EasyChair Preprint

№ 2031

Binary Autoencoder for Text Modeling

Ruslan Baynazarov and Irina Piontkovskaya

November 25, 2019

Binary Autoencoder for Text Modeling*

Ruslan Baynazarov[1]
bainazarov.rr@phystech.edu
And
Irina Piontkovskaya[2]
piontkovskaya.irina@huawei.com

[1] Moscow Institute of Physics and Technology
[2] Huawei Noah's Ark Lab, Moscow

Abstract. Variational Autoencoders play important role in text generation tasks, when semantically consistent latent space is needed. However, training VAE for text is not a trivial task due to mode collapse issue. In this paper, autoencoder with binary latent space trained using straight-through estimator is shown to have advantages over VAE on text modeling task. In our model, Bernoulli distribution is used instead of Gaussian (usual for VAE). The model can be trained with only reconstruction objective, without using any additional terms such as KL divergence. Experiments reported in this paper show binary autoencoder to have the main features of VAE: semantic consistency and good latent space coverage; while not suffering from the mode collapse and being a lot easier to train than VAE.

Keywords: text autoencoders · VAE · binary latent representations

1 Introduction

In Natural Language tasks, building semantically consistent latent space is crucial. Variational autoencoders provide an appealing algorithm of building such a vectors without supervision.

Main advantage of VAE is the ability to train good latent semantic space. This means that we expect correspondence between some distance in latent space and semantic similarity. Another desirable property is smoothness of the distribution over the latent space. We would like to be able to sample latent variables from known distribution and then generate realistic and diverse examples from the samples. In VAE framework, model learns distribution of latent values on training dataset to be close to predefined prior distribution. For data generation, latent values are usually sampled from this prior.

It was shown in previous work, that VAE often suffers from the so-called variable collapse: the model learns to ignore latent variables, and latent space loses semantic properties. Another known problem is the difference between prior and posterior distributions [12]. In this case we can have some areas in latent

* The code is released on github https://github.com/hocop/binary-autoencoder

space where there are no data points. Sampling from these areas could lead to non-realistic generated examples, because the generator didn't see these values during training.

In this work, we propose novel model for texts, which allows to overcome the above problems. In our model we use binary latent vectors. Our latent representations are stochastic Bernoulli variables. Unlike VAE, we train the model with maximum likelihood objective, without setting any pre-defined prior distribution. We show experimentally, that our model learns roughly uniform posterior distribution over latent space, and does not suffer from mode collapse.

2 Related work

2.1 AE and VAE for text generation

One of the most important achievements in text representations is seq2seq model [18], which learns latent text representation jointly with encoder and decoder by optimizing maximum likelihood objective. This model led to a great improvement in Machine Translation. Seq2seq autoencoders were also considered [11], but their application was limited because of lack of semantic consistency of learned latent space.

Variational autoencoders, proposed by [10], [14], are able not only to reconstruct data, but also to learn latent space with good semantic properties. They were successfully applied to paraphrase generation [6], dialogs [17], and other text generation tasks [16], [21].

Seq2seq model is trained to estimate conditional probability of generated data $p(x|z)$. In contrast, in VAE framework, latent distribution for each data example is learnt instead of determenistic latent representation. The training objective usually is not tractable and is estimated by evidence lower bound (ELBO):

$$\mathcal{L} = \mathbb{E}_{z\sim q(z|x)}[\log p(x|z)] - KL[q(z|x) \parallel p(z)]$$

ELBO can be considered as sum of two parts: the first is equal to maximum likelihood (the so-called *reconstruction term*); the second is Kullback-Leibler divergence between prior distribution, usually Gaussian, and actual posterior distribution over latent space. In order to backpropagate through the random node, the following reparametrization trick is considered:

$$z(x) = \mu(x) + \sigma(x) \cdot \xi,$$

$$KL = \frac{\mu^2 + \sigma^2}{2} - log(\sigma) - \frac{1}{2},$$

where $\xi \sim G(0, 1)$.

It was shown in [2], that keep a balance between these two terms in text VAE is not a trivial task. Different improvements of text VAEs was proposed, handling this problem, like using convolutional decoder [20], changes in inference procedure [9], skip connections from latents in sequence generator model [3].

To deal with difference between prior and posterior distribution, sophisticated sampling procedure in proposed in [4], taking into account the actual posterior distribution (realistic constrains).

Comparison between VAE and determenistic autoencoders were done in [5]. Authors show that autoencoders can outperform VAE in different tasks, if they are equipped with proper regularizations. In contrast, in our work there is no explicit regularization term added to autoencoder objective.

2.2 Discrete latent representations

Discrete latent representations were widely studied, e.g. [15], [8], [13]. Training discrete model is not trivial, because gradient cannot be calculated for non-smooth functions.

Several methods were proposed to overcome this issue. The simplest approach is straight-through estimation [1], which is used in our work. Yet effective and widely used in training quantized networks [23], this method still has no consistent theoretical background in general case, only some limited results are available [22].

Another way to deal with gradients through categorical variables is Gumbel Softmax [7], which provides smooth approximation for discrete categorical distribution. This method uses the following reparametrization trick:

$$p(x) = softmax(logits(x)),$$

$$z(x) = softmax(logits(x) + g),$$

$$KL = p \cdot log(p) + (1-p) \cdot log(1-p) - \frac{1}{2},$$

where g is sampled from Gumbel distribution using:

$$g = -log(-log(\xi)),$$

where $\xi \sim U(0,1)$.

3 Our model

We propose a textual autoencoder with latent space consisting of binary vectors. For example, $[0,1,1,0,0,0]$ is a binary vector of size 6. Encoder output is passed through a linear transformation followed by a sigmoid:

$$p = sigmoid(W_1 \cdot [h_f, h_b] + b),$$

Where h_f, h_b are the forward and backward encoder outputs respectively; W_1 is a matrix of shape $[2 \cdot hidden\_size \times latent\_size]$; b is the bias vector of shape $[latent\_size]$. $p = p(z|x)$ is a probability vector with the same size as the latent vector.

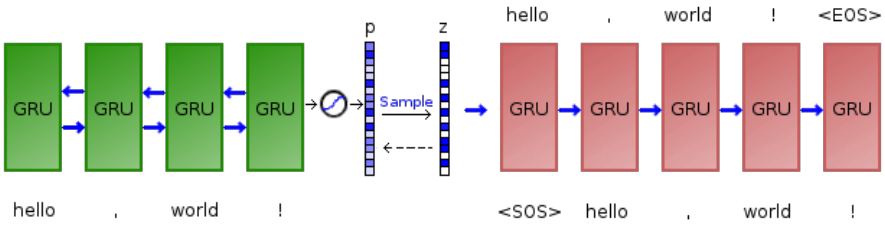


Fig. 1: Model architecture. Encoder output is passed through sigmoid to produce probability vector of multivariate Bernoulli distribution. Binary vector z is sampled from this distribution and then passed to the decoder. Gradient is backpropagated through sampling operation unchanged.

3.1 Training

Vector p is used as probability of a Bernoulli distribution. Discrete binary vector z of the same shape is sampled from the multivariate Bernoulli distribution: each p_i is the probability of discrete element z_i to be one, and $1 - p_i$ is the probability of z_i to be zero. z is sampled using the following operation:

$$z = step(p - \xi).$$

Where ξ is a vector of i.i.d. random variables $\xi_i \sim U(0, 1)$; *step* is elementvise Heaviside step function. Note that $\mathbb{E}z = p$. Also note that $z_i \neq 0.5$ almost surely.

Gradients do not backpropagate through *step* operation. Gradients are made to backpropagate from z to p unchanged:

$$\frac{\partial \mathcal{L}}{\partial p} := \frac{\partial \mathcal{L}}{\partial z};$$

or:

$$z = step(p - \xi) + (p - stop\_gradient(p)).$$

In literature, this is called Straight-Throw Estimation of gradient [1].

Randomly sampled discrete vector z is then passed to the decoder:

$$initial\_state = W_2 \cdot z,$$

Where W_2 is a matrix of shape $[latent\_size \times hidden\_size]$.

The loss function used here is the cross entropy of reconstruction made by decoder. Unlike in VAE, Kullback-Leibler divergence is not used here.

3.2 Inference

At the inference time there are two options. Vector p can be deterministically rounded to a vector of zeros and ones of the same size:

$$z = step(p - 0.5),$$

and then passed to the decoder. Also, p can be passed to the decoder as the mean value of z:

$$z := \mathbb{E}z = p.$$

3.3 Motivation to use the Straight-Throw Estimator

The key advantage of STE autoencoder against Gumbel-softmax autoencoder is that when sampling directly from Bernouli distribution, we get binary vectors consisting of zeros and ones. In contrast, when sampling from Gumbel-softmax, we get still continuous probabilities which means that the decoder does not see the actual binary vectors when training. For this reason, we refer to STE autoencoder as *binary autoencoder* in the next sections.

Considering that the latent space is constrained to be binary, we may not need to use any other restrictions to conserve the consistency of latent space, i.e. the closeness of the encoded sentences in the latent space. That means that usage of KL divergence can be unneccesary.

Using the stochastic process of sampling from Bernoulli distribution, we may also expect that the decoder will see enough points from the binary space during the training procedure.

4 Experiments

Model	PPL_{mean}	PPL_{greedy}	PPL	NLL (KL)
LM			120.7	106.8
Bottleneck			**7.4**	**50.9**
Gumbel	**18.9**	32.6	24.15	80.76
binary	26.3	**30.9**	36.3	89.8
binary (WD)	32.5	38.0	44.5	95.0
VAE (WD)	44.6		64.7	100.2 (**13.2**)
Gumbel VAE	22.3	35.3	31.1	86.4 (8.9)
Gumbel VAE (WD)	27.2	42.8	36.2	90.4 (9.7)
binary VAE	104.3	$1.5 \cdot 10^5$	104.5	107.0 (0.1)
binary VAE (WD)	71.9	94.0	115.4	109.1 (3.5)

Table 1: Language modeling results. PPL_{mean} stands for perplexity of the decoded text when z is picked as mean from $p(z|x)$; PPL_{greedy} stands for greedy picking of $z = argmax_z[p(z|x)]$ where z can only be binary vector; PPL stands for average perplexity when z is sampled from the distribution used by its specific autoencoder; NLL stands for average negative log-likelihood when z is sampled and KL stands for Kullback-Leibler divergence.

4.1 Dataset

We use large document classification dataset Yahoo Answer. We use $9.2M$ sentences for training, $10k$ for dev and $10k$ for test. We did tokenization, lowercasing and text clearing. Prepared dataset is available in our repository.

4.2 Model parameters

As encoder we used one layer bidirectional GRU with hidden size 512. We have compared our latent representation with Gaussian VAE, Gumbel VAE and bottleneck autoencoder. We found that latent size between 30 and 100 does not significantly affect quality. Latent size 50 is used for every autoencoder. The decoder is a single layer GRU with hidden size 512.

Other parameters: vocabulary size: $20k$; dropout rate: 0.2; batch size: $5k$ tokens. Adam optimizer with constant learning rate 10^{-3} is used here. As input embeddings, pretrained GLoVe with dimension 300 is used. Model is trained one epoch on $9.2M$ sentences dataset. When training VAE, KL cost annealing was used: KL cost was set to grow linearly from 0.01 to 1 for first $1M$ training samples. For models with word dropout, drop probability 0.5 was used.

4.3 Language modeling results

We have measured the performance of our model on language modeling task. Results are shown in Table 1. For reference, language modeling results for pure decoder without any inputs are reported ("LM" in the table). Gumbel-softmax autoencoder trained without KL divergence ("Gumbel" in the table) shows good language-modeling performance when decoding from mean values i.e. probabilities and from its own sampled values, i.e. sampled by Gumbel-softmax. However, our model ("binary" in the table) outperforms Gumbel autoencoder when decoding from vector rounded to zeros and ones i.e. the true latent space. Note that although PPL and NLL scores are higher for our model, it does not represent the real difference between autoencoders since values put into decoder were sampled from different distributions: Gumbel-softmax and Bernoulli.

Variational autoencoder with Gaussian prior was trained with word-dropout ("VAE (WD)" in the table). In our setup it performed slightly better than pure language model unlike in [2]. Gumbel-softmax VAE performed well both with and without word dropout. We also tried to train our binary autoencoder with straight-throw estimator with KL divergence term. As language model it shows poor quality.

Binary VAE without word dropout shows collapse of latent space. Note the high perplexity score when using greedily obtained binary vectors. The probabilities in the latent space are all close to 0.5. Also they are highly correlated. In our experiments, when rounding them to 0 and 1 we got vectors of mostly the same numbers (all zeros or all ones). This result was not obvious: if the weights of a model are initialized randomly than how did the latent space become correlated *with the same sign*? One explanation can be that for the decoder values 0 and

1 are not symmetric. When the binary vector is multiplied by a matrix, value 1 gives some "meaningful" (i.e. trained) additive contribution to the resulting vector, and value 0 gives always zero contribution. It seems that the decoder was trained to rely not on information from the encoder itself, but on the fact that *some* information is present. That explains the high perplexity score, since the decoder have not seen such binary vectors in the training process: the binary vector consisting of same numbers has very low probability of being sampled.

4.4 Generating sentences from the discrete space

Interesting results were obtained in [2]: two random points in VAE space were chosen and sentences were generated from the points between these two. Resulting sentences appeared to be grammatically correct and semantically layed *between* the original two sentences.

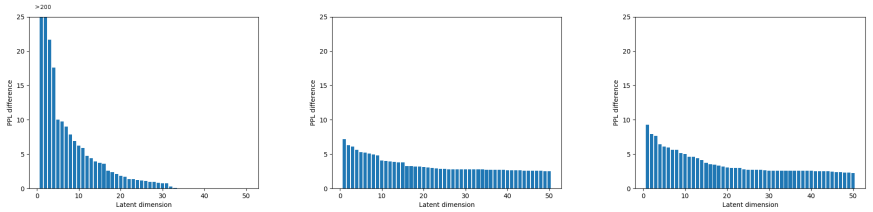
We tried to repeat this experiment with binary autoencoder. Path between two binary vectors z^1 and z^n can be defined as the set of binary vectors z^i such that Hamming distance between z^i and z^{i+1} equals one, $i \in \{1, n-1\}$. Unlike in VAE continuous latent space, in discrete space there can be many paths between two vectors. To generate sentences, we choose one of the paths randomly. Results can be seen in Table 2.

can i find a phone book on the phone at the same time?
can i watch a show at the same time?
can a website tell me the details of the show
do a girls watch the show and see what happens about the boys
do a visit and see what the us is talking about
every site is the store, and they will be in the same store for the same price.
everyone is in the store, and they will be able to show the information.
here is the list, and then click on the size of the store.
everyone is, and they are usually called the average joe.
everyone is different, and they will always be normal.

Table 2: Paths between pairs of random points in binary space: Note that intermediate sentences are grammatical, and that topic and syntactic structure are usually locally consistent.

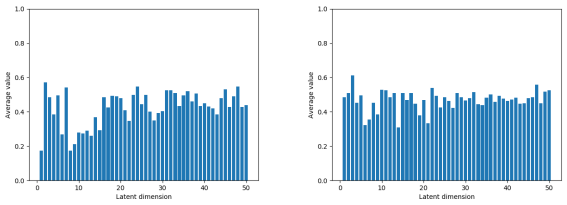
4.5 Latent collapse

It is known for VAE that some dimensions of latent space tend to collapse [2], [20]. This means that KL divergence of the dimension is close to 0 and this dimension does not encode any useful information. To measure this effect we have tried to "spoil" one dimension i.e. multiply all its values by (-1) and measure the perplexity on test set with only this one dimension spoiled. Results



(a) VAE (WD) (b) binary (c) Gumbel VAE (WD)

Fig. 2: Columns show how much does the perplexity grow when i'th dimension of latent space is distorted. Columns are sorted by height.



(a) binary (b) Gumbel VAE (WD)

Fig. 3: Average values of bits on the test set. Note that no bit is always 0 or 1. Dimensions are sorted in the same order as in Figure 2.

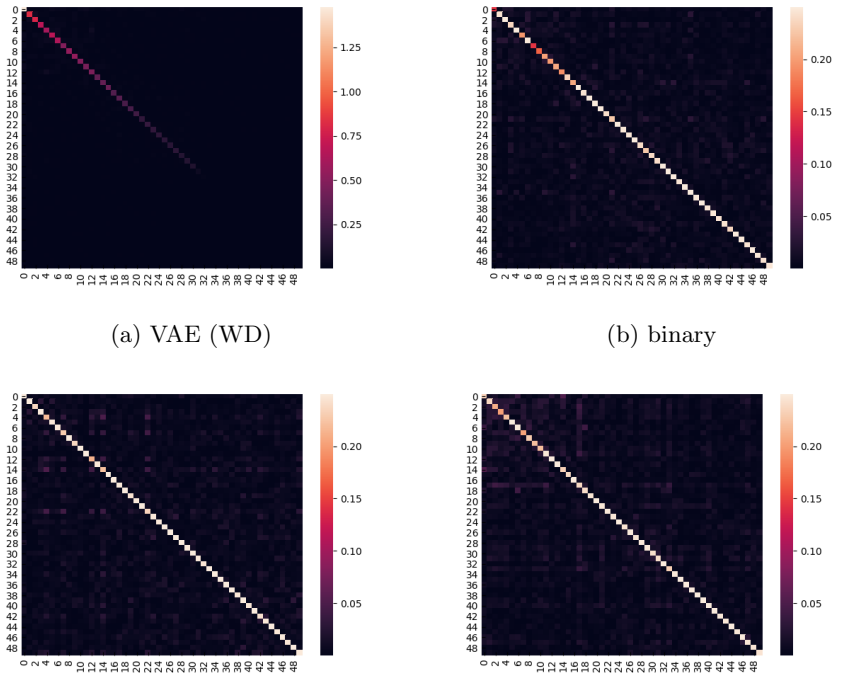
for all VAE dimensions are shown in Figure 2a. It is shown that decoding is not sensitive to changing some values, i.e. these dimensions encode close to zero useful information.

To conduct the same experiment with binary latent space we selected one bit and spoiled it by replacing 0 to 1 and 1 to 0. Results are shown in Figure 2b. As can be seen from the figure, spoiling any bit results in noticeable perplexity growth. However, no bit is as crucial for decoding quality as some of VAE's dimensions. Similar results were obtained for Gumbel-softmax VAE in Figure 2c.

It is also useful to encode test set to binary space and measure the average value of each bit. In case of some kind of latent collapse, some bits may be always 0 or 1. Spoiling these bits can result in perplexity growth not because these bits encode useful information but because the decoder has never seen such values. Figure 3 shows that the average values of bits are not too close to zero or one. Similar results were obtained for Gumbel-softmax VAE.

It is interesting to note that those bits which are more shifted towards one of the values $\{0, 1\}$ tend to show more "importance".

The fact that certain bit is not always 0 or 1 does not guarantee that it encodes useful information and that the latent space is fully covered. Some bits may have strong correlations and though may not be independent, i.e. encode



(a) VAE (WD) (b) binary

(c) Gumbel VAE (WD) (d) Gumbel AE

Fig. 4: Absolute value of covariance matrices between dimensions of latent space. Dimensions are sorted in the same order as in Figure 2.

the same information. Also this would mean that not all of the combinations of bits are observed by the decoder. That could be seen as a sort of latent collapse. Figure 4b shows that no such correlations are observed. The same covariance matrix for VAE in Figure 4a demonstrates latent collapse. Results similar to the binary case were obtained for Gumbel-softmax VAE. However, correlations for some of dimensions are slightly more noticeable.

A conclusion can be made that binary autoencoder does not suffer from latent collapse and that the useful information about the input sentence is distributed relatively uniformly between the binary dimensions.

4.6 Supervised classification using latent vectors

Yahoo dataset contains labels for each sentence representing one of the 10 classes: Science, Sports, Business etc. In order to test the ability of latent representation to preserve the meaning of a sentence we have tried to use the autoencoder's latent vectors as features for classification task. For this purpose, a dense model with 2 hidden layers containing 100 units each was trained. The final layer is

softmax over 10 classes. The input to this model is latent variable, obtained from autoencoder. Samples are normalized so that the mean and variance of a train set are 0 and 1 respectively. Training set consists of $9k$ samples. Quality is measured on development set, containing $1k$ samples. None of those sentences were used in the autoencoder training.

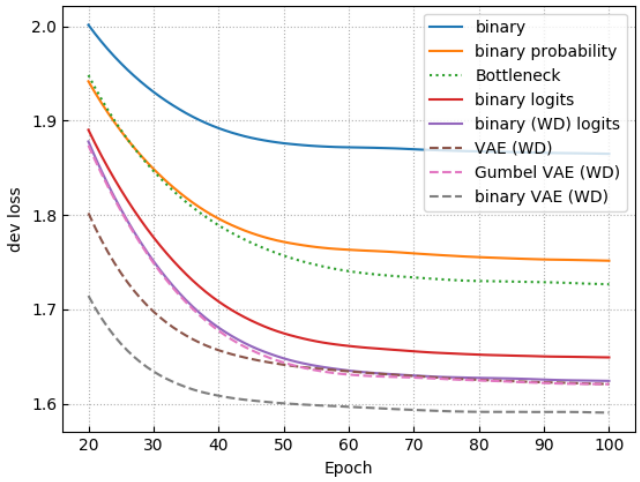


Fig. 5: Training curves for supervised classification task. Latent variables obtained from different autoencoders were used as input features to feedforward classifier.

Gaussian VAE representation was chosen as a strong baseline. Bottleneck autoencoder representation was also tested in this experiment. As was expected, VAE representation performs noticeably better than bottleneck, as shown in Figure 5. Using binary vector as input to classifyer appeared to perform poorly. Using probabilities of the Bernoulli distribution instead of binary samples improves the classification quality. However, it is still slightly worse than the bottleneck results. It is possible that probability is not a good feature for a neural network due to the NN's linear nature. Using logits before the sigmoid operation from binary autoencoder (i.e. the energy of Bernoulli distribution) as input features to classifyer significantly improves classification quality. It is still slightly worse than VAE. Assumption is that VAE makes better sentence meaning representation because it was trained with word dropout. Adding word dropout to binary autoencoder pushes the classification quality close to that of VAE. Using KL divergence when training binary autoencoder results in the highest classification quality in our experiments although it results in the worst language model-

Input Encoder	Input Features	NLL	Accuracy
Bottleneck	latent variable	1.73	0.408
VAE (WD)	mean	1.62	0.466
binary	zeros and ones	1.86	0.372
binary	probability	1.75	0.405
binary	logits	1.65	0.439
binary (WD)	logits	1.62	0.461
binary VAE (WD)	logits	**1.59**	**0.478**
Gumbel	logits	1.69	0.438
Gumbel VAE	logits	1.65	0.445
Gumbel VAE (WD)	logits	1.62	0.458

Table 3: Classification results. Negative log likelihood and accuracy metrics are presented.

ing scores. Experiments with Gumbel-softmax autoencoder showed the results slightly worse than using binary latent space.

Note that classification using the output of the autoencoder is not expected to be efficient. The obtained results are rather weak compared to the baselines [19] but are still useful for comparing between different latent representations.

5 Conclusion

This paper studies the binary autoencoder with straight-through gradient estimator used for natural language sentences. Comparing to VAE this autoencoder does not suffer from latent collapse when KL divergence tends to zero. This makes binary autoencoder much easier to train since it does not require tricks such as KL cost annealing and word dropout. Comparing to Gumbel-softmax VAE, STE binary autoencoder produces better latent representation of a sentence and also is simpler in training. Binary autoencoder preserves the main feature of VAE: it densely covers its own latent space and forms meaningful clusters in it. Also, binary autoencoder uses only reconstruction error as a loss function unlike VAE, which uses variational lower bound as loss. This can make it easier to apply autoencoders with binary latent space to more types of data in the future.

References

1. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
2. Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A.M., Jozefowicz, R., Bengio, S.: Generating sentences from a continuous space. arXiv preprint arXiv:1511.06349 (2015)
3. Dieng, A.B., Kim, Y., Rush, A.M., Blei, D.M.: Avoiding latent variable collapse with generative skip models. arXiv preprint arXiv:1807.04863 (2018)

4. Engel, J., Hoffman, M., Roberts, A.: Latent constraints: Learning to generate conditionally from unconditional generative models. arXiv preprint arXiv:1711.05772 (2017)
5. Ghosh, P., Sajjadi, M.S., Vergari, A., Black, M., Schölkopf, B.: From variational to deterministic autoencoders. arXiv preprint arXiv:1903.12436 (2019)
6. Gupta, A., Agarwal, A., Singh, P., Rai, P.: A deep generative framework for paraphrase generation. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
7. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144 (2016)
8. Kaiser, Ł., Bengio, S.: Discrete autoencoders for sequence models. arXiv preprint arXiv:1801.09797 (2018)
9. Kim, Y., Wiseman, S., Miller, A.C., Sontag, D., Rush, A.M.: Semi-amortized variational autoencoders. arXiv preprint arXiv:1802.02550 (2018)
10. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
11. Li, J., Luong, M.T., Jurafsky, D.: A hierarchical neural autoencoder for paragraphs and documents. arXiv preprint arXiv:1506.01057 (2015)
12. Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders. arXiv preprint arXiv:1511.05644 (2015)
13. van den Oord, A., Vinyals, O., et al.: Neural discrete representation learning. In: Advances in Neural Information Processing Systems. pp. 6306–6315 (2017)
14. Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. arXiv preprint arXiv:1401.4082 (2014)
15. Rolfe, J.T.: Discrete variational autoencoders. arXiv preprint arXiv:1609.02200 (2016)
16. Semeniuta, S., Severyn, A., Barth, E.: A hybrid convolutional variational autoencoder for text generation. arXiv preprint arXiv:1702.02390 (2017)
17. Shen, X., Su, H., Li, Y., Li, W., Niu, S., Zhao, Y., Aizawa, A., Long, G.: A conditional variational framework for dialog generation. arXiv preprint arXiv:1705.00316 (2017)
18. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)
19. Yang, Z., Diyi: Hierarchical attention networks for document classification. Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies. pp. 1480–1489 (2016)
20. Yang, Z., Hu, Z., Salakhutdinov, R., Berg-Kirkpatrick, T.: Improved variational autoencoders for text modeling using dilated convolutions. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3881–3890. JMLR. org (2017)
21. Yin, P., Zhou, C., He, J., Neubig, G.: Structvae: Tree-structured latent variable models for semi-supervised semantic parsing. arXiv preprint arXiv:1806.07832 (2018)
22. Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., Xin, J.: Understanding straight-through estimator in training activation quantized neural nets. arXiv preprint arXiv:1903.05662 (2019)
23. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)