Text Generation: from Uncontrollable to “Controllable”

Yue Zhao (data mining: ensemble learning, anomaly detection, ML tools & systems)
Heinz College of Information Systems and Public Policy
Carnegie Mellon University
Notes & Disclaimer

Mainly based on the following papers*:


*Disclaimer: concepts/formulas/figures may be copied/adapted from the above papers. Research discussion only, not for redistribution.

*As it is a developing area, some statements can be empirical and observational (open to discuss)
Outline

Go over some basic NLG techniques


Discuss how to generate text from a VAE in an uncontrollable manner


Demonstrate the possibility of decoding the latent space


Present the idea of controllable text generation

Text Generation – Motivation

Is that possible to generate sentences/dialogues by models automatically?

Infinite monkey theorem: the infinite monkey theorem states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, such as the complete works of William Shakespeare [1].

Even it is “almost surely” to happen, the chance is so slim given the search space.

The generative process should involve learning/control to reduce the complexity.

[1] both the figure and the explanation are from Wikipedia (https://en.wikipedia.org/wiki/Infinite_monkey_theorem)
Text Generation

(Prehistoric) Existing approaches:

- **Template or rule-based models:**
  - Define templates and fill in the blanks.
    - This __ a good __! (limit the searching space/may be viewed as a prior)
  - Define grammatical rules to make sure the correctness:
    - Subject-Verb Agreement: she work(s) hard for the exam.
  - **Advantages:** interpretable and well-behaved
  - **Limitations:** huge amount of feature engineering/preprocessing is needed

- **Probabilistic models:**
  - N-gram [1]: unigram, bigram, trigram, four-gram, five-gram, ...
  - See other approaches here [2]


Text Generation – N-gram

N-gram [1]: unigram, bigram, trigram, four-gram, five-gram, ... (https://en.wikipedia.org/wiki/N-gram#Examples)

- From Shannon’s information theory: given a sequence of letters, what the likelihood of the next one?

N-gram model predicts $x_i$ based on the words preceding it: $P(x_i|x_{i-(n-1)}, \ldots, x_{i-1})$

The independence assumptions are made: each word only depends on the last $n - 1$ words.

\[
\begin{align*}
N = 1 &: \text{This is a sentence} \quad \text{unigrams:} \\
N = 2 &: \text{This is a sentence} \quad \text{bigrams:} \\
N = 3 &: \text{This is a sentence} \quad \text{trigrams:}
\end{align*}
\]

Figure Source: https://stackoverflow.com/questions/18193253/what-exactly-is-an-n-gram

Text Generation – N-gram

Advantages [1]:
- Intuitively simple, relatively fast, and easy to learn
- Work with almost any type of (sequence) data, e.g., images and genetic sequences
- Capture the ordering automatically
- Models are not biased by handcrafted features (compared with templated based generation) but learn from data directly

Challenges [1]:
- Lack of explicit representation of long-range dependency (depending on $n$ preceding tokens only)
- Sparsity may lead to disasters (low frequency/rare phrases), although smoothing is possible
- Performance tends to saturate with more training data (diminishing return)

Text Generation – Neural Machines

Neural machine-based models are proposed:

- Great empirical performance
- Can digest large amount of data (usually automatically)
- Limited interpretability and explainability

Tradeoff between predictability and flexibility (Fig.1):

This is analogous to classical machine learning models vs. deep learning models

- Classical models: fast, interpretable (linear models/decision tree), good for simple datasets, requires feature engineering
- DL models: computationally heavy, limited interpretability, learn representation automatically, good for large datasets

Balance the Tradeoff

How to balance the tradeoff between predictability (controllability) and flexibility (expressivity)?

Hopefully, we would like the model to be **flexible** enough to be powerful but still with some **controllability**.
A Brief History of Neural Nets in NLG

The use of neural networks in Natural Language Generation (NLG) evolves with time:

- Early stage: feedforward Fully Connected Neural Net
- Recurrent Neural Net (RNN) and its variations
- Convolutional Neural Net (CNN)
- RNN+CNN
- and more

The basic idea is to do next step prediction conditioned on the source and the previous targets, i.e., $p(y_t | X, y_{t-1}, ... y_1)$.


Encoder-Decoder Models

For example, for machine translation tasks $X$ might be a sentence in English and $y$ the translated sentence in Chinese.

**Encoder-decoder** models, also referred to as **sequence-to-sequence models**, is the most widely used architecture.

**Encoder**: maps the input sequence to a hidden vector $v$

**Decoder**: perform the language modeling based on $v$

**Prediction**: $p(y_j|v, y_1, ..., y_{j-1})$

Source: https://www.coursera.org/lecture/language-processing/encoder-decoder-architecture-bGV7m

Training Process

Existing approaches models the likelihood of the training data: \[ \mathcal{L}(\theta) = \sum_{t=1} log p(y_t|X, y_{t-1}, \ldots, y_1; \theta) \]

Training (encoding process):

- Maximize the likelihood of the training data (minimize the cross entropy loss)
- Optimize may use gradient descent, stochastic gradient descent, with optimizers like Adam [1]
- As mentioned, different types of neural nets can be used for this purpose

Sequence-2-sequence models work great for many predictive tasks in NLP, such as machine translation.

Can we use them for text generation?

*We are going to use this trick repetitively in this presentation (see https://www.quora.com/What-are-the-differences-between-maximum-likelihood-and-cross-entropy-as-a-loss-function)

Leveraging VAE in NLG

Why VAE?

Standard NN language models (e.g., RNN) generate sentences in a “discrete” and “local” manner: word-by-word with evolving distributed representation

- Breaking the model structure down into a series of next-step predictions leads to the loss of an interpretable representation of global features, e.g., topics, sentiment, high-level features

VAE approach:

- Capture global representation in a continuous manner

- Practical training technique for powerful NN training with latent variables (Gaussian reparameterization as Sam mentioned last time) so SGD is possible

Leveraging VAE in NLG

Existing Unsupervised Sentence Encoding Approaches:

- Predict the next word conditioned on the previous ones and the evolving hidden state. It does not learn the sentence representation but discrete ones (not the distributed representation)

- Need a systematic way to map sentences to distributed sentences in an unsupervised setting

There are methods which could learn distributed representations but cannot be used a generative setting

- Sequence autoencoder
  - Standard autoencoders which could extract data representation, but the learned intermediate representation is usually ungrammatical and do not transition from one to another smoothly
  - Another drawback is it does not incorporate a prior over hidden space $z$, so it could not be used to generate new samples. Standard autoencoders are not generative models

- Skip thought models [1]

- Paragraph vectors models [2]


Leveraging VAE in NLG

VAE (regularized version of standard autoencoders)

- Discussed in the last lecture by Sam

Another way to see it: impose a prior on the latent space $z$:

- Enforce regular geometry (intuitively, learn the code not as single points but as soft ellipsoidal regions (recall the shape of Gaussian) in the latent space)
  - Fill in space instead of memorizing training data as isolated codes

- Make it possible to do sampling

- It is noted that $KL(Q(z|x)||P(z|X))$ is non-negative. Removing it from LHS: LHS $\geq$ RHS; RHS is workable with optimization tools

Setting up the objective

$$\log P(X) - KL(Q(z|x) \parallel P(z|x)) = E_{z \sim q}[\log P(X|z)] - KL(Q(z|x) \parallel P(z))$$

• Remember variational inference?

$$ELBO(q) = E[\log p(x|z)] - KL(q(z) \parallel p(z))$$

Fig. 1. Objective Function of VAE (source: Sam’s Tutorial on VAE)
Challenges in Text Generation

Discrete vs. Continuous

Images are **contiguous** (RGB, pixels), while text are **discrete** [1]:

- Back-propagation would not be feasible for discrete outputs
- It is not straightforward to pass the gradients through the discrete output words of the generator

Text data is often considered as **discrete** and **non-differentiable**. Need word-embedding, simple bow, or one-hot vectors.

VAE has shown great performance on continuous domains but working on discrete sequence could still be challenging.

---

VAE Architecture in NLG

**Encoder:** Long Short Term Memory (LSTM)

**Decoder:** LSTM

Advantages of LSTM [1]:

- Able to bridge long time gaps in sequence problems
- Could handle noise, distributed representations, and continuous values (theoretically could handle unlimited state numbers)
- Less dependent on parameter tuning*
- Relatively fast compared with other sequence models*

---

Background

Force the model to decode plausible sentences from latent space $z$ that have a reasonable probability under the prior:

- Prior $P(z)$ and posterior $q(z|x) \rightarrow$ diagonal Gaussians
  - Reconstruction cost is calculated from samples generated by $q(z|x)$
  - As Sam pointed out last time, KL divergence term here has a closed form solution

In summary, a sequence autoencoder with Gaussian Prior working as regularizers on the latent space (recall we are trying to force it is close to the prior).

How to quantify the learning process and the quality of learned global features?

$$L_{VAE}(\theta; x) = -KL(q_\theta(z|x)||p(z) + E_{q_\theta(z|x)}[\log p_\theta(x|z)] \leq \log p(x)$$

A good models will have:

- non-zero KL Divergence (otherwise it learns nothing)
- Relatively small cross entropy term (data likelihood is high under the posterior)
Challenges in Model Training

Straightforward implementation will fail to optimize the target:

Training this complex model can be challenging; the optimization process may favor “low-hanging fruit”. One possibility is to focus on KL divergence term too heavily in the early stage.

- LSTM could worsen the situation as using it as decoder is sensitivity to the variation in the hidden space
- When all (or some of) these happen, the training is failed
  - The decoder will ignore the encoders and shows little to no gradient pass

To fix this optimization problem, two mechanisms are proposed (see the paper for details):

- KL cost annealing
- Word dropout and historyless decoding
KL Cost Annealing

**Variable (varying) the weight** to the KL term in the cost function during training

- Set weight=0 in the beginning, little to no regularization, encode as much info as possible in \( z \)
- Gradually increase the weight of the KL term to force the model to smooth out and get close to the prior

This can be viewed as an annealing process from standard autoencoder to VAE

- KL spikes in the early stage so model can encode \( z \) cheaply
- KL then decrease quickly with higher weight
- Slowly increase again while the model converges
- See red curve here (spikes and then decreases)
- Somehow you could relate this to adaptive learning rate and momentum

Generating sentences from a continuous space. Bowman et.al. 2015
Word Dropout

Decoder predicts words conditioned on the ground-truth of previous words:

- Robustness can be acquired by **dropping some information from the prediction conditions**, e.g., replacing some conditioning information word by “unknown (UNK)”

- This will force the model to more rely on the learned latent representation $z$ for better generalization capacity, instead of memorizing words

One understanding on this is that word should fit in its context. So human could still understand the main meaning of a sentence even some words are “missing”. [1]

In other words, more high-level info, e.g., semantic info, is captured by the model, which is a sign of success.

To certain extent, the rational sounds like the standard dropout in neural nets (randomly shut down neurons during training).

Results of Bowman 2015:

- Successfully use VAE for text generation (coherent and diverse sentences)
- Show some interpretability on the latent space (did linear interpolation to understand what is encoded in $z$ – what does neighborhoods in $z$ look like)
- VAE is not “controllable”!

Using the VAE architecture discussed above, text can be generated in a randomized and uncontrollable fashion.

The goal of Hu 2017 is to do “Controllable Text Generation”.

- The idea in a nutshell is to learn good latent representations of text
- Then enforce disentanglement to understand/separate certain dimensions in the latent space for high level semantics (sentiment, topics, etc.)
- So the generated text can be controlled by manipulating these dimensions (sentiment, topics, etc.,)
Toward Controlled Generation of Text

Why controllable text generation?

Use cases:

- **Dialogue system** (commercial bots): make sure the sentiment is positive/neutral
- **Grammatical considerations**: the tense should be consistent
- **What is your thought?**
Challenges – Non-differentiability

Non-differentiability hinders the use of global discriminators for backpropagation (recall our discussion on “continuous vs. discrete”)

To handle this non-differentiability gap, multiple approaches have been proposed:

- **Policy learning** [1]: modeling the data generator as a stochastic policy in reinforcement learning. **May have high variance.**

- **Approximation methods** [2,3]: Gumbel-softmax distribution, which is a continuous approximation to a multinomial distribution parameterized in terms of the softmax function [3]. **Only have preliminary qualitative results.**

- **Generative semi-supervised model** [4]: can also be seen as a hybrid continuous-discrete mixture model where the different mixture components share parameters (element-wise reconstruction error). **Lose the holistic view.**

However, none of these approaches are suitable for capturing global attributes, e.g., sentiment.


Challenges in Controllable Text Generation

How to enforce the disentanglement of the latent representation? We would like to control the generation process. It might be a good idea to learn certain attributes explicitly with GAN.

Generative adversarial network (GAN):

- **Discriminative Net** $D$: learns to distinguish whether a point is generated or real. Penalize $G$ for fake data.
- **Generative Net** $G$: learns to generate high quality data points to confuse $D$. Generated instances become negative training samples of $D$.

- Intuitively, we could view this in a "game theory setting" (not strictly though)

Thanks for Jonathon’s review on this topic 😊
Challenges in Controllable Text Generation

From the figure [3] here we could notice the output of generator is connected to the discriminator directly, and the generator’s weights will be updated by the discriminator by backpropagation.

Applying GAN architecture in generating sequences, e.g., text, shows limitations [1]:

- GAN is designed for generating **real-valued continuous data** instead of discrete tokens such like text [2], “slight change”?
- GAN can only give loss/score for the full sequence but not the partial state. For text generation tasks, it may be challenging as it may generate word by word.


Challenge – Disentangled Representation

One underlying assumption is that we could learn disentangled latent representations.

In other words, we understand the meaning of certain latent dimensions.

Ideally, we could then manipulate certain dimensions of the latent space to generate the samples we expect.

How to enforce the full independence property on the full latent representation?
Disentangled Latent Representation

As pointed out in Bowman et.al. 2015, text generation by VAE is conditioned on extrinsic features via varying latent codes.

In an ideal scenario, we wish to understand the corresponding (high-level) meaning of each latent dimension (at least some).

If is this feasible, then we could control the text generation process by specifying the value for certain dimension:

- $z_1$ denotes sentiment
- $z_2$ denotes tense
- ...

As mentioned before, latent space is a compressed representation of the original feature space. If the independence constraint is not enforced, it is very unlikely the learned latent space is disentangled.

- $z_1 + z_3 + z_k \rightarrow$ controls both sentiment and tense

So the generation process is not controllable.
The Feasibility of Disentanglement Learning

**Exploratory experiment in an unsupervised setting**: see details in [1] Xu et.al. 2019.

**Dataset**: Yelp restaurant

**Architecture**: $\beta$-VAE; encoder and decoder are both LSTM

**Latent space**: 80 dimensions

**Identification process**:

- Normalize the value of each latent code by subtracting the mean estimated over all training samples.
- For each normalized latent code, use its polarity to classify the sentiment
- **Identify the latent code with the highest accuracy** for sentiment prediction -> leading factor for sentiment

The Feasibility of Disentanglement Learning

The experiment shows:

- **There is one latent code has >90% accuracy on predicting sentiment by its value alone**
- **All other factors have ~50% accuracy (~random guess)**

However, **it is still hard to generated controllable text by modifying this latent code**. Four strategies have been tested:

1. Fixing the relevant code to \((u - 2\sigma, u + 2\sigma)\)
2. Fixing the relevant code to \((u - \sigma, u + \sigma)\)
3. Fixing the relevant code to the \(max\) and \(min\) value from the training samples
4. Use the latent vector by 10 manually constructed parallel sentences with opposite sentiment (keep all other factors unchanged)
The Feasibility of Disentanglement Learning

The experiment shows none of the four strategies work well (reported the result of strategy 1 which looks best): the decoder network fails to generated the desired result, though specific code is identified as “sentiment” code.

So the conclusion is: disentangling latent representation in an unsupervised manner can be very challenging!

One of the selected papers for Nov 13th will give a more thorough analysis on this controversial topic: whether unsupervised learning of disentangled representation is feasible.

Controllable Generation: Proposed Model

Controlled Sentiment Generation:
- Assigned one specific code in the latent space to encode “positive” and “negative” semantics
- During the generation process: specifying the code value
- Repeat this process for all attributes to be controlled (tense, formality, etc.)
- Each specified code could capture a certain attribute and independent from each other

Challenges:
- VAE is unsupervised, how to disentangle the latent representations?
- If discriminators are used, how to handle the non-differentiability of text sequence?
Model Overview

It is based on the previous model we just discussed [Bowman 2015]

- New sentences are generated condition on latent code $z$

The proposed model adds two parts for representation disentanglement:

- **Structured code** $c$; each corresponds to an independent semantic feature of the sentence.
- New sentences are generated condition on **unstructured code** $z$ and **structured code** $c$, i.e., $(z, c)$.
- For each attribute code in $c$, a discriminator $d$ is set to measure the consistency between the desired result and the generated result $\hat{x}$
- $D$ will then “backpropagate” to $c$ for improving the generators
  - Challenge: Applying discriminators on text samples are challenging (discrete and non-differentiable)
- VAE encoder will be regarded as an additional discriminator to enforce the attributes captured in $z$ can be recovered from the generated sample (dashed blue line)
  - Ideally, varying code $c$ should **NOT** affect unstructured attributes if $z$ is unchanged.
Model Overview

In an ideal world, each structure code $c$ can “independently” control its target feature (free from the impact of other attributes).

Explicitly modeled codes are in $c$ and “irrelevant” attributes are completely captured in unstructured code $z$.

**VAE + Discriminator(s):**

- VAE trains the generator for generating plausible text via reconstructing real sentences (this process is unsupervised)
- Discriminators enforce the generator to create the samples that are coherent with the structure code $c$ (this process is supervised with labeled samples)
- The encoder may be viewed as an additional discriminator for the generator to enforced the generated samples have consistent unstructured code $z$
Model Overview

In an ideal world, each structure code $c$ can “independently” control its target feature (free from the impact of other attributes).

Explicitly modeled codes are in $c$ and “irrelevant” attributes are completely captured in unstructured code $z$.

So the generator, discriminators, and the encoder are working in a collaborative manner.

This can be very challenging!
Generator Learning

As mentioned, the generation process is conditioned on latent code \((z, c)\):

\[
\hat{x} \sim G(z, c) = p_G(\hat{x} | z, c) = \prod_t p(\hat{x}_t | \hat{x}^{<t}, z, c)
\]

\(\hat{x}\) is the generated token sequence; \(\hat{x}^{<t}\) stands for the tokens preceding \(\hat{x}^t\). So the generation is a sequence of discrete decision making by softmax at each timestamp \(t\).

For a single token, it is generated by \(\hat{x}_t \sim \text{softmax}(o_t / \tau)\). We will discuss the temperature parameter \(\tau\) soon, but it is normally set to 1.

In other words, sampling a token from a multinomial distribution parametrized by softmax. The reason a softmax function is used here is because the text is discrete (we will revisit this shortly).
Generator Learning

Modeling of \((z, c)\)

- \(z\) : standard Gaussian prior \(p(z)\) as a continuous distribution
- \(c\) : depends on specific high-level semantics to model; need to find appropriate priors for different \(p(c)\)
  - Continuous: formality, sentiment scores
  - Discrete: tense categories (past, present, future), sentiment categories (negative, neutral, positive)

Let the \(\theta_G\) and \(\theta_E\) denote the parameters of the generator \(G\) and encoder \(E\), the VAE tries to minimize the reconstruction error of the observed real sentences and regularize \(E\) to be close to the prior.
Generator Learning

Generator learning depends on two parts: VAE + discriminator(s)

Recalled the standard VAE learning objective (copied from Sam’s VAE tutorial last week):

\[ \log P(X) - KL(Q(z | X) \parallel P(z | X)) = E_{z \sim q(E)}[\log P(X | z)] - KL(Q(z | X) \parallel P(z)) \]

It is easy to see these two forms are very close with two major differences on the second term:

- the reconstruction loss depends on \((z, c)\) other than \(z\) only
- the reconstruction loss also depends on \(q_D(c|x)\), the conditional distribution defined by discriminator \(D\) on each structured code \(c\)

It is noted that the distribution over \((z, c)\) can be factored in to \(q_E\) and \(q_D\) as we are learning disentangled representation (independence).
Generator Learning

The generator learning also depends on the discriminator $D$, as it will affect structured code $c$.

In other words, the discriminator provides additional learning signals for generators $G$ to create sentences that are coherent with their designated $c$ values.
Note on Structured Codes $c$

$P(c)$ can be a multinomial distribution. Each code in $c$ denotes a specific attribute that we would like to control.

As pointed out in the last page, $q_E$ and $q_D$ are disentangled.

The training/learning of code $c$ and discriminator(s) $D$ does NOT depend on the standard VAE loss. However,

- It will affect the generation process that is why it appears in the loss of the generator $G$
- We hope the generated sentences have the expected targets as structured attributes $c$
- The optimization of $c$ and $D$ will be described shortly
Challenges of Training on Sequence Data

Sequence text is discrete, but backpropagation requires real-value continuous samples. If you recall the discussion before, “slightly change” does not make sense in text.

Discriminators work with full sequence, not parts of sequences.

The authors use a continuous approximation to replace discrete one-hot vector \( \hat{x}_t \) by softmax probability vector.

\[
\hat{x}_t \sim \text{softmax}(o_t / \tau) \\
\hat{x}_t(a) = \frac{\exp(o_t(a) / \tau)}{\sum_{i=1} \exp(o_t(i) / \tau)}
\]

This is a common trick in learning problems:

- Temperature \( \tau \) is first set to 1 so it is a standard softmax (directly on logits), and
- A low temperature, e.g., \( \tau \to 0^+ \), will emulate the discrete case as the highest logit tends to 1.
- Additional note, \( \tau \to +\infty \) will result in (nearly) all equal probabilities.

In this paper, \( \tau \) is first set to 1 and decrease while training proceeds.
Challenges of Training on Sequence data

Sequence text is discrete, but backpropagation requires real-value continuous samples. If you recall the discussion before, “slightly change” does not make sense in text setting.

Discriminators work with full sequence, not parts of sequences.

The authors use a **continuous approximation** to replace discrete one-hot vector \( \hat{x}_t \) by softmax probability vector.

\[
\hat{x}_t \sim \text{softmax}(o_t / \tau)
\]

\[
\hat{x}_t(a) = \frac{\exp(o_t(a) / \tau)}{\sum_{i=1} \exp(o_t(i) / \tau)}
\]

**Justification [see details in the paper]:**

- This could reduce variance (can be viewed as a smoothed-out approximation) and expedite convergence
- This approximation is only applied on the discriminator side for attribute code learning, **NOT** affecting base sentence generation by VAE
Objective of Learning Structured Code $c$

Continuous approximation + annealing leads to a “soft” generated sentences, denoted as $\tilde{G}_\tau(z, c)$.

As shown in the figure below, it is then fed into the discriminator(s) $D$ to check the fitness of the targeted attributes (generated by $G$).

So the discriminator will also define a loss for Generator $G$ regarding the fitness on targeted attributes:

$$L_{\text{Attr}, c}(\theta_G) = \mathbb{E}_{p(z)p(c)} [\log q_D(c | \tilde{G}_\tau(z, c))]$$

There are two things worth noticing: $p(z)$ and $p(c)$ are disentangled; each structured attribute is controlled by the corresponding code in $c$ and independent to unstructured representation $z$.

It is noted that the discriminator $D$ is not trained/optimized here.
Are $z$ and $c$ really Disentangled?

Even with the assumption of the disentanglement of $z$ and $c$, it is still possible the code in $c$ are entangled with other variables in the latent representation.

If that is the case, varying $c$ will lead to unexpected results on $z$.

The fix is to enforce the independency among $c$ and other attributes by introducing another cross-entropy loss to let these attributes fully captured by $z$. In other words, training the generator so that the unstructured attributes can be recovered from the generated samples.

Unstructured attributes are invariant if $z$ is unchanged—so we could then manipulate $c$!

Similarly, a cross-entropy loss (maximizing likelihood) term is introduced as the constraint for representation disentanglement:

$$L_{Attr,z}(\theta_G) = \mathbb{E}_{p(z)p(c)}[\log q_E(z|\tilde{G}_\tau(z,c))]$$

The key here is to view the encoder of the VAE as a discriminator for $z$ to enforce independence.
Generator Learning Loss

Finally, we could piece three loss term together for the generator $G$:

$$\min_{\theta_G} L_G = L_{VAE} + \lambda_c L_{Attr,c} + \lambda_z L_{Attr,z}$$

1. $L_{VAE}$ is the standard VAE loss we discussed (reconstruction error + KL term for regularizing $p(z)$)
2. $\lambda_c L_{Attr,c}$ measures the finiteness of the generated samples regarding the targeted attributes $c$.
3. $\lambda_z L_{Attr,z}$ measures whether the “irrelevant attributes” can be fully captured by latent representation $z$.

$\lambda_c$ and $\lambda_z$ are the balancing factors to decide the strength of the three terms.

Encoder is simply trained by minimizing the VAE loss: $\min_{\theta_E} L_{VAE}$

The beauty of this design is:

$D$ is the discriminator for the structured code part $c$, and the encoder $E$ is used as the discriminator of unstructured code $z$.

No additional discriminators are needed.
Discriminator Learning

Different from the generator $G$ and encoder $E$, the discriminator(s) $D$ has to be trained in a (semi-)supervised manner:

1. For categorical attributes (e.g., sentiment, tense), the discriminator(s) can be formulated as classifiers.
2. For continuous attributes (e.g., formality), the discriminator(s) can be formulated as probabilistic regressors.

This categorical target variables make VAE training inaccessible for discriminators (continuity issue again).

Another consideration is we would like to use label to entail (strong and accurate) designated semantics (e.g., sentiment).

*Each code in $c$ has a corresponding discriminator $D$. For notation simplicity, we consider there is only one discriminator (although it could have more and it does have more).
Discriminator Learning

Let us first define the parameter of discriminator $D$ as $\theta_D$ and a set of labeled samples are used $X_L = \{(x_L, c_L)\}$

- {“This is a great restaurant!”, positive}
- {“The food here sucks 😞”, negative}
- ...

For different semantics, e.g., sentiment, tense, and formality, separate labels are needed.

$$L_s(\theta_D) = \mathbb{E}_{X_L} [\log q_D(c_L|x_L)]$$

The loss on the supervised part, $L_s(\cdot)$, is as simple as minimizing the cross-entropy as we saw before.
Discriminator Learning

Recall for each structured code we have an independent discriminator.

So different attributes in $c, c_1, c_2, \ldots, c_k$, can be trained individually on different labeled sets.

- Sentiment attribute can be trained on dataset 1
- Tense can be trained with dataset 2
- ...

The sentences do not need to be annotated on all structured codes (sentiment, tense, etc.).

Because this part does not involve direct sentence generation mainly decided by $z$, more choices are possible for the training data, e.g., labeled phrases/words.
Discriminator Learning

Semi-supervised learning:

- Labeled samples are always expensive. Can we leverage the power of VAE to **synthesize training pairs** to train the discriminator?

- That will be cheap to do 😊

We may leverage the generator $G$ to synthesize “noisy” sentence-attribute pairs $(\tilde{x}, c)$: this can be viewed as a data augmentation!

**Challenges**: the quality of the generated samples can vary

**Fix**: Minimum Entropy Regularization (MER) [1, 2]

**Key idea**: in semi-supervised problems, how to use unlabeled data to facilitate supervised learning is non-trivial

- Observation 1: observing $X$ does not inform about $y$, unless one assumes some the relationships between $X$ and $y$

- Observation 2: unlabeled data may be encoded as the prior under Bayesian framework


Discriminator Learning

Unlabeled samples may be useful under certain assumptions [1]:

- Parametric statistics
- Unlabeled data have small class overlap (measured by conditional entropy $H(Y|X)$)
- Under Bayesian frameworks, assumptions can be encoded as priors on the model parameters

In summary, unlabeled data **could be useful** if it has small class overlap. Under Bayesian framework, maximum a posteriori (MAP), can be used to control the weights of unlabeled examples, so it can:

- Help the classification
- Control the harm to classification

Discriminator Learning

To alleviate the issue of unsupervised noisy data and improve the optimization dynamics, MER is used:

$$\mathcal{L}_u(\theta_D) = \mathbb{E}_{p_G(\hat{x} | z, c)p(z)p(c)}[\log q_D(c | \hat{x}) + \beta \mathcal{H}(q_D(c' | \hat{x}))]$$

- $\mathbb{E}[\log q_D(c | \hat{x})]$ measures the cross-entropy of the sample
- $\beta \mathcal{H}(q_D(c' | \hat{x}))$ measures the class overlap over the generated noisy sentences by Shannon entropy, $\beta$ is the balancing factor to control the strength

Intuitively, the loss on the unsupervised part, $\mathcal{L}_u(\cdot)$, encourages the model to have high confidence in predicting targeted attributes.

Recall the loss on the supervised part, $\mathcal{L}_s(\theta_D) = \mathbb{E}_{x_L}[\log q_D(c_L | x_L)]$

The Discriminator $D$ has the following loss for learning from both labeled and synthesized data:

$$\min_{\theta_D} \mathcal{L}_D = \mathcal{L}_s + \lambda_u \mathcal{L}_u,$$

where $\lambda_u$ is the balancing parameter
Training Dynamics

It is obvious training all these parts (VAE, generator, and discriminator) together is not easy, as they depend on each other:

- Generator learning depends on VAE training and the feedback from the discriminator (on structured code c)
- Discriminator learning depends on the output of the generator for synthesizing sentences

Learning procedure:

1. **Initialize VAE** on a large corpus of **unlabeled** text data to minimize the loss in Eq. (4), the standard VAE loss
2. The latent code c is sampled from prior 𝑝(𝑐) for now
3. The full model is then trained by **alternating** the optimization of the generator and the discriminator(s)
Training Dynamics

Combine the **generator (Eq. 8)**, **VAE (Eq. 4)**, and **discriminator (Eq. 11)** learning together, we have the following procedure:

**Algorithm 1 Controlled Generation of Text**

**Input:** A large corpus of unlabeled sentences $\mathcal{X} = \{x\}$
- A few sentence attribute labels $\mathcal{X}_L = \{(x_L, c_L)\}$
- Parameters: $\lambda_c, \lambda_z, \lambda_u, \beta$ - balancing parameters

1: Initialize the base VAE by minimizing Eq.(4) on $\mathcal{X}$ with $c$ sampled from prior $p(c)$
2: repeat
3: Train the discriminator $D$ by Eq.(11)
4: Train the generator $G$ and the encoder $E$ by Eq.(8) and minimizing Eq.(4), respectively.
5: until convergence

**Output:** Sentence generator $G$ conditioned on disentangled representation $(z, c)$

**Eq. (4):** $\mathcal{L}_{VAE}(\theta_G, \theta_E; x) = -KL(q_E(z|x) || p(z) +$

**Eq. (11):** $\min_{\theta_D} \mathcal{L}_D = \mathcal{L}_S + \lambda_u \mathcal{L}_u$

**Eq. (8):** $\min_{\theta_G} \mathcal{L}_G = \mathcal{L}_{VAE} + \lambda_c \mathcal{L}_{Attr,c} + \lambda_z \mathcal{L}_{Attr,z}$
Training Dynamics

The model can be viewed as a VAE with extended wake-sleep method.

- **Wake-sleep [1]:** alternating optimization VAE and discriminator (can be viewed as competitively optimization)
- An extension of the previous work [2]. Refer for details here.

**Short summary:**

- Black arrows indicate generation and inference
- Red arrows denote gradient propagation
- Generator $G$ synthesizes noisy samples for discriminator $D$ to learn (sleep)
- Generator $G$ samples code $c$ from discriminator distribution $q_D(c|x)$ (wake)

With this complex training dynamics, the limitation of discrete text data (on gradient descent), the challenge of disentangled representations, and the efficient mutual sampling/dependency can be overcome.

---


Experiment

Sentence length: \( \leq 15 \) words

Controlled semantics: (i) sentiment: positive or negative; (ii) tense (“past”, “present”, “future”)

Dataset for generative model (unlabeled): IMDB text corpus (each at most 15 words); 1.4M sentences; 16k vocabulary size

Dataset for sentiment control (labeled):
- Stanford Sentiment Treebank-2 (SST-full): 2837 training samples and 1821 test samples
- Stanford Sentiment Treebank-2 (SST-small): sample 250 labeled sentences from SST-full
- Lexicon: word level sentiment control: 2700 words with sentiment; test on SST-full test set
- IMDB: randomly selected positive and negative reviews from IMDB corpus; 5K/1K/10K sentences for train/dev/test

Dataset for tense control (labeled):
- Timebank: 5,250 labeled words and phrases; “was”, “will be”, time expressions like “in the future”
Experiment

**Generator:** single-layer LSTM RNN with input/hidden dimension of 300; max sample length 15 (max words per sentences)

**Encoder:** single-layer LSTM RNN with input/hidden dimension of 300; max sample length 15 (max words per sentences)

**Discriminators:** ConvNets; see supplements for details [Hu et.al. 2017]

Other details, such like balancing parameters, annealing temperature could be found in the supplement as well.
Results – Sentiment Classification

Automatic Accuracy by Sentiment Classifier [1], denoted as $\psi_S$ (>90% accuracy on SST-full)

- Generate sentence by specifying sentiments on code $c$
- Evaluate the sentiment consistency (accuracy) of the generated sentences by $\psi_S$

The table shows the accuracy on 30K generated sentences:

- Compared with semi-supervised S-VAE [2] (no structured code $c$ and the discriminator)
- Using SST-full, SST-small, and Lexicon (word-level label) for training the discriminator

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SST-full</td>
<td>SST-small</td>
<td>Lexicon</td>
</tr>
<tr>
<td>S-VAE</td>
<td>0.822</td>
<td>0.679</td>
<td>0.660</td>
</tr>
<tr>
<td>Ours</td>
<td>0.851</td>
<td>0.707</td>
<td>0.701</td>
</tr>
</tbody>
</table>


Results – Sentiment Classification

Observations:
- Improved accuracy on all test datasets; (at least sentiment) representations are disentangled
- Even word-level label can train model effectively—the knowledge is successfully transferred
- Little to no supervision is needed as SST-small only consists of 250 labeled samples

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SST-full</td>
<td>SST-small</td>
<td>Lexicon</td>
</tr>
<tr>
<td>S-VAE</td>
<td>0.822</td>
<td>0.679</td>
<td>0.660</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>0.851</strong></td>
<td><strong>0.707</strong></td>
<td><strong>0.701</strong></td>
</tr>
</tbody>
</table>
Results – Quality of the Generated Sentences

It is hard to measure the quality of the generated sentences—the most straightforward but tedious way is to manually check them!

An implicit way to assess the generation quality can be:
- Test whether the generated sentences used as additional training samples can improve the sentiment classifier $\psi_S$
- If so, the quality of the generated sentences and the sentiment label is good
- This is an auxiliary measure of the sentence quality (high quality sentences + accurate sentiment labels $\Rightarrow$ better sentiment classifier)

Experiment Setting:
- Generate sentences and the corresponding sentiment labels by manipulating code $c$
- Use the $(\hat{x}, c_s)$ pairs as additional data to train $\psi_S$
- Measure $\psi_S$’ performance on the original test set
Results – Quality of the Generated Sentences

To bring more insights, **multiple generation variations** are introduced:

- **Std**: trained on supervised data only, no noisy dataset
- **H-Reg**: using minimum entropy regularization on the generated sentences
- **Ours**: the proposed framework in this paper
- **S-VAE [1]**: semi-supervised VAE

**Observations**:

- Using generated samples for discriminator learning can be useful
- Minimum entropy regularization shows positive effect on sentence generation
- The proposed method consistently outperform

Disentangled Representation

The effect of **independency constraint**: \( L_{\text{Attr}}(\theta_G) = \mathbb{E}_{p(z)p(c)}[\log q_E(z|G_T(z,c))] \)

<table>
<thead>
<tr>
<th>w/ independency constraint</th>
<th>w/o independency constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>the film is strictly routine!</td>
<td>the acting is bad.</td>
</tr>
<tr>
<td>the film is full of imagination</td>
<td>the movie is so much fun.</td>
</tr>
<tr>
<td>after watching this movie, i felt disappointed.</td>
<td>none of this is very original.</td>
</tr>
<tr>
<td>after seeing this film, i’m a fan.</td>
<td>highly recommended viewing for its courage, and ideas.</td>
</tr>
<tr>
<td>the acting is uniformly bad either.</td>
<td>too bland</td>
</tr>
<tr>
<td>the performances are uniformly good.</td>
<td>highly watchable</td>
</tr>
<tr>
<td>this is just awful.</td>
<td>i can analyze this movie without more than three words.</td>
</tr>
<tr>
<td>this is pure genius.</td>
<td>i highly recommend this film to anyone who appreciates music.</td>
</tr>
</tbody>
</table>

**Observations:**

- With the independency constraint, varying sentiment codes would not change the structure of the sentence drastically.
- In comparison, the pairs on the right-hand result in obviously different structures, implying varying \( c \) affects unstructured attributes.
Tense Control via Structured code $c$

<table>
<thead>
<tr>
<th>Varying the code of tense</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I thought the movie was too bland and too much</td>
<td>this was one of the outstanding thrillers of the last decade</td>
</tr>
<tr>
<td>I guess the movie is too bland and too much</td>
<td>this is one of the outstanding thrillers of the all time</td>
</tr>
<tr>
<td>I guess the film will have been too bland</td>
<td>this will be one of the great thrillers of the all time</td>
</tr>
</tbody>
</table>

*Table 3.* Each triple of sentences is generated by varying the tense code while fixing the sentiment code and $z$. 
# Sentiment + Tense Examples

<table>
<thead>
<tr>
<th>Varying the unstructured code $z$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(&quot;negative&quot;, &quot;past&quot;)</td>
<td>(&quot;positive&quot;, &quot;past&quot;)</td>
</tr>
<tr>
<td>the acting was also kind of hit or miss .</td>
<td>his acting was impeccable</td>
</tr>
<tr>
<td>i wish i ´d never seen it</td>
<td>this was spectacular, i saw it in theaters twice</td>
</tr>
<tr>
<td>by the end i was so lost i just did n’t care anymore</td>
<td>it was a lot of fun</td>
</tr>
<tr>
<td>(&quot;negative&quot;, &quot;present&quot;)</td>
<td>(&quot;positive&quot;, &quot;present&quot;)</td>
</tr>
<tr>
<td>the movie is very close to the show in plot and characters</td>
<td>this is one of the better dance films</td>
</tr>
<tr>
<td>the era seems impossibly distant</td>
<td>i ´ve always been a big fan of the smart dialogue .</td>
</tr>
<tr>
<td>i think by the end of the film , it has confused itself</td>
<td>i recommend you go see this, especially if you hurt</td>
</tr>
<tr>
<td>(&quot;negative&quot;, &quot;future&quot;)</td>
<td>(&quot;positive&quot;, &quot;future&quot;)</td>
</tr>
<tr>
<td>i wo n’t watch the movie</td>
<td>i hope he ´ll make more movies in the future</td>
</tr>
<tr>
<td>and that would be devastating !</td>
<td>i will definitely be buying this on dvd</td>
</tr>
<tr>
<td>i wo n’t get into the story because there really is n’t one</td>
<td>you will be thinking about it afterwards, i promise you</td>
</tr>
</tbody>
</table>

Table 4. Samples by varying the unstructured code $z$ given sentiment ("positive"/"negative") and tense ("past"/"present"/"future") code.
## Failure Cases

<table>
<thead>
<tr>
<th>Failure cases</th>
<th>Outcome cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>the plot is not so original</td>
<td>it doesn’t get any better the other dance movies</td>
</tr>
<tr>
<td>the plot weaves us into <code>&lt;unk&gt;</code></td>
<td>it doesn’t reach them, but the stories look</td>
</tr>
<tr>
<td>he is a horrible actor ’s most part</td>
<td>i just think so</td>
</tr>
<tr>
<td>he ’s a better actor than a standup</td>
<td>i just think !</td>
</tr>
</tbody>
</table>

*Table 5. Failure cases when varying sentiment code with other codes fixed.*
Conclusions

Summary: A deep generative model for generating controllable text (sentiment and tense) by manipulating specific codes.

Highlights:

- **Disentangled representation** leads to controllability and interpretability
- Semi-supervised framework requires **little to no supervision** for learning structured code $c$
- **Intricate training dynamics** (VAE + discriminators + extended wake-sleep framework)
  - encoder can also be viewed as a discriminator for unstructured part $z$
- **Continuous approximation** for discrete text with annealing (replace one-hot vector by probabilistic vector)
Limitations & Future Directions

**Limitations:**
- The length of the generated sentences cannot exceed 15 words
- The controllable attributes are limited
- Complex training dynamics and unpredictable results (see failure cases)

**Future Directions:**
- How to work with longer sequences?
- How to control more attributes with fine-grained structures?

**Full Presentation Video** by Zhiting Hu at ICML 2017: [https://vimeo.com/238222247](https://vimeo.com/238222247)
Critics

Along with “Adversarial Generation of Natural Language” [1], this paper is challenged by Yoav Goldberg (Professor at Bar Ilan University) for overclaiming contributions. See more information here [2].

“This is, again, both over-selling and grossly disrespecting language.

For some context, the average sentence length in a Wikipedia corpus I have around here is 19 tokens. Many are way longer than that. Sentiment is much more nuanced than positive or negative. And English has a somewhat more elaborate time system than past, present, and future.

So ok, Hu et al created an actor-critic-VAE framework with some minimal control options, and made it work with some short text fragments. Is “Controllable Text Generation” really the most descriptive title here? (Although, to be fair, they did not say Natural Language in the title, only in the abstract, so that’s something I guess).”

Summary

Go over some basic NLG techniques


Discuss how to generate text from a VAE in an uncontrollable manner


Demonstrate the possibility of decoding the latent space


Present the idea of controllable text generation