressive generation (Ren et al., 2020). Thus, we develop an autoregressive model that can generate text from left to right in the inverse stage. To achieve this, we change affine coupling and permutation in the flow step to be uni-directional, i.e., each timestep can only attend to timesteps that precede it. However, we have to remove the multi-scale architecture to fulfill the autoregressive requirement. See sample outputs in Table 1 for comparison to those from the non-autoregressive model.

Uni-directional Permutation. Since the permutation in each flow step designed in our non-autoregressive flow model is bidirectional, we mask the 1×1 convolution to a lower triangular matrix. Therefore, each token can only attend to previous tokens in the permutation, i.e., uni-directional permutation.

Non-Autoregressive Samples	Autoregressive Samples
what does house way when when that little he when the even?	what does wilson probably do after drawing?
what did richard know when he she else there the	what did jamie want after charlie forget her immediately
what does nelson going when he she when he what that to?	what is brian aware
what did richard know when he she else there the	what did caleb say after he went out?
what does nelson going when he she when he what that to?	what does phoebe think?

Table 1: Data samples generated by our flow models. We sample from a Gaussian distribution and generate questions by our non-autoregressive or autoregressive flow decoders. Models are trained on TVQA questions.

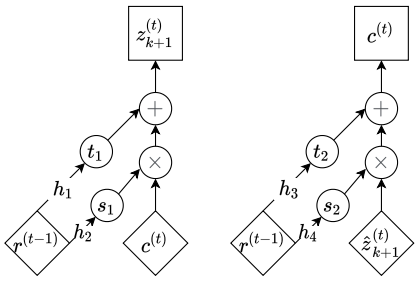

Figure 4: The two figures illustrate the h_a and h_b functions of the autoregressive affine coupling in one flow step.

Uni-directional Affine Coupling. We then introduce an autoregressive version of affine coupling, shown by the AC-cell in Figure 3. For each flow step, we denote the input sequence as $\hat{\mathbf{z}}_{k+1}^{(0):(T)} = [\hat{\mathbf{z}}_{k+1}^{(0)}, ..., \hat{\mathbf{z}}_{k+1}^{(T)}]$, and then the autoregressive coupling is defined as:

$$\mathbf{r}^{(t-1)} = \text{NN}([\mathbf{c}^{(t-1)}; \mathbf{z}_{k+1}^{(t-1)}]) \quad (11)$$

$$\mathbf{c}^{(t)} = \mathbf{h}_a(\mathbf{r}^{(t-1)}, \hat{\mathbf{z}}_{k+1}^{(t)}) \quad (12)$$

$$\mathbf{z}_{k+1}^{(t)} = \mathbf{h}_b(\mathbf{r}^{(t-1)}, \mathbf{c}^{(t)}) \quad (13)$$

We recurrently obtain the outputs, $[\mathbf{z}_{k+1}^{(1)}, ..., \mathbf{z}_{k+1}^{(T)}]$. Note that $\mathbf{z}_{k+1}^{(0)} = \hat{\mathbf{z}}_{k+1}^{(0)}$, so the computation starts from $\mathbf{z}_{k+1}^{(1)}$. When computing $\mathbf{z}_{k+1}^{(1)}$, we cannot get $\mathbf{c}^{(0)}$, so we set it to be zero. $\mathbf{h}_a$ and $\mathbf{h}_a$ are both affine coupling structured, as shown in Figure 4. NN is either RNN or transformer.

In the inverse stage, to obtain $\hat{\mathbf{z}}_{k+1}$, we start from $\hat{\mathbf{z}}_{k+1}^{(0)} = \mathbf{z}_{k+1}^{(0)}$ and $\mathbf{c}^{(0)}$:

$$\mathbf{r}^{(t-1)} = \text{NN}([\mathbf{c}^{(t-1)}; \hat{\mathbf{z}}_{k+1}^{(t-1)}]) \quad (14)$$

$$\mathbf{c}^{(t)} = \mathbf{h}_b^{-1}(\mathbf{r}^{(t-1)}, \mathbf{z}_{k+1}^{(t)}) \quad (15)$$

$$\hat{\mathbf{z}}_{k+1}^{(t)} = \mathbf{h}_a^{-1}(\mathbf{r}^{(t-1)}, \mathbf{c}^{(t)}) \quad (16)$$

Since both decoded tokens $\mathbf{z}^{(t)}$ and context $\mathbf{c}^{(t)}$ only depend on previous tokens $\mathbf{z}^{(0):(t-1)}$, we can perform autoregressive decoding and beam search with cosine similarity as the probability distribution of output tokens.

Autoregressive Flow Step. The changes of affine coupling and permutation to uni-directional allow the flow step to be autoregressive. And the whole autoregressive flow model will contain K such flow steps. At each flow step, the log-determinant is the summation of the log-determinant of all time steps:

$$\log p(\mathbf{z}_{k+1}) = \sum_t \log p(\mathbf{z}_{k+1}^{(t)}) \quad (17)$$

$$= \sum_t \log p(\mathbf{z}_k^{(t)}) + \log \left| \det \left(\frac{d\mathbf{z}_{k+1}^{(t)}}{d\mathbf{z}_k^{(t)}} \right) \right| \quad (18)$$

3 Language Generation with Flow

We next apply our flow model to several downstream tasks. Despite the flow's rigid model structure, it has a strong potential in density estimation due to its complex transformation of inputs into a continuous latent space. We aim to use this property to improve standard encoder-decoder text generation models. Moreover, as the flow model has a strong ability in generating diverse text, we show that it has the capability for data augmentation to improve QA tasks.

3.1 Downstream Datasets

SQuAD. SQuAD is a textual question answering dataset containing 100,000+ questions/answers with corresponding short articles as context. We use it to evaluate both question generation and data augmentation (by generating new articles) for question answering.

TVQA. TVQA is a large-scale video QA dataset based on 6 TV shows. It consists of 152,545 QA pairs from 21,793 video clips with subtitle text. We use it to evaluate both question generation and data augmentation (by generating new subtitles) for question answering.

Sample Ratio	Sample Pair 1	Sample Pair 2
Sentence1	where did sheldon and beverley go after they came up stairs?	what did rachel do before chandler said something wasn't true?
0.6	where did sheldon and joey go after they came up?	what did rachel do before chandler said something wasn't out?
0.5	where was rachel when joey said after , guys around huh?	what did house do before chandler when she was walking out?
0.4	where was rachel when joey said no guys around huh?	what did house do to chandler when she was walking out?
Sentence2	where was rachel when joey said, no guys around, huh?	what did house say to sam when she was walking out the door?

Table 2: Interpolation results. We sample two pairs of questions from TVQA. For each pair, we perform interpolation of their latent vectors learned by our autoregressive flow model with different mix ratios (0.4, 0.5, 0.6)

WMT16 (RO-EN). WMT16 (RO-EN) is a machine translation dataset between English and Romanian with around 610k sentence pairs. We use it for our machine translation experiment and only test for the Romanian to English direction.

3.2 Seq-to-Seq Generation with Flow

Similar to FlowSeq (Ma et al., 2019), we use flow as an extra module on top of a typical encoder-decoder language generation model and test on Question Generation (QG) and neural machine translation (NMT). As the flow model has the ability for exact density estimation, it provides the exact density components of context information and we assume that it provides a better hidden representation of context and thus helps with language generation. It can also be viewed as a self-supervised learning method that can provide new features for downstream tasks.

Concretely, while the original QG model[3] is formulated as $\mathbf{u}_i = \mathbf{E}(\mathbf{x}_i), \hat{\mathbf{q}}_i = \mathbf{G}(\mathbf{u}_i)$; the new QG model with flow is formulated as:

$$\mathbf{u}_i = \mathbf{E}(\mathbf{x}_i), \hat{\mathbf{q}}_i = \mathbf{G}(\mathbf{h}_{att}(\mathbf{u}_i, \mathbf{z}_i)) \tag{19}$$

where $\mathbf{E}$ refers to encoder and $\mathbf{G}$ decoder. $\mathbf{z}_i$ refers to latent features of the non-autoregressive flow model. $\mathbf{h}_{att}$ is essentially a MLP with sigmoid activation.

The loss function has two parts:

$$\mathcal{L}_{gen} = \frac{1}{N}\sum_{i=1}^{N} -\log p(\mathbf{q}_i) \tag{20}$$

$$\mathcal{L} = \lambda \mathcal{L}_{nll} + \mathcal{L}_{gen} \tag{21}$$

where $\mathbf{q}_i$ represents the target questions and λ is a hyperparameter for NLL loss (Eq. 6)

[3]We replicate Zhang and Bansal (2019)'s standard encoder-decoder attention QG model with BERT features as input embeddings.

3.3 Context Generation for Data Augmentation

Context Generation. We propose to use flow to generate diverse contexts for data augmentation as both TVQA and SQuAd are question answering tasks with textual context. We generate new context (video subtitles for TVQA; articles for SQuAD) by injecting noise to the hidden vector of the original context, $\mathbf{z}_i$, and reconstructing it to new sentences, $\hat{\mathbf{x}}_i$. Note that, we can also do the same thing for questions, however, we find that changing one word in the question will dramatically change its meaning, so we limit this augmentation to the context and keep the original question unchanged.

The generation process is formulated as:

$$\mathbf{z}_i = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_i) \tag{22}$$

$$\hat{\mathbf{x}}_i = \mathbf{f}_{\boldsymbol{\theta}}^{-1}(\mathbf{z}_i + \mathbf{z}_0) \tag{23}$$

where $\mathbf{f}_{\boldsymbol{\theta}}$ refers to the flow model and $\mathbf{x}_i$ the input text and $\mathbf{z}_i$ the latent space. The transformation is performed by simply sampling a Gaussian noise $\mathbf{z}_0$, add it to $\mathbf{z}_i$, and reconstruct the new context $\hat{\mathbf{x}}_i$ in the reverse stage. In this task, we use the autoregressive flow model as this variant is designed for text generation. We also use the non-autoregressive flow model additionally leveraged by an additional autoregressive decoder, as an alternative approach.

While the standard RNN-based language model does not have an explicit global sentence representation, our flow model is similar to Bowman et al. (2016)'s VAE framework that encodes the sentence into a continuous hidden vector, $p(\mathbf{z}|\mathbf{x})$. And sampling around the hidden vector can naturally be viewed as injecting noise without changing key information. Therefore, we do not aim for paraphrasing the original context because the flow model can reconstruct different information from random noise injection in latent space. Notably, this method has the risk of changing the context's meaning and making the question unanswerable, however, empirically, we find that as long as we

Original	Generated
TVQA Generated Subtitle Example 1: varied expression of the subject.	
A: boy do i feel bad! oh yeah. very bad. B: chandler what are you doing? chandler! oh my god! B: you're smoking again? A: well actually yesterday i was smoking again. today i'm smoking still.	B: oh my god ! why are you doing this again ? A: i don't feel bad . B: why are you still smoking again? A: i was just smoking again after i first started smoking again.
SQuAD Generated Article Example 1: paraphrasing.	
while a computer may be viewed as running one gigantic program stored in its main memory, in some systems it is necessary to give the appearance of running several programs simultaneously. this is achieved by multitasking i.e. having the computer switch rapidly between running each program in turn.	the main memory of the gigantic computer is running the gigantic computer program. in some systems, it is necessary to have the computer switch rapidly between each program achieved by multitasking.

Table 3: Sample context generation results: we show two examples that are filtered as positive. Via a long and a short TVQA example, we show that our model is not entirely paraphrasing the original dialogue but changing content while keeping the central theme unchanged; via a SQuAD example, we show that our model can paraphrase complex semantics.

keep the noise small enough, the generation will be either paraphrases or different expressions of the same subject without affecting the answerability.

Data Filtering. To better utilize the generated data, we design a data filter as filtering out the low-quality generated text is useful in helping improve the data augmentation (Zhang and Bansal, 2019). We use pretrained QA baseline models (see Table 8 Baseline TVQA+ and Table 9 Baseline BERT) to filter out the low-quality context. The generated context will be filtered out if the model performs worse on predicting correct answers when original context is replaced by its generated counterpart.

4 Experimental Setup

We follow Zhang and Bansal (2019) to split the development set of SQuADv1.1 (Rajpurkar et al., 2016) into two halves and show the result on the test split. We generally follow previous work on evaluation metrics. For density estimation, we use negative log-likelihood (NLL) for comparison and bits per dimension to regularize the negative log-likelihood loss, formulated as $\frac{\mathcal{L}}{M \log(2)}$, where M represents the dimension of input. We evaluate QG by BLEU4 (Papineni et al., 2002), Meteor (Lavie and Agarwal, 2007), Rouge-L (Lin, 2004), and Amazon MTurk human evaluation. We use the BLEU score to evaluate NMT. We use accuracy to evaluate the TVQA QA model and EM (exact match) and F1 score to evaluate the SQuAD QA model.

We replicate Zhang and Bansal (2019)'s baseline QG model. We use the STAGE model with GloVe embeddings developed by Lei et al. (2020) as the TVQA QA baseline and use BERT as the SQuAD QA baseline. See appendix A for more experiment/reproducibility details.

Model	TVQA Subtitle	SQuAD Article
Bi-LSTM	-7.31	-1.27
Att-C	0.68	-2.01
RNN-C	0.50	-0.37
Att-S	-8.02	-17.12
RNN-S	-8.35	-17.17
Att-AR	-9.62	-17.12
RNN-AR	-9.63	-17.26

Table 4: The NLL results of flow models and an LSTM baseline on the validation split of TVQA subtitles and test split of SQuAD articles. The difference between C (e.g., Att-C) and S (e.g., Att-S) is whether the affine coupling/permutation is based on channel-dim (C) or time-dim (S). AR means autoregressive architecture. Att- refers to transformer nonlinear mapping, and RNN- refers to RNN nonlinear mapping.

5 Results

5.1 Negative Log-Likelihood Results

First of all, to evaluate the density estimation ability, we compare the negative log-likelihood (NLL, Eq.6)[4] of our different flow models on the context data of SQuAD and TVQA against a baseline model (a 3-layer bidirectional LSTM-RNN model with hidden size 300). As shown in Table 4, the flow model of time-dim coupling/permutation

[4]Note that since our $p(\mathbf{x})$ is over continuous word embeddings, so it is the probability density of a continuous variable which is not bounded by [0,1].

generally outperforms the baseline LSTM model. The flow model of time-dim coupling/permutation largely outperforms the flow model of channel-dim coupling/permutation. We also test our autoregressive model to check its density estimation ability, and we find it performs well and even sometimes slightly better than the non-autoregressive model. Note that we do not claim the autoregressive model is better at density estimation than the non-autoregressive version, instead, we aim to show that it can perform reasonably with the proposed autoregressive adaptation.

5.2 Seq-to-Seq Generation Results

Question Generation. Through the ablation studies shown in Table 5 and Table 6, we demonstrate that the proposed flow-aided QG model significantly improves the QG performance. The statistical significances for all metric improvements (BLEU4, Rouge-L, Meteor) are $p < 0.001$ for both TVQA QG and SQuAD QG.[5] We also conduct a human evaluation. We random sample 200 examples[6], and we present the participants two questions per example generated by two different models and let them judge which question is better in terms of answerability and overall quality. See more human evaluation details in Appendix A.3. We compare our flow model to the pure encoder-decoder baseline as well as the FlowSeq model (Ma et al., 2019) in human evaluation. As shown in the last rows in Table 5 and Table 6, humans favor our model more than the baseline in both tasks, which indicates our flow model indeed provides useful latent features for better generation. Plus, our model also always outperforms FlowSeq. We conjecture that it is because FlowSeq is non-autoregressive whereas our QG model is autoregressive.

Neural Machine Translation. We also test the effectiveness of our approach on a neural machine translation (NMT) task. We first replicate Lee et al. (2018)'s transformer autoregressive model baseline, and then we add our flow architecture on top of it. As shown in Table 7, our proposed flow-aided MT model can improve the machine translation performance over the strong transformer baseline on the WMT16 (Cettolo et al., 2012) Romanian to English translation task. See A.7 for more details. We hope that these promising initial NMT results will also encourage the community to use continuous flow models for other NMT and NLG tasks.

[5]Statistical significance is computed using the bootstrap test (Efron and Tibshirani, 1994).

[6]We exclude those examples where the two models generate identical questions.

Models	BLEU4	Rouge-L	Meteor
FlowSeq	12.19	41.02	18.51
QG baseline	10.68	39.58	17.38
+ Lang-Flow	12.55	41.32	18.68
+ LSTM-Flow	11.48	40.49	17.94

Models	Lang-Flow	Baseline	Tie
Human Eval 1	89	54	57

Models	Lang-Flow	FlowSeq	Tie
Human Eval 2	98	50	52

Table 5: TVQA-QG Evaluation: comparison between FlowSeq (Ma et al., 2019), a BERT QG baseline, Flow aided QG model (Lang-Flow), and simple density estimation model (3-layer LSTM) aided QG baseline model (LSTM-Flow) on TVQA QG validation split.

Models	BLEU4	Rouge-L	Meteor
FlowSeq	14.95	44.83	19.69
QG baseline	18.08	46.68	21.86
+ Lang-Flow	19.21	47.62	22.38
+ LSTM-Flow	18.93	47.27	21.97

Models	Lang-Flow	Baseline	Tie
Human Eval 3	76	63	61

Models	Lang-Flow	FlowSeq	Tie
Human Eval 4	96	69	35

Table 6: SQuAD-QG Evaluation: comparison between FlowSeq (Ma et al., 2019), a BERT QG baseline, Flow aided QG baseline model (Lang-Flow) and simple density estimation model (3-layer LSTM) aided QG baseline model (LSTM-Flow) on the SQuAD-QG test split.

5.3 QA Data Augmentation Results

As shown in Table 8 and Table 9, using the augmented data generated by our Language Flow model (refers to our autoregressive language flow model), we achieve significant performance improvements over strong baselines on both TVQA QA (Lei et al., 2020) ($p < 0.0001$) and SQuAD QA (Rajpurkar et al., 2016) ($p < 0.0005$) for both EM and F1. Furthermore, when we add an LSTM autoregressive decoder to our non-autoregressive encoder (referred to as Language Flow+) and use it to perform data augmentation, we observe even slightly better results. This may indicate the stronger encoding ability of our non-autoregressive model due to its multi-scale architecture. Mean-

Models	BLEU
Transformer Baseline	30.27
+Lang-Flow	30.87

Table 7: MT results on WMT16 RO-EN dev split.

Models	Valid-Accuracy
Baseline TVQA+	69.42
+ Context (Back-Translation)	69.52
+ Context (Paraphrasing)	69.98
+ Context (Language Flow)	70.45
+ Context (Language Flow+)	70.86

Table 8: QA results on TVQA dev split. Language Flow refers to the autoregressive language flow we propose, and Language Flow+ refers to the model with a non-autoregressive flow model encoder plus an LSTM autoregressive decoder.

while, we compare to two other data augmentation techniques: paraphrasing (Niu and Bansal, 2018) and back-translation (Sennrich et al., 2016). Note that for a fair comparison, we apply the same data filter and training schema for all data augmentation methods. It can be seen that both methods perform worse than our Language Flow or Language Flow+ models.

6 Discussion

We show some sample questions generated by our non-autoregressive and autoregressive flow models in Table 1. The autoregressive samples are better organized and grammatically sound, while non-autoregressive generation fails at the latter part of the sentence. It might because the non-autoregressive structure has a weaker ability to model the temporal dependency during generation, which is consistent with the observations from previous works (Ren et al., 2020). To show that our model generates samples from a continuous space, we generate interpolation samples from our autoregressive flow model shown in Table 2. Those samples are mostly grammatically sound and correctly reflect the intermediate content of the two interpolated sentences.

While variational autoencoder has the issue of ignoring latent space (Li et al., 2019), our models do not suffer from this issue. We introduced two types of language generation models in the paper: (1) the autoregressive flow model (used in data augmentation tasks) and (2) the model that uses flow latent features as extra input (e.g., for QG tasks). Our autoregressive flow model's decoder is

Models	EM	F1
Baseline BERT	81.34	88.76
+ Context (Back-Translation)	81.02	88.79
+ Context (Paraphrasing)	81.65	88.92
+ Context (Language Flow)	82.28	89.22
+ Context (Language Flow+)	82.49	89.44

Table 9: QA results on SQuAD test split. The augmented data (new articles) significantly improves a strong SQuAD QA baseline.

the inverted version of its encoder with the same weights, so it ensures the decoder uses the latent features. When we use flow latent features as extra inputs, it significantly improves QA performance (Table 5 and Table 6), which implies the latent features are usefully involved in generation.

7 Conclusion

We have proposed a language generative flow model with non-autoregressive and autoregressive variants. The non-autoregressive flow model achieves strong performance on density estimation and helps improve question generation and machine translation by providing additional useful latent features to the decoder. Moreover, the autoregressive variant largely improves question answering by generating new contexts with noise injection.

Acknowledgments

We thank the reviewers for their helpful feedback. This research is supported by NSF-CAREER Award 1846185, ONR Grant N00014-18-1-2871, and ARO-YIP Award #W911NF18-1-0336. The views contained in this article are those of the authors and not of the funding agency.

References

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *CoNLL*.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. Wit3: Web inventory of transcribed and translated talks. In *Conference of european association for machine translation*, pages 261–268.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Laurent Dinh, David Krueger, and Yoshua Bengio. 2015. Nice: Non-linear independent components estimation. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. 2018. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087. PMLR.

Unnat Jain, Ziyu Zhang, and Alexander G Schwing. 2017. Creativity: Generating diverse questions using variational autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6485–6494.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Durk P Kingma and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 228–231. Association for Computational Linguistics.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *EMNLP*.

Jie Lei, Licheng Yu, Mohit Bansal, and Tamara L. Berg. 2018. TVQA: localized, compositional video question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1369–1379. Association for Computational Linguistics.

Jie Lei, Licheng Yu, Tamara L Berg, and Mohit Bansal. 2020. Tvqa+: Spatio-temporal grounding for video question answering. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Bohan Li, Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick, and Yiming Yang. 2019. A surprisingly effective fix for deep latent variable modeling of text. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Hong Kong.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421. The Association for Computational Linguistics.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard H. Hovy. 2019. Flowseq: Non-autoregressive conditional sequence generation with generative flow. 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 4281–4291. Association for Computational Linguistics.

Tong Niu and Mohit Bansal. 2018. Adversarial over-sensitivity and over-stability strategies for dialogue models. In *Proceedings of the 22nd Conference on Computational Natural Language*

Learning, CoNLL 2018, Brussels, Belgium, October 31 - November 1, 2018, pages 486–496. Association for Computational Linguistics.

George Papamakarios, Theo Pavlakou, and Iain Murray. 2017. Masked autoregressive flow for density estimation. In *NeurIPS*.

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.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

Yi Ren, Jinglin Liu, Xu Tan, Zhou Zhao, Sheng Zhao, and Tie-Yan Liu. 2020. A study of non-autoregressive model for sequence generation. In *ACL*.

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*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1530–1538. JMLR.org.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. 2019. Discrete flows: Invertible generative models of discrete data. In *Advances in Neural Information Processing Systems*, pages 14692–14701.

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.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Shiyue Zhang and Mohit Bansal. 2019. Addressing semantic drift in question generation for semi-supervised question answering. 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 2495–2509. Association for Computational Linguistics.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. 2018. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3901–3910.

Appendix

A Experimental Setup

In this section, we introduce our experiment settings ranging from datasets usage, flow implementation details, question generation model, and data augmentation settings. We use a fixed seed 2020 for PyTorch random seed.

A.1 Software/Hardware Usage

We use PyTorch 1.5 (Paszke et al., 2017) to build our model. We use Nvidia GeForce RTX 2080ti and Intel CPU (Intel(R) Xeon(R) Silver 4114 CPU

@ 2.20GHz) built on Ubuntu 16.01 for each training or inference process.

A.2 Datasets

We use SQuADv1.1 (Rajpurkar et al., 2016)[7] and TVQA (Lei et al., 2018)[8] to perform our experiments, since SQuAD is a widely explored QA and QG dataset, and TVQA is a video-based multimodal dataset with rich dialogue context. Therefore, question generation, context generation, and language density estimation, and data augmentation can be well performed and evaluated comprehensively on these two datasets and tasks.

TVQA consists of 152,545 QA pairs from 21,793 clips, spanning over 460 hours of video. The subtitles in TVQA dataset has time-stamp annotations of localized clip in the full subtitle clip. The localized clip is the relevant interval of a clip for question answering. In both the video question generation task and the context generation task for data augmentation, we use localized subtitles. The TVQA context features are dialogues or video subtitles; hence data augmentation on this dataset should consider an additional frame-level dimension.

SQuAD has over 100,000 questions and 23,215 paragraphs for the 536 articles covering a wide range of topics. We follow Zhang and Bansal (2019) to split the development set of SQuADv1.1 (Rajpurkar et al., 2016) into two splits and show the result on the second split.

A.3 Preprocessing

We tokenize the data to be used for both GLoVe embedding and BERT features extraction, and we add the start of sentence token and the end of sentence token for every input.

A.4 Evaluation Metrics

We generally follow previous work on evaluation metrics across density estimation, question generation, and question answering augmentation.

Flow Model. For flow density estimation, we follow previous work (Kingma and Dhariwal, 2018; Dinh et al., 2017) to use negative log-likelihood for comparison.

Question Generation Model. We evaluate the generation quality by BLEU4 (Papineni et al., 2002), Meteor (Lavie and Agarwal, 2007), and Rouge-L (Lin, 2004) to provide an insight into the performance of our model. We also use Amazon Turk human evaluation that compares the baseline generation and the proposed model generation by proving a suitable QA context. For SQuAD QG, we present the article context, question pairs, and the answer for the users to select their preference in terms of answerability and overall quality of the question pair. For TVQA QG, we present the video clip, subtitle context, question pairs, and answer candidates for the users to select their preference in terms of answerability and overall quality of the questions pair.

Machine Translation Model. We evaluate the generation quality by BLEU (Papineni et al., 2002) to provide an insight into the performance of our model.

Data Augmentation Model. We use accuracy scores to evaluate TVQA QA model, and follow previous work (Rajpurkar et al., 2016) to use EM (exact match) and F1 score to evaluate SQuAD QA model.

A.5 Flow Implementation Details

The experiment on base flow models does not involve extensive hyperparameter search trials since flow models follow the principle: the deeper, the better. We use small-sized flow models across different versions of (K=8, L=3, parameter number: 128M for transformer module and 196M for RNN module) flow models for ablation study. The autoregressive flow model has K=24, L=1 with approximately the same parameter number, 130M, by changing the nonlinear functions complexity for a fair comparison.

Based on the sequence length distribution of the dataset, we designate the maximum fixed flow sequence length for TVQA-subtitles as 64, SQuAD-paragraphs as 256. We set L=3 or 4 for all experiments while changing the number of flow steps K with a multiple of 8. While it follows that the more K, the better, setting L to a reasonable value is essential as each block will reduce sequence length by half. Therefore, L is set according to the length of the input.

The discretization, d, in the negative log-likelihood loss function (Eq.6) is set to 2^n, where n=6. Noise, $\mathbf{u}$, is set as Gaussian sample with $\boldsymbol{\alpha} = \frac{1}{2^m}$, where m=6. We follow previous work (Kingma and Dhariwal, 2018) to use bits per

[7] Online link for SQuAD: rajpurkar.github.io/SQuAD-explorer/explore/1.1/dev/

[8] Online link for TVQA: tvqa.cs.unc.edu/

dimension to regularize the negative log-likelihood loss, formulated as $\frac{\mathcal{L}}{(M \log(2))}$, where M represents the dimension of input.

We use a learning rate between 1e-4 and 1e-5, specifically 5e-5, to achieve stable and faster convergence (with Adam optimizer (Kingma and Ba, 2015), beta1=0.9, beta2=0.999). With prior knowledge of Adam optimizer, we perform 5 trials to test learning rate (1e-3, 5e-4. 1e-4. 5e-5, 1e-5) to find the fastest convergence rate.

The average training time is 50 epochs for a (k=8 L=3 parameter number: 128M for transformer module and 196M for RNN module) flow model, as each epoch takes 20 minutes. Inference for one sample takes around 0.01s.

The density estimation by the LSTM model we use for baseline comparison in NLL and QG models is designed to be well defined as a density estimation model. Flow density estimation models with no invertibility are not well-defined. Therefore, we mimic a model structure that the transformation is through only non-singular matrix weight to obtain an arithmetically invertible model.

A.6 Question Generation Implementation Details

The experiment on question generation models does not involve extensive hyperparameter search trials, as the proposed model has stable convergence under varied circumstances. We take the last latent space output of the flow model as the features used for the QG model decoder or attention map.

We use (K=16, L=3 parameter number: 256M parameters for transformer module) flow models with transformer modules without autoregressive decoding for all the QG experiments. The loss weight of λ_1 is set 1.0; the weight will not significantly affect the result as long as it is set to a reasonably large value. We use gradient descent with momentum optimizer (momentum = 0.8, lr = 1e-3) for both base model and flow model. With prior knowledge of the SGD optimizer, we perform four trials to test the learning rate (1e-2, 5e-3. 1e-3. 5e-4) to find the fastest convergence rate and stable training.

We employ Zhang and Bansal (2019)'s baseline QG model, which is a robust encoder-decoder attention generation network with a maxout pointer network and self-gated attention (Zhao et al., 2018) for both tasks.[9] We use pretrained BERT (Devlin et al., 2019) hidden features with 768 dimensions by a small uncased BERT model to replace GLoVe embedding to make the baseline stronger to show that the flow model can still improve well on a strong baseline.

The average training time is 20 epochs for the joint training of the QG model and the (k=16 L=3) flow model, as each epoch takes 50 minutes. Inference for one sample takes around 0.03s.

A.7 Machine Translation Implementation Details

For the machine translation dataset WMT16, the source and target languages share the same set of subword embeddings. The maximum text length is set to 64 and we filter out all data that is above this range. We use (K=4, L=3 with transformer module) non-autoregressive flow models with transformer modules for all the data augmentation experiments. We use Adam optimizer (Kingma and Ba, 2015) with beta1=0.9, beta2=0.999, and a learning rate 5e-5 for flow model training.

A.8 Data Augmentation Implementation Details

The experiment on context generation models generally follows empirical hyperparameter settings.

We use (K=32, L=4 parameter number: 512M parameters for transformer module) autoregressive flow models with transformer modules for all the data augmentation experiments. We use Adam optimizer (Kingma and Ba, 2015) with beta1=0.9, beta2=0.999, and a learning rate 5e-5 for flow model training and an empirically stable learning rate 3e-4 for attention decoder training. We set $\mathbf{z}_0$ to a Gaussian noise sample with mean 0.0 and variance 1.0 during training and variance 0.5 during inference. For inference variance tuning, we start from variance 1.0 and gradually decrease by 0.1 until 0.1 to manually check which setting has generated samples with reliable quality and diversity suitable for robust data augmentation.

The average training time is 100 epochs for the (k=32 L=4 parameter number: 512M) augmentation flow model, as each epoch takes 30 minutes. Inference for one sample takes around 0.5s.

Base QA model. We use model, backbone + Attn. Sup. + Temp. Sup. + local (STAGE) with GloVe embeddings, developed in TVQA+ dataset (Lei et al., 2020) as the QA baseline for TVQA data

[9] Maxout pointer is not used in the TVQA QG model since the number of words out of vocabulary is small.

augmentation model. We use the BERT baseline (Devlin et al., 2019) for SQuAD QA (Rajpurkar et al., 2016); this BERT Baseline is pretrained and uncased with 768 base dimension and finetuned on the SQuAD dataset. These two models are also used as data filters.

Data Augmentation Strategies. The training schemes are crucial for context generation since the TVQA model has heavy dependence on the subtitles and SQuAD model on the paragraphs: similar to Zhang and Bansal (2019)'s strategies, we obtain approximately ten times the amount of augmented data than the original amount, and filter them to obtain approximately 40% of augmented data to be used for training. We set a probability, 0.5, for replacing the original data with newly generated filtered data for each batch in training. For TVQA data augmentation, we generate localized subtitles and replace the corresponding part in non-localized full-subtitles. For SQuAD data augmentation, We generate trunks of paragraphs that do not contain answers to replace the corresponding trunks in the original paragraphs.