Conditional Prototype Editing for Abstractive Summarization

Demi Guo Kevin Liu Artidoro Pagnoni

Abstract

We explore a new method for abstractive summarization that first samples a prototype summary from a training corpus, then edits it based on the input text into a target summary. Our model is an extension of the technique to generate random sentences by editing prototypes (Guu et al., 2017). We propose a truly generative approach, unlike the original paper, combining the traditional recurrent Neural Language Models with the prototype-then-edit approach.

1. Introduction

Summarization is a core challenge in neural machine understanding. The objective is to compress the information from an input text into a text capturing the meaning of the original text. There are two main approaches to machine summarization. Extractive methods identify and select the most important segments of the input text and build the output text by concatenating them. Abstractive methods, build an internal semantic representation of the input text and generate an output text that tries to reflect the summary that is closer to what a human might express. Generating sentences is thus a key building block of abstractive summarization. It is also central to other natural language processing tasks such as translation, and dialogue.

In this work we focus on abstractive summarization, in particular on exploring a new method for text generation applied to the task of sentence-level summarization. The traditional approach to sentence generation uses recurrent Neural Language Models (NLMs) which generate from scratch left-to-right. NLMs have been shown to lead to a number of issues. Namely, they favor generic sentences and naive strategies to increase diversity often compromise on grammatical correctness (Li et al., 2016). This indicates that language models that generate from scratch left-to-right may not be able to fully capture the diversity of sentences. Furthermore, humans generally do not seem to write perfectly in one go. Rather, they tend make edits through multiple drafts. Hence, it may make sense to have a model of sentence generation that does the same. (Guu et al., 2017) proposes a model for generating sentences based on editing prototypes sampled from the training corpus.

Our contribution consists in the exploration of a new generative model of sentences for abstractive summarization. The proposed model incorporates a prototype-editing technique and combines it with the traditional Neural Language Models. In this paper, we begin by outlining the proposed generative model. We then present the methods for inference that we use and the specific representation of the terms in the probabilistic model. Finally, we evaluate the summarization model with the proposed sentence generation method comparing it to the NLM baseline model.

2. Related Work

There has been a lot of work in neural summarization models. (Rush et al., 2015) first applied modern neural networks with an attention mechanism to abstractive text summarization and achieved state-of-the-art performance on Gigaword and DUC-2004. Recently, there has been a lot of innovative augmentations on the model, such as pointing and coverage technique (See et al., 2017), selective encoding mechanism (Zhou et al., 2017), deep recurrent generative decoder (Li et al., 2017), and hierarchical networks (Nallapati et al., 2016). Our work differs from all of their work as we model summary generation process as editing prototype summaries.

(Guu et al., 2017) proposes a prototype editing sentence generation framework on which our model is based. They model the sentence generation process with two parts. First, sample a prototype and an edit vector, then use the prototype and edit vector to generate the output sentence. However, their model framework only works on the unconditional language generation case. Our work extends their work as we develop a framework for conditional language generation task, specifically abstractive summarization. Instead of using a uniform prototype selection process, we select prototype conditionally based on a trained attention-based sequence-to-sequence neural network. In addition, their model is not fully generative because the target sentence is required as an input to parts of their model (to their neural nets). In contrast, our model is fully generative.

In another related work, (Gu et al., 2017) proposes a search-engine guided non-parametric neural machine translation. They uses an off-the-shelf, black-box search engine and fuzzy matching score to retrieve pairs of similar data samples . Then, they use a translation memory enhanced Neural Machine Translation (NMT) model to incorporate retrieved samples in translation process. Our model differs from their work mainly because of following two reasons: (1) They treat retrieval process as a non-trainable black-box, whereas we model our prototype retrieval process as a probabilistic model approximated using a trainable neural network. (2) The way in which retrieved prototypes are incorporated into the generation process differs. They store the retrieved samples into a key-value memory. Then, at each timestep of the generation process, they incorporate one value from the key-value memory based on some attention-based selection. On the other hand, we assign each prototype a probability. The influence of each prototype on our sentence generation is based on this probability.

3. Model

We extend the prototype editing model of (Guu et al., 2017) to the conditional case. Our primary goal is to learn a generative model of sentences given a condition. In particular we model sentence generation as a prototype-then-edit process. There are two steps in the generative process:

1. **Prototype Selector**: Given a training corpus $S = \{(x, y)\}$ of pairs of inputs and outputs, and an input x, we sample a prototype sentence $y' \sim p(y'|x)$ from the corpus.

2. **Neural editor**: We draw an edit vector from $p(z|y', x)$ which will encode the type of edit. Then we draw the final sentence from $p_{edit}(y|y', z, x)$, which takes as argument the prototype sentence y', the edit vector z, and the input x. The dependence on the input x is mean to furnish additional information to perform the correct edits.

Under this model the likelihood of a sentence y given an input x is:

$$p(y|x) = \sum_{y' \in Y} p(y|y', x)p(y'|x) \quad (1)$$

$$p(y|y', x) = \int_z p_{edit}(y|y', z, x)p(z|y', x)dz \quad (2)$$

Where both y' and z are latent variables. Just as the model in paper, these are computationally difficult or intractable to compute, and thus must approximated.

This model is the extension to the conditional setting of the model proposed by (Guu et al., 2017). The formulation of text generation as a prototype then edit process is based on the observation that many sentences in large corpora can be represented as minor transformations from other sentences. For example, in the OpenNMT summarization dataset, over 70% of the summaries lie within 0.5 Jaccard distance from another summary. Thus, a neural editor applying lexical transformation to prototype sentences should be an effective generative model for a very large part of the dataset.

4. Inference

Ideally we would train our prototype selector and neural editor directly by maximizing the marginal likelihood defined by Equations 1 and 2. However, the exact likelihood is an intractable quantity as it requires to compute a very costly sum over all latent prototype sentences and an integration over all latent edit vectors. Following the approach by (Guu et al., 2017) we use two approximation methods to overcome these challenges:

1. We approximate the sum over all prototypes y' by the sum over prototypes in a neighborhood $\mathcal{N}(y)$ of sentences lexically similar to y.

2. We approximate the integration over the latent variable z with the evidence lower bound (ELBO), from a conditional variational autoencoder, making inference possible through SGD and sampling.

4.1. Approximate sum on prototypes y'

In Equation 1, the probability of generating a summary y is modeled as the sum of the probabilities of reaching y by editing prototypes $y' \in S$. However, most prototypes should be highly unlikely to be transformed into y as they will be significantly different from the target sentence y. We thus approximate the sum over all possible prototypes y' with the sum over all sentences that are lexically similar to y. We refer to the set of sentences $\mathcal{N}(y)$ that are lexically similar to y as the neighborhood of y.

It is important to note that the definition of neighborhood used by (Guu et al., 2017), which involves a direct dependence on the target sentence y, does not allow for a truly generative model. Their model requires the knowledge of the desired output y in order to construct a significant piece of the model which is meant to generate y. We therefore propose a definition of the neighborhood that does imply a dependence on the output y. We approximate the neighborhood as the summaries that share a lexically similar input text with y, or formally $\mathcal{N}(y) = \{y' : (x', y') \in$

S and $d_J(x, x') < 0.65\}$, where x is the input text.

Then the log likelihood of the prototype-then-edit process is lower-bounded by:

$$\log p(y|x) \geq \log \left(\sum_{y' \in N(y)} p(y'|x) p(y|y', x) \right)$$
$$\geq \sum_{y' \in N(y)} \log p(y'|x) + \log p(y|y', x) \quad (3)$$

Where the second step comes from the Jensen's inequality. Thus, the objective of our training becomes maximizing the log likelihood:

$$\mathcal{L}_{LEX} = \sum_{x,y} \sum_{y' \in N(y)} \log p(y'|x) + \log p(y|y', x) \quad (4)$$

4.2. Approximate integration on edit vectors z'

To approximate the integration over the latent edit vector z we use the evidence lower bound (ELBO). Equation 2 is then lower bounded by:

$$\log p(y|y', x) = \log \int_z p_{edit}(y|y', x, z) p(z|x, y') dz$$
$$\geq \mathbb{E}_{z \in q(z|y', x)} [\log p_{edit}(y|y', z, x) \frac{p(z)}{q(z|y', x)}]$$
$$= \mathbb{E}_{z \in q(z|y', x)} [\log p_{edit}(y|y', z, x)]$$
$$- KL(q(z|y', x)||p(z))$$
$$= l(y, y', x)$$

The important terms in $l(y, y', x)$ are the edit prior $p(z)$, the approximate edit posterior $q(z|y', x)$, and the neural editor $p_{edit}(y|y', z, x)$. Combining the ELBO with Equation 5, the objective becomes the maximization:

$$\mathcal{L}_{ELBO} = \sum_{x,y} \sum_{y' \in N(y)} \log p(y'|x) + l(y, y', x) \quad (5)$$

The above objective is a lower bound to the likelihood of a corpus S, and maximizing this lowerbound consists in maximizing the likelihood of the corpus. $\mathcal{L}_{ELBO}$ is maximized over the parameters of the edit posterior q and of the neural encoder p_{edit}. With the introduction of the approximate posterior q, we train $p(y|y', x)$ as a conditional variational autoencoder (C-VAE). In particular, the maximization is done via SGD with Monte Carlo sampling to approximate the gradients of the expectation term. We follow the approach described by (Guu et al., 2017) based on the work of (Kingma & Welling, 2013).

4.3. Representation of the model components

We define bellow how we model each term in the expression of the objective from Equation (5).

Neural Editor $p_{edit}(y|y', x, z)$**:**
The neural editor is the core component of the model. As in the original paper, it is implemented as a left-to-right sequence-to-sequence model with attention (though now both on y' and x), with an encoder-decoder architecture similar to (Wu et al., 2016).

We extend this part of the model by conditioning the neural editor on the original input x in addition to the prototype and edit vector. As a result, this model can be interpreted as a traditional summarization model guided by a prototype and edit vector. We made this design choice for two main reasons. First, basic methods of constructing an edit vector, such as the concatenation of sums of word vectors of inserted and deleted words, are imperfect and likely not very effective in capturing all the information needed to make the proper edits. In addition, since we are attempting to make the model fully generative (and thus cannot condition on y as done in the paper), the edit vector is even one step further removed (taking difference between x and y' as opposed to taking the difference between y and y'). Second, the conditioning on x allows us to combine and evaluate a combination of a pure edit model and a traditional summarization model and ideally determine the extent of which providing a prototype sentence helps in the content diversity vs grammaticality trade-off that in part motivated the (Guu et al., 2017) paper.

Edit Prior $p(z)$**:**
As in the original paper, z is sampled from the prior by drawing a random magnitude $z_{norm} \sim$ Unif$(0, 10)$ and then draw a random direction $z_{dir} \sim$ vMF(0) where vMF(0) is uniform over the unit sphere. Then set $z = z_{norm} z_{dir}$. Note that it is important that the edit prior is of this form, because along with the vMF noise of the approximate edit posterior, the KL term in the above lower bound expansion has a closed form expression independent of z and thus can be dropped while training.

Prototype Selector $p(y'|x)$:
The prototype selector serves the purpose of selecting a prototype from the training corpus given x. In the original paper, it is assumed that prototypes are sampled uniformly. In a conditional case, however, we can no longer randomly choose prototypes independently of x. The prototype selector must judge how compatible any given y' is as a prototype for x and becomes a key component of the conditional edit model. In our headline generation task, $p(y'|x)$ will represent a document-headline relevancy score. We hypothesize that $p(y'|x)$ can be very similar to a summarization model and use a NLM to approximate $p(y'|x)$. Since y' is discrete and there are a lot of possible y', in order to approximate such probability distribution, we propose the following ways:

1. We could model $p(y'|x)$ with a traditional sequence-

to-sequence neural language model with attention where we evaluate the likelihood of sentence y' given input x. In this case, the sample space will be all sentences.

2. We could model $p(y'|x)$ as a MLP where the last layer will have a list of scores for each y'. Usually, we would do a softmax over all y' in the last layer to get $p(y'|x)$. However, because there are so many possible y', it will be very expensive to compute. Thus, we would only take the softmax over $y' \in N(y)$.

3. Each neural net will take a single y' and x as input, and produce a raw score $score(y', x)$. Then, we do a softmax over all $y' \in N(y)$. To avoid this to become too expensive, we will also set a limit on the size of $|N(y)|$.

In our current implementation, we are using the first method. We use a model similar to the neural editor, an encoder-decoder architecture with attention, which takes x, y' as input.

Approximate Edit Posterior $q(z|y', x)$:
The term approximate edit posterior refers to the fact that the ELBO bound is tight when q matches the true posterior, which is the best estimate of the distribution over the edit vector z given the prototype y' and the input text x. Following the work of (Guu et al., 2017), we model the edit vector z as a noisy version of the concatenation of the sum of word embeddings from words added and words deleted in the prototype to reach the input text. More formally, z represented by $f(y', x)$ with addition of noise where:

$$f(y', x) = \sum_{w \in I} \Phi(w) \oplus \sum_{w \in D} \Phi(w) \tag{6}$$

Where $I = x \backslash y'$, $D = y' \backslash x$, $\Phi(w)$ is the word vector for w, and $\oplus$ denotes the concatenation of two vectors. Here again, since our goal is to have a truly generative model for sentences given an input, contrary to (Guu et al., 2017) we model the the edit posterior independent of the target output y. The term q is the encoder part of the conditional VAE, it therefore cannot deterministically output f. Without any entropy the KL term in the objective (Equation 5) would be infinite, which is a well known property of VAE models. The interesting contribution by (Guu et al., 2017) is the introduction of a specific type of noise which makes the KL term be independent have a closed form expression independent of the edit vector z. In particular, the approximate edit posterior q perturbs the norm of f with uniform noise and the direction of f by adding von-Mises Fisher noise. The von-Mises-Fisher distribution is a distribution over vectors with unit norm, such that the log likelihood of a vector decays linearly with the cosine similarity to the mean μ of the distribution. The linearity coefficient is κ. Formally:

$$\begin{aligned} q(z_{dir}|y', x) &= \text{vMF}(z_{dir}; f_{dir}, \kappa) \\ &\propto \exp(\kappa z_{dir}^\top f_{dir}) \\ q(z_{norm}|y', x) &= \text{Unif}(z_{norm}; [\tilde{f}_{norm}, \tilde{f}_{norm} + \epsilon]) \end{aligned}$$

Where $\tilde{f}_{norm}$ the truncated norm of f and the resulting $z = z_{dir} z_{norm}$. This distribution is very well suited to be used in a VAE as the KL term has a closed form expression independent on z. This property comes from the use of vMF noise with uniform prior. The KL term between a uniform distribution over a sphere (prior) and a vMF with a mean specifying a direction (approx. posterior), only depends on the precision coefficient κ , and not the direction of the mean z_{dir}. The same trick can be used for the norm of z. Therefore the KL term becomes in the objective of Equation (5) becomes:

$$KL(q(z|y', x) \| p(z)) = KL(\text{vMF}(\kappa) \| \text{vMF}(0)) \tag{7}$$

This makes the parameters of the approximate prior: (Φ, κ, ϵ), where κ and ϵ are hyperparameters. We use the same value used by (Guu et al., 2017) for these parameters.

5. Experiments

Our model was build on top of the code base accompanying the paper by (Guu et al., 2017). We implemented from scratch the prototype selector, added attention on the input text in the neural editor, adapted the model to the conditional setting where there is both an input text and a target summary, and made the model truly generative by changing various parts. Furthermore, we made these changes configurable in such a way as to evaluate the effect of each one of the components of the model independently. Finally, we devised an extensive set of experiments to evaluate the performance of our model compared to the baseline of NLMs for summarization. The evaluation of the generated summaries uses the BLEU metric, which is commonly adopted in summarization. Since this is a fully-generative model, we select the prototype by choosing the one that gives the highest likelihood when combining the component from the prototype selector and the editor. This is very slow because it essentially requires to generate several summaries to then only pick the best one.

The implementation of our model and the instructions on how to run it can be accessed at https://github.com/demiguo/conditional-edit-prototype.

5.1. Dataset

The dataset used for experiments is a pre-processed version of the summarization data provided by OpenNMT of the form of article - prototype pairs. A set of 1 million article text-title pairs is taken as the training corpus that serves as the set of available prototypes. Then, training, validation, and test datasets are generated by matching each candidate sentence with (up to 10) candidate prototypes from the prototype training corpus. Matchings are done with the criteria of Jaccard distance between article texts (x and x') being greater than a specified threshold (0.5). Getting neighborhoods defined by Jaccard distance thresholds can be done efficiently with locality sensitive hashing (LSH) and minhashing. Note that for many articles in the OpenNMT dataset, we are unable to find candidate prototypes with texts of Jaccard distance greater than 0.5 away. Consequently, our processed dataset is somewhat biased in that only articles with sufficiently close neighbors (as measured by Jaccard distance) are included. With a Jaccard threshold of 0.5, a prototype can be found for approximately 70-80% of the data.

5.2. Summarization Baseline

We built and trained a recurrent Neural Language Model as a baseline for our model. This model is a sequence-to-sequence model with attention on the input and a beam search decoder that selects the most likely sequence. This model uses the same encoder and decoder structure with attention as our prototype selector and neural editor. This is meant to allow a fair comparison of the proposed model.

5.3. Full Model

This is simply the full model as described in the previous section. We include the prototype selector, the neural editor with attention on the input text, and the edit vector.

5.4. Randomized Prototypes

This experiment is run to verify that the neural editor considers the provided prototype when generating the output summary. The input text x is given to the neural editor along with random prototypes y'. To run this experiment we generated a data set that matches text - summary pairs (x, y) with random prototype pairs (x', y'). We tried two variants of this experiment. In the first, the model is trained with good prototypes and is evaluated on data with random prototypes. In the second, the model is both trained and evaluated on data with random prototypes.

5.5. Attention on Input Text

This experiment is run to demonstrate the importance of the summarization component of the model. The neural editor is trained with and without attention on the input text x and evaluated.

5.6. Zeroed Edit Vector

This experiment is run to determine the significance of the edit vector. Zeroed out edit vectors are passed into the neural editor both during training and evaluation, and performance is evaluated.

5.7. Pretrained Prototype Selector

This experiment is intended to verify if the training procedure correctly trains the prototype selector. We pretrain the prototype selection part of the model as a summarization model. We then continue training the full model, including the neural editor and evaluate the performance.

6. Results

Our main results are presented in Tables 1 and 2, which present train and validation BLEU scores and ELBOs, respectively. In each table, the first two columns are evaluated with the full model (both edit and prototype selection) while the last two columns are evaluated with only the editor (omitting the selection of the most likely prototype). We run experiments on the OpenNMT English Summarization dataset (3 billion sentences for training, and random 128 sentences for evaluation). A qualitative example of the summaries produced by our method can be found in Figure 1.

We notice in particular that in these examples, the output text summarizes relatively well the input. The model seems to be combining information from the input text and the prototype into the generated sentence. In the fourth example in Figure 1., the prototype matches almost entirely the output sentence but gathers the information in the input text in order to specify "Tokyo" instead of "Australian share". However, the other examples also highlight two limitations of our model. First, the prototype selector does not learn well to select the closest prototype as in the first example on Israeli troops where the prototype is lexically very different from the target sentence. Second, as a consequence the model that we constructed seems to be bypassing the prototype in the generation of the final summary, essentially behaving as an attention-based NLM on the input text.

Quantitatively, although the results look favorable, we note that because of relatively small evaluation sets, there is some fluctuation in these values, which may due to the small set of evaluation examples (random 128). Furthermore, there also seems to be a degree of overfitting.

Experiment	Train BLEU	Validation BLEU	Train (Edit) BLEU	Test (Edit) BLEU
Full Model	0.549	0.547	0.440	0.409
Summarization Baseline	0.442	0.443	None	None
Train and Evaluate on Random Prototype	0.367	0.394	0.395	0.399
Evaluate on Random Prototype	0.389	0.250	0.324	0.340
No Attention	0.149	0.134	0.100	0.096
Zeroed out Edit Vector	0.533	0.493	0.412	0.463

Table 1. Experiment Results: BLEU

Experiment	Train ELBO	Validation ELBO	Train (Edit) ELBO	Test (Edit) ELBO
Full Model	23.14	59.396	40.924	60.754
Summarization Baseline	None	None	None	None
Train and Evaluate on Random Prototype	45.21	62.64	62.89	64.22
Evaluate on Random Prototype	57.96	76.39	75.64	76.35
No Attention	45.48	77.45	77.20	78.26
Zeroed out Edit Vector	24.94	59.432	44.098	60.548

Table 2. Experiment Results: ELBO

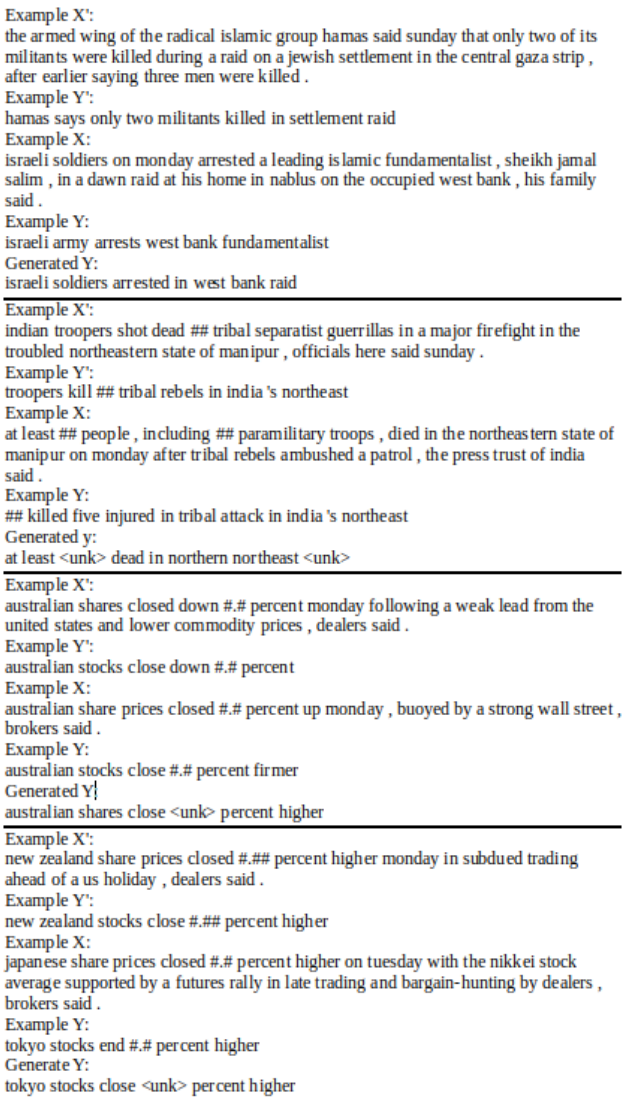

Figure 1. Sample Summaries and Related prototypes

7. Discussion

In this section we discuss the results obtained and explain their implication on the choices in our model.

First, the BLEU score is better on average when we evaluate the model with both the prototype selector and the neural editor compared to just the neural editor alone (first two columns of the two tables vs the last two columns). This indicates that our prototype selector is an important part of our model and improves the summarization results. It reflects the fact that the edit model alone is limited, and the choice of good prototypes is important to achieve good results.

Looking at ”zeroed out Edit Vector” experiment, although not significant, the edit vectors contribute positively to the performance of the summaries. This justifies the use of edit vector to, rather than just using prototypes and input text.

From randomizing prototypes, we see that our model may be learning a useful prototype selector. On one hand, we observe that evaluating our full model using random prototypes gives us a worse BLEU score on both train and validation. On the other hand, this further justifies our last argument. Previously, we observe that our BLEU score is better when we use prototype selector compared to edit-only model. (comparing first two columns with last two columns). However, one may argue that such contribution is due to the nature of using beam search decoder: the pro-

totype selector can be ignored, but choosing from several generated sentences (corresponding to multiple prototypes) might have an ensemble effect. If that's the only reason, in train/evaluate on random prototype experiments, we would also see a better BLEU score in first two columns than the last two columns, which is not the case.

By comparing our full model experiment result with train/evaluate with random prototype results, we see our prototype selector may be learning some useful information. In particular, for most models with prototype selector, both train and validation BLEU are higher than edit-only train and validation BLEU. However, in train/evaluate with random prototype experiment, it is not the case.

From the comparison between evaluating the model with and without attention on the input text, the model with no attention seems consistently worse compared to the full model. This demonstrates the importance of the input text on the editor, and hence that the influence of traditional summarization on our editor is still quite significant, and also that prototype-editing and traditional summarization seem to be complementary.

One limitation of the current prototype-editing model seems to be the representation of the approximate edit posterior. We see that the presence or absence of the edit vector does not affect a lot the performance of the summaries. It currently is set as the concatenation of the sum of word vectors of inserted and deleted words. While this representation can serve as a basic proof-of-concept for prototype-editing based approaches, it is extremely limiting in terms of encapsulating the information necessary to properly edit a sentence (ex: only capturing lexical information). One extension that we believe would greatly improve the potential of a prototype-editing based model is having a more complex representation for the approximate edit posterior, for instance with a neural network. This is a natural extension of our model, and would not imply any change in our inference approach. This change would only affect the mean of the approximate edit prior $q(z|y', x)$, which does not change the expression of the KL term.

Another important limitation of our model, and also part of the reason why our results are not evaluated on larger sets is that the generation is very costly at the moment. As mentioned earlier, the current model computes the likelihood term from the prototype selector and the neural editor for all the prototypes in the neighborhood, and then only keeps the highest. This is a very costly operation, and is hard to scale. An extension to this work would be to skip the step of generating all summaries in a neighborhood and selecting the best, and add a direct attention-based dependence of a fixed number of prototypes in the neural editor. This would allow the neural editor to generate words while attending on all prototypes and could streamline the generation process.

8. Conclusion and Future Work

We contribute an exploration of a conditional, fully-generative sentence model based on prototype-editing applied to summarization. We provide the probabilistic framework for such a model and propose techniques and a basic representation of its components to be used for abstractive summarization. Further work can be done to explore different representations of the components of the model to achieve better performance, as well as getting a faster and more accurate evaluation. It would also be interesting to explore the application of this generative model to other conditional tasks. Ultimately, we show that the prototype-then-edit technique could be promising for the modeling of natural language generation.

References

Gu, Jiatao, Wang, Yong, Cho, Kyunghyun, and Li, Victor O. K. Search engine guided non-parametric neural machine translation. *CoRR*, abs/1705.07267, 2017. URL http://arxiv.org/abs/1705.07267.

Guu, Kelvin, Hashimoto, Tatsunori B., Oren, Yonatan, and Liang, Percy. Generating sentences by editing prototypes. *CoRR*, abs/1709.08878, 2017. URL http://arxiv.org/abs/1709.08878.

Kingma, Diederik P. and Welling, Max. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. URL http://arxiv.org/abs/1312.6114.

Li, Jiwei, Galley, Michel, Brockett, Chris, Gao, Jianfeng, and Dolan, Bill. A diversity-promoting objective function for neural conversation models. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pp. 110–119, 2016. URL http://aclweb.org/anthology/N/N16/N16-1014.pdf.

Li, Piji, Lam, Wai, Bing, Lidong, and Wang, Zihao. Deep recurrent generative decoder for abstractive text summarization. *CoRR*, abs/1708.00625, 2017. URL http://arxiv.org/abs/1708.00625.

Nallapati, Ramesh, Xiang, Bing, and Zhou, Bowen. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016. URL http://arxiv.org/abs/1602.06023.

Rush, Alexander M., Chopra, Sumit, and Weston, Jason. A neural attention model for abstractive sentence sum-

marization. *CoRR*, abs/1509.00685, 2015. URL http://arxiv.org/abs/1509.00685.

See, Abigail, Liu, Peter J., and Manning, Christopher D. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.

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

Zhou, Qingyu, Yang, Nan, Wei, Furu, and Zhou, Ming. Selective encoding for abstractive sentence summarization. *CoRR*, abs/1704.07073, 2017. URL http://arxiv.org/abs/1704.07073.