Generating Sentences by Editing Prototypes

K. Guu$^2$, T.B. Hashimoto$^{1,2}$, Y. Oren$^1$, P. Liang$^{1,2}$

$^1$Department of Computer Science
Stanford University

$^2$Department of Statistics
Stanford University

Reviewed by: Bill Zhang
University of Virginia
https://qdata.github.io/deep2Read/
Outline

Introduction

Problem Statement

Approach

Experiments

Summary
Introduction
Basic Premise and Motivation

- Current state-of-the-art sentence generators generate from scratch
  - Tend to favor generic, short statements
  - More complex sentences sacrifice grammar
- Prototype-then-edit model inspired by drafting for papers
- Start from a high quality sentence with no bias towards short or grammatically incorrect statements and edit with an "edit vector"
- Compare performance with generate from scratch models through two metrics: language generation quality and semantic properties
Problem Statement
Primary Goals

- Learn a generative model of sentences
  - Select a prototype sentence, $x'$, from a training set of sentences, $X$
  - Select an edit vector, $z$, from a distribution of edit vectors, $p(z)$
  - Select final sentence from distribution of sentences resulting from applying $z$ to $x'$ ($p_{edit}(x|x', z)$)

- Likelihood of a sentence
  - $p(x) = \Sigma_{x' \in X} p(x|x')p(x')$
  - $p(x|x') = \int_z p_{edit}(x|x', z)p(z)dz$

- Process chosen because sentences in a large data set tend to be minor transformations of other sentences
Problem Statement

Secondary Goals

- Capture semantic properties
  - Each edit should only slightly change semantics of sentence, more edits should accumulate change
  - Applying the same edit vector to different sentences should yield similar semantic changes
Approach

Approximations

- Previous equations expensive to calculate and maximize
- \[ p(x) = \sum_{x' \in X} p(x|x') p(x') \]
  - Only sum across \( x' \) lexically similar to \( x \), as measured by Jaccard Distance, \( d_J \)
  - \( N(x) = \{x' \in X | d_J(x, x') < 0.5\} \)
- \[ p(x|x') = \int_z p_{edit}(x|x', z) p(z) dz \]
  - Generate lower bound by modeling \( z \) with a variational autoencoder, which admits tractable inference via the Evidence Lower Bound (ELBO)
- Jensen’s Inequality used in approximations
Approach
Approximation Derivations

\[
\log p(x) \geq \log \left( \sum_{x' \in N(x)} p(x') p(x|x') \right) \\
\geq \log \left( \sum_{x' \in N(x)} |N(x)|^{-1} p(x|x') \right) - \log |X| \\
\geq |N(x)|^{-1} \sum_{x' \in N(x)} \log p(x|x') - \log |X|
\]

\[
p(x') = \frac{1}{|X|} \Rightarrow \sum_{x' \in N(x)} p(x') p(x|x') = \frac{\sum_{x' \in N(x)} p(x|x')}{|X|}
\]

\[
\Rightarrow p(x) \geq \frac{\sum_{x' \in N(x)} |N(x)|^{-1} p(x|x')}{|X|}
\]

Therefore, treating \(|N(x)|\) as a constant and summing over all \(x \in X\), we get the objective function:

\[
L_{Lex} = \sum_{x \in X} \sum_{x' \in N(x)} \log p(x|x')
\]
Approach
Approximation Derivations

\[ p(x|x') = \int_z p_{\text{edit}}(x|x', z) p(z) dz \]

\[ \log p(x|x') = \log \int_z p_{\text{edit}}(x|x', z) p(z) dz \]

\[ = \log \int_z \frac{p_{\text{edit}}(x|x', z) p(z)}{q(z|x, x')} q(z|x, x') dz \]

\[ = \log E_q \left[ \frac{p_{\text{edit}}(x|x', z) p(z)}{q(z|x, x')} \right] \]

\[ \geq E_q [\log p_{\text{edit}}(x|x', z)] + E_q [\log p(z)] - E_q [\log q(z|x, x')] \]
Approach
Approximation Derivations

\[ D_{KL}(q(z|x, x') \| p(z)) = E_q[\log \frac{q(z|x, x')}{p(z)}] \]
\[ = E_q[\log q(z|x, x')] - E_q[\log p(z)] \]

\[ \log p(x|x') \geq I(x, x') = E_q[\log p_{edit}(x|x', z)] - D_{KL}(q(z|x, x') \| p(z)) \]
Therefore, the final objective function is:

\[ L_{Lex} \geq L_{ELBO} = \sum_{x \in X} \sum_{x' \in N(x)} I(x, x') \]
Approach

Approximation Definitions

- **Neural Editor:** \( p_{edit}(x|x', z) \)
  - Seq-to-seq model with attention, concatenate \( z \) to decoder input
- **Edit prior:** \( p(z) \)
  - \( z_{\text{norm}} \sim \text{Unif}(0, 10) \)
  - \( z_{\text{dir}} \sim \text{vMF}(0) \)
- **Approximate Edit Posterior:** \( q(z|x, x') \)
  - Want edit vector to represent word insertions and deletions
  - Let \( I = x \setminus x' \) be word insertions, \( D = x' \setminus x \) be word deletions
  - \( f(x, x') = \sum_{w \in I} \phi(w) \oplus \sum_{w \in D} \phi(w) \), but need to add entropy
    - Add uniform noise to \( \tilde{f}_{\text{norm}} \), which has been truncated to 10
    - Add vMF noise to \( f_{\text{dir}} \)
Experiments

Datasets

- Yelp review corpus
- One Billion Word Language Model Benchmark
- Replaced named entities and replaced rare tokens from datasets with special token
Experiments
Approaches Compared

- Neural Editor (Proposed)
- NLM (Standard generation from scratch)
- KN5 (5-gram language model)
- Memorization
- SVAE (Sentence variational autoencoder)
Experiments
Results: Perplexity

- Proximity to prototype is chief determinant of perplexity performance
  - Majority of sentences in Yelp testing set (70%) have similar structure to a training set sentence

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity (Yelp)</th>
<th>Perplexity (BILLIONWORD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN5</td>
<td>56.546</td>
<td>78.361</td>
</tr>
<tr>
<td>KN5+MEMORIZATION</td>
<td>55.180</td>
<td>73.468</td>
</tr>
<tr>
<td>NLM</td>
<td>40.174</td>
<td>55.146</td>
</tr>
<tr>
<td>NLM+MEMORIZATION</td>
<td>38.980</td>
<td>50.969</td>
</tr>
<tr>
<td>NLM+KN5</td>
<td>38.149</td>
<td>47.472</td>
</tr>
<tr>
<td>NEURALEDITOR(κ = 0)</td>
<td>27.600</td>
<td>48.755</td>
</tr>
<tr>
<td>NEURALEDITOR(κ = 25)</td>
<td><strong>27.480</strong></td>
<td>48.921</td>
</tr>
</tbody>
</table>
Experiments
Results: Human Evaluation

- NeuralEditor is on par with best tuned NLM in terms of grammaticality and plausibility, while also having larger diversity.
- Initial prototypes already inject sentence diversity without having to increase the temperature of the model substantially and thus preserve grammaticality and plausibility.
  - High temperature NLMs have more diversity, but less grammaticality and plausibility; low temperature NLMs have the opposite problem.
- Higher temperature NeuralEdit results in more deviation from training set.
Experiments

Results: Semantics

- Semantic smoothness
  - NeuralEditor: Randomly select prototype sentence and repeatedly apply edits drawn from edit distribution to produce sequence
  - SVAE: Randomly select prototype sentence and repeatedly encode and decode again after adding random Gaussian with variance 0.4 to produce sequence
  - NeuralEditor frequently paraphrases while SVAE often repeats sentences exactly or generates unrelated sentences

- Smoothly controlling sentences
  - Generate more sequences like before and select sequence for each method which has greatest likelihood to match desired attributes
  - NeuralEditor tends to have less tradeoff of semantic similarity for attribute satisfaction
Experiments
Results: Semantics

- Consistent edit behavior
  - Take sentences $x_1$ and $x_2$ with some semantic relation $r$; given $y_1$, find $y_2$ such that $y_1$ and $y_2$ also have relation $r$
  - Approximate edit vector from $x_1$ to $x_2$ as $\hat{z} = f(x_1, x_2)$
  - Apply $\hat{z}$ to $y_1$ and evaluate top $k$ candidates of resulting distribution to see if any match $y_2$
  - Sentence analogies are generated from single replacement from existing word analogies
  - SVAE had close to 0 accuracy, so instead compared to baseline of randomly sampling $\hat{z}$
  - Performance on par with models for word-level analogies
The prototype-then-edit model allows for grammaticality and plausibility without the cost of sentence structure diversity.

Compared to other language models, the prototype-then-edit model is much better at preserving semantics and maintaining semantic smoothness and is on par with current state-of-the-art in perplexity.

The prototype-then-edit model typically only allows for small deviations from training set, which means it performs poorly if training set is too different from testing set.
References

- http://mathworld.wolfram.com/JensensInequality.html