Generating Sentences by Editing Prototypes

Kelvin Guu*[2] Tatsunori B. Hashimoto*[1,2] Yonatan Oren[1] Percy Liang[1,2]
(* equal contribution)
[1]Department of Computer Science [2]Department of Statistics
Stanford University
{kguu,thashim,yonatano}@stanford.edu pliang@cs.stanford.edu

Abstract

We propose a new generative language model for sentences that first samples a prototype sentence from the training corpus and then edits it into a new sentence. Compared to traditional language models that generate from scratch either left-to-right or by first sampling a latent sentence vector, our prototype-then-edit model improves perplexity on language modeling and generates higher quality outputs according to human evaluation. Furthermore, the model gives rise to a latent edit vector that captures interpretable semantics such as sentence similarity and sentence-level analogies.

1 Introduction

The ability to generate sentences is core to many NLP tasks, including machine translation, summarization, speech recognition, and dialogue. Most neural models for these tasks are based on recurrent neural language models (NLMs), which generate sentences from scratch, often in a left-to-right manner (Bengio et al., 2003). It is often observed that such NLMs suffer from the problem of favoring generic utterances such as "I don't know" (Li et al., 2016). At the same time, naive strategies to increase diversity have been shown to compromise grammaticality (Shao et al., 2017), suggesting that current NLMs may lack the inductive bias to faithfully represent the full diversity of complex utterances.

Indeed, it is difficult even for humans to write complex text from scratch in a single pass; we often create an initial draft and incrementally revise it (Hayes and Flower, 1986). Inspired by this process, we propose a new unconditional generative model of text which we call the prototype-then-edit model, illustrated in Figure 1. It first samples a random *prototype* sentence from the training corpus, and then invokes a *neural editor*, which draws a random "edit vector" and generates a new sentence by attending to the prototype while conditioning on the edit vector. The motivation is that sentences from the corpus provide a high quality starting point: they are grammatical, naturally diverse, and exhibit no bias towards shortness or vagueness. The attention mechanism (Bahdanau et al., 2015) of the neural editor strongly biases the generation towards the prototype, and therefore it needs to solve a much easier problem than generating from scratch.

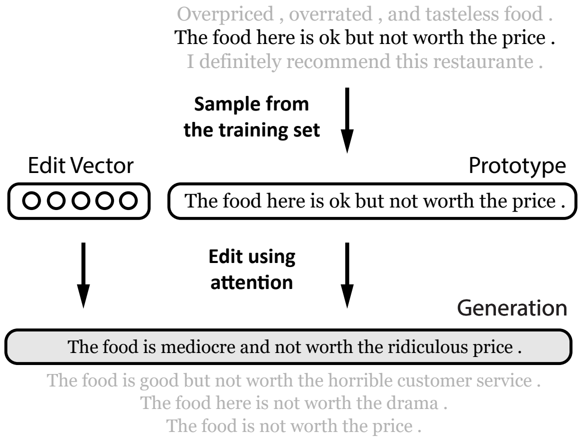
Overpriced , overrated , and tasteless food .
The food here is ok but not worth the price .
I definitely recommend this restaurante .
Sample from the training set
Edit Vector
Prototype
The food here is ok but not worth the price .
Edit using attention
Generation
The food is mediocre and not worth the ridiculous price .
The food is good but not worth the horrible customer service .
The food here is not worth the drama .
The food is not worth the price .

Figure 1: The prototype-then-edit model generates a sentence by sampling a random example from the training set and then editing it using a randomly sampled edit vector.

We train the neural editor by maximizing an approximation to the generative model's log-likelihood. This objective is a sum over lexically-

Transactions of the Association for Computational Linguistics, vol. 6, pp. 437–450, 2018. Action Editor: Trevor Cohn.
Submission batch: 9/2017; Revision batch: 12/2017; Published 7/2018.

similar sentence pairs in the training set, which we can scalably approximate using locality sensitive hashing. We also show empirically that most lexically similar sentences are also semantically similar, thereby endowing the neural editor with additional semantic structure. For example, we can use the neural editor to perform a random walk from a seed sentence to traverse semantic space.

We compare our prototype-then-edit model to approaches that generate from scratch on both language generation quality and semantic properties. For the former, our model generates higher quality generations according to human evaluations, and improves perplexity by 13 points on the Yelp corpus and 7 points on the One Billion Word Benchmark. For the latter, we show that latent edit vectors outperform standard sentence variational autoencoders (Bowman et al., 2016) on semantic similarity, locally-controlled text generation, and a sentence analogy task.

2 Problem statement

Our primary goal is to learn a generative model of sentences for use as a language model.[1] In particular, we model sentence generation as a prototype-then-edit process:

1. **Select prototype**: Given a training corpus of sentences $\mathcal{X}$, randomly sample a *prototype* sentence x' from a *prototype distribution* $p(x')$ (in our case, uniform over $\mathcal{X}$).
2. **Edit**: Sample an edit vector z (encoding the type of edit to be performed) from an *edit prior* $p(z)$. Then, feed the edit vector z and the previously selected prototype x' into a *neural editor* $p_{\text{edit}}(x \mid x', z)$, which generates a new sentence x.

Under this model, the likelihood of a sentence is:

$$p(x) = \sum_{x' \in \mathcal{X}} p(x \mid x') p(x') \tag{1}$$

$$p(x \mid x') = \mathbb{E}_{z \sim p(z)} \left[p_{\text{edit}}(x \mid x', z) \right], \tag{2}$$

where both prototype x' and edit vector z are latent variables.

Our formulation stems from the observation that many sentences in a large corpus can be represented as minor transformations of other sentences. For example, in the Yelp restaurant review corpus (Yelp, 2017) we find that 70% of the test set is within word-token Jaccard distance 0.5 of a training set sentence, even though almost no sentences are repeated verbatim. This implies that a neural editor which models lexically similar sentences should be an effective generative model for large parts of the test set.

A secondary goal for the neural editor is to capture certain semantic properties; we focus on the following two in particular:

1. Semantic smoothness: an edit should be able to alter the semantics of a sentence by a small and well-controlled amount, while multiple edits should make it possible to accumulate a larger change.
2. Consistent edit behavior: the edit vector z should model/control the variation in the type of edit that is performed. When we apply the same edit vector on different sentences, the neural editor should perform *semantically analogous* edits across the sentences.

In Section 4, we show that the neural editor can successfully capture both properties, as reported by human evaluations.

3 Approach

We would like to train our neural editor $p_{\text{edit}}(x \mid x', z)$ by maximizing the marginal likelihood (Equation 1) via gradient ascent, but the objective cannot be computed exactly because it involves a sum over all prototypes x' (expensive) and an expectation over the edit prior $p(z)$ (no closed form).

We therefore propose two approximations to overcome these challenges:

1. We lower bound the sum over latent prototypes x' (in Equation 1) by only summing over x' that are *lexically similar* to x.
2. We lower bound the expectation over the edit prior (in Equation 2) using the evidence lower bound (ELBO) (Jordan et al., 1999; Doersch, 2016) which can be effectively approximated.

[1] For many applications such as machine translation or dialogue generation, there is a context (e.g. foreign sentence, dialogue history), which can be supplied to both the prototype selector and the neural editor. This paper focuses on the unconditional case, proposing an alternative to LSTM based language models.

We describe and motivate these approximations in Sections 3.1 and 3.2, respectively. In Section 3.3, we combine the two approximations to give the final objective. Sections 3.4 and 3.5 drill down further into our specific model architecture.

3.1 Approximate sum on prototypes, x'

Equation 1 defines the probability of generating a sentence x as the total probability of reaching x via edits from every prototype $x' \in \mathcal{X}$. However, most prototypes are unrelated and should have very small probability of transforming into x. Therefore, we approximate the summation over prototypes by only considering prototypes x' that have *high lexical overlap* with x. To that end, define a lexical similarity neighborhood as:

$$\mathcal{N}(x) \overset{\text{def}}{=} \{x' \in \mathcal{X} : d_J(x, x') < 0.5\},$$

where $d_J(x, x')$ is the Jaccard distance between x and x' (treating each as a set of word tokens).

We will now lower bound $\log p(x)$ in two ways: (i) we will sum over only prototypes in the neighborhood $\mathcal{N}(x)$ rather than over the entire training set $\mathcal{X}$ as discussed above; (ii) we will push the log inside the summation using Jensen's inequality, as is standard with variational lower bounds. Recall that the distribution over prototypes is uniform ($p(x') = 1/|\mathcal{X}|$), and define $R(x) = \log(|\mathcal{N}(x)|/|\mathcal{X}|)$. The derivation is as follows:

$$\begin{aligned}
\log p(x) &= \log \left[\sum_{x' \in \mathcal{X}} p(x \mid x') p(x') \right] \\
&\overset{(i)}{\geq} \log \left[\sum_{x' \in \mathcal{N}(x)} p(x \mid x') p(x') \right] \qquad (3) \\
&= \log \left[|\mathcal{N}(x)|^{-1} \sum_{x' \in \mathcal{N}(x)} p(x \mid x') \right] + R(x) \\
&\overset{(ii)}{\geq} \underbrace{|\mathcal{N}(x)|^{-1} \sum_{x' \in \mathcal{N}(x)} \log p(x \mid x')}_{\overset{\text{def}}{=} \text{LEX}(x)} + R(x).
\end{aligned}$$

Assuming the neighborhood size $|\mathcal{N}(x)|$ is constant across x, then LEX(x) is a lower bound of $\log p(x)$ up to constants. For each x, the neighborhood $\mathcal{N}(x)$ can be efficiently precomputed with locality sensitive hashing (LSH) and minhashing. The full procedure is described in Appendix 6.

Note that LEX(x) is still intractable to compute because each $\log p(x|x')$ term involves an expectation over the edit prior $p(z)$ (Equation 2). We address this in Section 3.2, but first, an interlude.

Interlude: lexical similarity semantics. So far, we have motivated lexical similarity neighborhoods via computational considerations, but we found that lexical similarity training also captures semantic similarity. One can certainly construct sentences with small lexical distance that differ semantically (e.g., insertion of the word "not"). However, since we mine sentences from a corpus grounded in real world events, most lexically similar sentences are also semantically similar. For example, given "my son enjoyed the delicious pizza", we are far more likely to see "my son enjoyed the delicious macaroni", versus "my son hated the delicious pizza".

Human evaluations of 250 edit pairs sampled from lexical similarity neighborhoods on the Yelp corpus support this conclusion. 35.2% of the sentence pairs were judged to be exact paraphrases, while 84% of the pairs were judged to be at least roughly equivalent. Sentence pairs were negated or change in topic only 7.2% of the time. Thus, a neural editor trained on this distribution should preferentially generate semantically similar edits.

Note that semantic similarity is not needed if we are only interested in modeling the distribution $p(x)$. But it does enable us to learn an edit model $p(x|x')$ that prefers semantically meaningful edits, which we explore in Section 4.3.

3.2 Approximate expectation on edit vectors, z

In Section 3.1, we approximated the marginal likelihood $\log p(x)$ by LEX(x), which is a summation over terms of the form:

$$\log p(x \mid x') = \log \mathbb{E}_{z \sim p(z)} \left[p_{\text{edit}}(x \mid x', z) \right]. \quad (4)$$

Unfortunately the expectation over $p(z)$ has no closed form, and naively approximating it by Monte Carlo sampling $z \sim p(z)$ will have unacceptably high variance, because $p_{\text{edit}}(x \mid x', z)$ will be almost zero for nearly all z sampled from $p(z)$, while being large for a few important but rare values.

To address this, we introduce an *inverse neural editor* $q(z \mid x', x)$: given a prototype x' and a revised sentence x, it generates edit vectors that are *likely* to map x' to x, concentrating probability on the few rare but important values of z.

We can then use the evidence lower bound (ELBO) to lower bound Equation 4:

$$\log p(x|x') \geq \underbrace{\mathbb{E}_{z\sim q(z|x',x)}\left[\log p_{\text{edit}}(x \mid x', z)\right]}_{\mathcal{L}_{\text{gen}}} - \underbrace{\text{KL}(q(z \mid x', x) \parallel p(z))}_{\mathcal{L}_{\text{KL}}} \overset{\text{def}}{=} \text{ELBO}(x, x').$$

Since $\mathcal{L}_{\text{gen}}$ is an expectation over $q(z \mid x', x)$ instead of $p(x)$, it can be effectively Monte Carlo estimated by sampling $z \sim q(z \mid x', x)$. The second term, $\mathcal{L}_{\text{KL}}$, penalizes the difference between $q(z \mid x', x)$ and $p(x)$, which is necessary for the lower bound to hold. A thorough introduction to the ELBO is provided in Doersch (2016).

Note that $q(z \mid x', x)$ and $p_{\text{edit}}(x \mid x', z)$ combine to form a *variational autoencoder* (VAE) (Kingma and Welling, 2014), where $q(z \mid x', x)$ is the *variational encoder* and $p_{\text{edit}}(x \mid x', z)$ is the *variational decoder*.

3.3 Final objective

Combining the lower bounds LEX(x) and ELBO(x, x'), our final approximation of the log-likelihood is

$$\sum_{x'\in\mathcal{N}(x)} \text{ELBO}(x, x').$$

We optimize this objective using stochastic gradient ascent with respect to $\Theta = (\Theta_p, \Theta_q)$, where Θ_p are the parameters for the neural editor and Θ_q are the parameters for the inverse neural editor.

3.4 Model architecture

To recap, our model features three components: the **neural editor** $p_{\text{edit}}(x \mid x', z)$, the **edit prior** $p(z)$, and the **inverse neural editor** $q(z \mid x', x)$. We detail each of these components below.

Neural editor $p_{\text{edit}}(x \mid x', z)$. We implement our neural editor as a left-to-right sequence-to-sequence model with attention, where the prototype x' is the input sequence and the revised sentence x is the output sequence. We employ an encoder-decoder architecture similar to Wu (2016), extending it to condition on an edit vector z by concatenating z to the input of the decoder at each time step.

The prototype encoder is a 3-layer bidirectional LSTM. The inputs to each layer are the concatenation of the forward and backward hidden states of the previous layer, with the exception of the first layer, which takes word vectors initialized using GloVe (Pennington et al., 2014).

The decoder is a 3-layer LSTM with attention. At each time step, the hidden state of the top layer is used to compute attention over the top-layer hidden states of the prototype encoder. The resulting attention context vector is then concatenated with the decoder's top-layer hidden state and used to compute a softmax distribution over output tokens.

Edit prior $p(z)$. We sample the edit vector z from the prior by first sampling its scalar length $z_{\text{norm}} \sim \text{Unif}(0, 10)$ and then sampling its direction z_{dir} (a unit vector) from the uniform distribution on the unit sphere. The resulting $z = z_{\text{norm}} \cdot z_{\text{dir}}$. As we will see later, this particular choice of the prior enables us to easily compute $\mathcal{L}_{\text{KL}}$.

Inverse neural editor $q(z \mid x', x)$. Given an edit pair (x', x), the inverse neural editor must infer what vectors z are likely to map x' to x.

Suppose that x' and x only differed by a single word w. Then one might propose that the edit vector z should be equal to the word vector for w. Generalizing this intuition to multi-word edits, we would like multi-word insertions to be represented as the sum of the inserted word vectors, and similarly for deletions.

Formally, define $I = x\backslash x'$ to be the set of words added to x', and $D = x'\backslash x$ to be the words deleted. We represent the difference between x' and x using the following vector:

$$f\left(x, x'\right) = \sum_{w\in I} \Phi\left(w\right) \oplus \sum_{w\in D} \Phi\left(w\right)$$

where $\Phi(w)$ is the word vector for word w and $\oplus$ denotes concatenation. The word embeddings Φ are parameters of q. In our work, we initialize $\Phi(w)$ to be 300-dimensional GloVe vectors.

Since we construct our edit vectors as the sum of word vectors, and similarities between word vectors have traditionally been measured with cosine similarity, we design q to add noise to perturb the direction of the vector f. In particular, a sample from q is simply a perturbed version of f: obtained by adding von-Mises Fisher (vMF) noise, and we perturb the magnitude of f by adding uniform noise. We visualize this perturbation process in Figure 2.

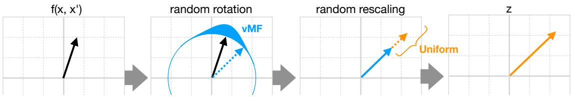

Figure 2: The inverse neural editor q outputs a perturbed version of $f(x, x')$. The perturbation process is a random rotation (according to the von-Mises Fisher distribution) followed by a random rescaling (according to the uniform distribution).

Formally, let $f_{\text{norm}} = \|f\|$ and $f_{\text{dir}} = f/f_{\text{norm}}$. Let vMF $(v; \mu, \kappa)$ denote a vMF distribution over points v on the unit sphere (i.e., directions) with mean vector μ and concentration parameter κ (in such a distribution, the log-likelihood of a point decays linearly with its cosine similarity to μ, and the rate of decay is controlled by κ). Finally, define:

$$q(z_{\text{dir}} \mid x', x) = \text{vMF}\left(z_{\text{dir}}; f_{\text{dir}}, \kappa\right)$$
$$q(z_{\text{norm}} \mid x', x) = \text{Unif}(z_{\text{norm}}; [\tilde{f}_{\text{norm}}, \tilde{f}_{\text{norm}} + \epsilon])$$

where $\tilde{f}_{\text{norm}} = \min(f_{\text{norm}}, 10 - \epsilon)$ is the truncated norm. The resulting edit vector is $z = z_{\text{dir}} \cdot z_{\text{norm}}$.

The inverse neural editor q is parameterized by the word vectors Φ and has hyperparameters κ and ϵ. Further details are provided in Section 3.5.

3.5 Details of the inverse neural editor

Differentiating w.r.t. Θ_q. To maximize our training objective, we must be able to compute $\nabla_{\Theta_q}\text{ELBO}(x, x') = \nabla_{\Theta_q}\mathcal{L}_{\text{gen}} - \nabla_{\Theta_q}\mathcal{L}_{\text{KL}}$.

To compute $\nabla_{\Theta_q}\mathcal{L}_{\text{gen}}$, we use a reparameterization trick. Specifically, we can rewrite $z \sim q(z \mid x', x)$ as $z = h(\alpha)$ where h is a deterministic function differentiable with respect to Θ_q and $\alpha \sim p(\alpha)$ is an auxiliary random variable not depending on Θ_q (the details of h and α are given in Appendix 6). We can then write:

$$\begin{aligned}\nabla_{\Theta_q}\mathcal{L}_{\text{gen}} &= \nabla_{\Theta_q}\mathbb{E}_{z\sim q(z|x',x)}\left[\log p_{\text{edit}}(x \mid x', z)\right] \\ &= \mathbb{E}_{\alpha\sim p(\alpha)}\left[\nabla_{\Theta_q}\log p_{\text{edit}}(x \mid x', h(\alpha))\right].\end{aligned}$$

This moves the derivative inside the expectation. The inner derivative can now be computed via standard backpropagation.

Next, we turn to $\nabla_{\Theta_q}\mathcal{L}_{\text{KL}}$. First, note that:

$$\begin{aligned}\mathcal{L}_{\text{KL}} &= \text{KL}(q(z_{\text{norm}}|x', x)\|p(z_{\text{norm}})) \\ &\quad + \text{KL}(q(z_{\text{dir}}|x', x)\|p(z_{\text{dir}})). \qquad (5)\end{aligned}$$

It is easy to verify that the first KL term does not depend on Θ_q. The second term has the closed form

$$\begin{aligned}\text{KL}(\text{vMF}(\mu,\kappa)\|\text{vMF}(\mu,0)) &= \kappa\frac{I_{d/2}(\kappa) + I_{d/2-1}(\kappa)\frac{d-2}{2\kappa}}{I_{d/2-1}(\kappa) - \frac{d-2}{2\kappa}} \\ &\quad - \log(I_{d/2-1}(\kappa)) - \log(\Gamma(d/2)) \\ &\quad + \log(\kappa)(d/2-1) - (d-2)\log(2)/2, \qquad (6)\end{aligned}$$

where $I_n(\kappa)$ is the modified Bessel function of the first kind, Γ is the gamma function, and d is the dimensionality of f. We can see that this too is constant with respect to Θ_q via the following intuition: both the KL divergence and the prior do not change under rotations, and thus we can see $\text{KL}(\text{vMF}(\mu,\kappa)\|\text{vMF}(\mu,0))) = \text{KL}(\text{vMF}(e_1,\kappa)\|\text{vMF}(e_1,0)))$ by rotating μ to the first canonical basis vector. Hence $\nabla_{\Theta_q}\mathcal{L}_{\text{KL}} = 0$.

Comparison with existing VAE encoders. Our design of q differs from the typical choice of a standard normal distribution (Bowman et al., 2016; Kingma and Welling, 2014) for two reasons:

First, by construction, edit vectors are sums of word vectors and since cosine distances are traditionally used to measure distances between word vectors, it would be natural to encode distances between edit vectors by the cosine distance. The von-Mises Fisher distribution captures this idea, as the log likelihood decays with cosine similarity.

Second, our design of q allows us to explicitly control the tradeoff between the two terms in our objective, $\mathcal{L}_{\text{gen}}$ and $\mathcal{L}_{\text{KL}}$. Note from equations 5 and 6 that $\mathcal{L}_{\text{KL}}$ is purely a function of the hyperparameters ϵ and κ, and can thus be controlled exactly. By taking $\kappa \to 0$ and ϵ to the maximum norm, we can drive

$\mathcal{L}_{\text{KL}}$ arbitrarily close to 0. As a tradeoff, smaller values of κ produce a noisier edit vector, leading to a smaller $\mathcal{L}_{\text{gen}}$. We find a good balance by tuning κ.

In contrast, when using a Gaussian variational encoder, the KL term takes a different value per example and cannot be explicitly controlled. Consequently, Bowman et al. (2016) and others have observed that training tends to aggressively drive these KL terms to zero, leading to uninformative values of z — even when multiplying $\mathcal{L}_{\text{KL}}$ by a carefully tuned and annealed importance weight.

4 Experiments

We divide our experimental results into two parts. In Section 4.2, we evaluate the merits of the prototype-then-edit model as a generative modeling strategy, measuring its improvements on language modeling (perplexity) and generation quality (human evaluations of diversity and plausibility). In Section 4.3, we focus on the semantics learned by the model and its latent edit vector space. We demonstrate that it possesses interpretable semantics, enabling us to smoothly control the magnitude of edits, incrementally optimize sentences for target properties, and perform analogy-style sentence transformations.

4.1 Datasets

We evaluate perplexity on the Yelp review corpus (Yelp, Yelp (2017)) and the One Billion Word Language Model Benchmark (BillionWord, Chelba (2013)). For qualitative evaluations of generation quality and semantics, we focus on Yelp as our primary test case, as we found that human judgments of semantic similarity were much better calibrated in this focused setting.

For both corpora, we used the named-entity recognizer (NER) in spaCy[2] to replace named entities with their NER categories. We replaced tokens outside the top 10,000 most frequent tokens with an "out-of-vocabulary" token.

4.2 Generative modeling

We compare NeuralEditor as a language model against the following baseline language models:

1. NLM: a standard left-to-right neural language model generating from scratch. For fair comparison, we use the exact same architecture as the decoder of NeuralEditor.
2. KN5: a standard 5-gram Kneser-Ney language model in KenLM (Heafield et al., 2013).
3. Memorization: generates by sampling a sentence from the training set.

Perplexity. We start by evaluating NeuralEditor's value as a language model, measured in terms of perplexity. We use the likelihood lower bound in Equation 3, where we sum over training set instances within Jaccard distance < 0.5, and for the VAE term in NeuralEditor, we use the one-sample approximation to the lower bound used in Kingma (2014) and Bowman (2016).

To evaluate NeuralEditor's perplexity, we use linear smoothing with NLM to account for rare sentences not within our Jaccard distance threshold. This smoothing corresponds to occasionally sampling a special prototype sentence that can be edited into any other sentence and we use a smoothing weight of 0.1 (for full details, see Appendix 6). We find NeuralEditor improves perplexity over NLM and KN5. Table 1 shows that this is the case for both Yelp and the more general BillionWord, which contains substantially fewer test-set sentences close to the training set. On Yelp, we surpass even the best ensemble of NLM and KN5, while on BillionWord we nearly match their performance.

Comparing each model at a per-sentence level, we see that NeuralEditor drastically improves log-likelihood for a significant number of sentences in the test set (Figure 3). Proximity to a prototype seems to be the chief determiner of NeuralEditor's performance.

Model	Perplexity (Yelp)	Perplexity (BillionWord)
KN5	56.546	78.361
KN5+Memorization	55.184	73.468
NLM	39.026	55.146
NLM+Memorization	38.086	50.969
NLM+KN5	37.312	**47.472**
NeuralEditor($\kappa = 0$)	**26.87**	48.755
NeuralEditor($\kappa = 25$)	27.41	48.921

Table 1: Perplexity of NeuralEditor with the two VAE parameters κ outperform all methods on Yelp and all non-ensemble methods on BillionWord.

[2] honnibal.github.io/spaCy

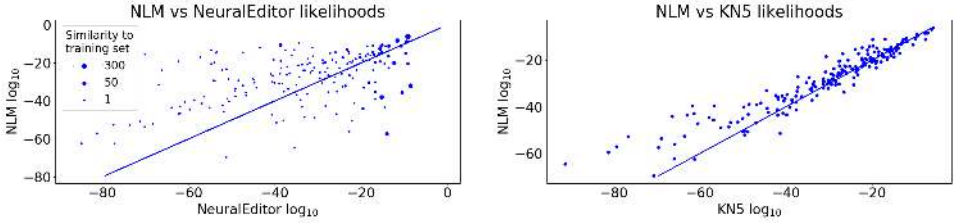

Figure 3: NEURALEDITOR outperforms NLM on examples similar to those in the training set (left panel, point size indicates number of training set examples with Jaccard distance < 0.5). The N-gram baseline (right) shows no such behavior, with NLM outperforming KN5 on most examples.

Prototype x'	**Revision** x
this place gets <cardinal> stars for its diversity in its menu .	this place gets <cardinal> stars although not for the prices .
great food and the happy hour deals were out of this world .	the deals are great and the food is out of this world .
i've been going to <person> for <date> and i used to really like this place .	i've been going to this place for <date> now and love it .
their food is great , and you can't beat the price .	you can't beat the service and food here .

Table 2: Edited generations are substantially different from the sampled prototypes.

Since NEURALEDITOR draws its strength from sentences in the training set, we also compared against a simpler alternative, in which we ensemble NLM and MEMORIZATION (retrieval without edits). NEURALEDITOR performs dramatically better than this alternative. Table 2 also qualitatively demonstrates that sentences generated by NEURALEDITOR are substantially different from the original prototypes.

Human evaluation. We now turn to human evaluation of generation quality, focusing on grammaticality and plausibility. We evaluated plausibility by asking human raters, "How plausible is it for this sentence to appear in the corpus?" on a scale of 1– 3. We evaluate generations from NEURALEDITOR against an NLM with a temperature parameter on the per-token softmax[3] as well as a baseline which generates sentences by randomly sampling from the training set and replacing synonyms, where the probability of substitution follows $\exp(s_{ij}/\tau)$, where s_{ij} is the cosine similarity between the original word and its synonym according to GloVe word vectors.

Decreasing the temperature parameter below 1 is a popular technique for suppressing incoherent and ungrammatical sentences. Many NLM systems have noted an undesirable tradeoff between grammaticality and diversity, where a temperature low enough to enforce grammaticality results in short and generic utterances (Li et al., 2016).

Figure 4 illustrates that both the grammaticality and plausibility of NEURALEDITOR without any temperature annealing is on par with the best tuned temperature for NLM, with a far higher diversity, as measured by the discrete entropy over unigram frequencies. We also find that decreasing the temperature of NEURALEDITOR can be used to slightly improve the grammaticality, without substantially reducing the diversity of the generations.

Comparing with the synonym substitution model, we find both models have high plausibility, since synonym substitution maintains most of the words, but low grammaticality compared to both NEURALEDITOR and the NLM. Additionally, applying synonym substitutions to training examples has extremely low coverage – none of the sentences in the test set can be generated via synonym substitution, and thus this baseline has higher perplexity than all other baselines in Table 1.

A key advantage of edit-based models thus emerges: Prototypes sampled from the training set organically inject diversity into the generation process, even if the temperature of the decoder in NEURALEDITOR is zero. Hence, we can keep the decoder at a very low temperature to maximize grammaticality and plausibility, without sacrificing diversity. In contrast, a zero temperature NLM would collapse to outputting one generic sentence.

This also suggests that the temperature parameter for NEURALEDITOR captures a more natural notion of diversity — a temperature of 1.0 encourages more aggressive extrapolation from the training set

[3] If s_i is the softmax logit for token w_i and τ is a temperature parameter, the temperature-adjusted distribution is $p(w_i) \propto \exp(s_i/\tau)$.

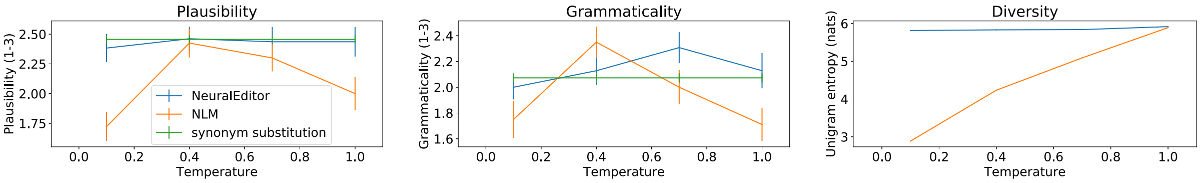

Figure 4: NEURALEDITOR provides plausibility and grammaticality on par with the best, temperature-tuned language model without any loss of diversity as a function of temperature. Results are based on 400 human evaluations.

while lower temperatures favor more conservative mimicking. This is likely to be more useful than the tradeoff for generation-from-scratch, where low temperature also affects the diversity of generations.

Categorizing edits. To better understand the behavior of NEURALEDITOR, we measured the frequency with which random edits from NEURALEDITOR matched known syntactic transformations.

We use the rule-based transformations defined in He (2015) as our set of transformations to test, and search the corpus for sentences where these rules can be applied. We then apply the rule-based transformation, and measure the log-likelihood that NEURALEDITOR generates the transformed outputs. We find that the edit model assigns relatively high probability to the identity map (no edits), followed by simple reordering transformations such as reordering *to/that* Clauses (*It is* `ADJP` *to/that* `SBAR/S` → *To* `S/BARS` *is* `ADJP`). Of the rules, active / passive receives the lowest probability, partially due to the rarity of passive voice sentences in the Yelp corpus (Table 3).

In all cases, the model assigns substantially higher probability to these rule-based transformations over editing to random sentences or shuffling the tokens randomly to match the Levenstein distance of each rule-based transform.

4.3 Semantics of NEURALEDITOR

In this section, we investigate the learned semantics of NEURALEDITOR, focusing on the two desiderata discussed in Section 2: semantic smoothness, and consistent edit behavior.

In order to establish a baseline for these properties, we consider existing sentence generation techniques which can sample semantically similar sentences. The most similar language modeling approach which can capture semantics is the sentence variational autoencoder (SVAE) which imposes semantic structure onto a latent vector space, but uses the latent vector to represent the entire sentence, rather than just an edit.

To use the SVAE to "edit" a target sentence into a semantically similar sentence, we perturb its underlying latent sentence vector and then decode the result back into a sentence — the same method used in Bowman et al. (2016).

Semantic smoothness. A good editing system should have fine-grained control over the semantics of a sentence: i.e., each edit should only alter the semantics of a sentence by a small and well-controlled amount. We call this property semantic smoothness.

To study smoothness, we first generate an "edit sequence" by randomly selecting a prototype sentence, and then repeatedly editing via NEURALEDITOR (with edits drawn from the edit prior $p(z)$) to produce a sequence of revisions. We then ask human annotators to rate the size of the semantic changes between revisions. An example is given in Table 4.

We compare to two baselines, one based upon the sentence variational autoencoder (SVAE) and another baseline which simply samples similar sentences from the training set according to average word vector similarity (COSINE).

For SVAE, we generate a similar sequence of sentences by first encoding the prototype sentence, and then decoding after the addition of a random Gaussian with variance 0.4.[4] This process is repeated to produce a sequence of sentences which we can view as the SVAE equivalent of the edit sequence.

For COSINE, we generate sentences from the training set using exponentiated cosine similarity be-

[4] The variance was selected so that SVAE and NEURALEDITOR have the same average human similarity judgement between two successive sentences. This avoids situations where SVAE produces completely unrelated sentence due to the perturbation size.

Type of edit	$-\log(p)$ per token	Example	Transformed example
Identity	0.33 ± 0.006	It is important to remain watchful.	It is important to remain watchful.
to Clause reordering	1.62 ± 0.156	It is important to remain watchful.	To remain watchful is important.
Quotative verb reordering	2.02 ± 0.0359	They announced that the president will restructure the division.	The president will restructure the division, they announced.
Conjunction reversal	2.52 ± 0.0520	We should march because winter is coming.	Winter is coming, because of this, we should march.
Genitive reordering	2.678 ± 0.0477	The best restaurant of New York.	New York's best restaurant.
Active / passive	3.271 ± 0.0298	The talk was denied by the boycott group spokesman.	The boycott group spokesman denied the talk.
Random sentence reordering	4.42 ± 0.026	It is important to remain watchful.	It remain is to important watchful.
Editing to random sentences	6.068 ± 0.084		

Table 3: NeuralEditor assigns high probabilities to the syntactic transformations defined in He (2015) compared baselines of editing to random sentences or randomly reordering tokens to match the Levenstein distance of a syntactic edit. Small transformations, such as clause reordering, receive higher probability than more structural edits such as changing from active to passive voice.

NeuralEditor	SVAE
this food was amazing one of the best i've tried, service was fast and great.	this food was amazing one of the best i've tried, service was fast and great.
this is the best food and the best service i've tried in <gpe>.	this place is a great place to go if you want a quick bite.
some of the best <norp> food i've had in <date> i've lived in <gpe>.	the food was good, but the service was terrible.
i have to say this is the best <norp> food i've had in <gpe>.	this is the best <norp> food in <gpe>.
best <norp> food i've had since moving to <gpe> <date>.	this place is a great place to go if you want to eat.
this was some of the best <norp> food i've had in the <gpe>.	this is the best <norp> food in <gpe>.

Table 4: Example random walks from NeuralEditor, where the top sentence is the prototype.

tween averaged word vectors. The temperature parameter for the exponential was selected as before to match the average human similarity judgement.

Figure 5 shows that NeuralEditor frequently generates paraphrases despite being trained on lexical similarity, and only 1% of edits are unrelated from the prototype. In contrast, SVAE often repeats sentences exactly, and when it makes an edit it is equally likely to generate unrelated sentences. Cosine performs even worse likely due to the difficulty of retrieving similar sentences for rare and long sentences.

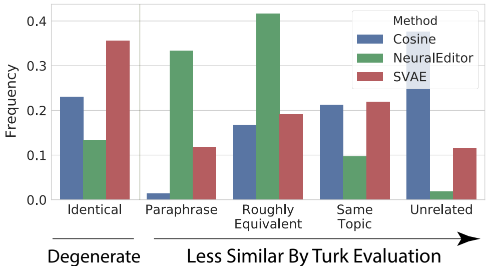

Figure 5: Compared with baselines, NeuralEditor frequently generates paraphrases and similar sentences while avoiding unrelated and degenerate ones.[6]

Qualitatively (Table 4), NeuralEditor seems to generate long, diverse sentences which smoothly change over time, while the SVAE biases towards short sentences with several semantic jumps, presumably due to the difficulty of training a sufficiently informative SVAE encoder.

Smoothly controlling sentences. We now show that we can selectively choose edits sampled from NeuralEditor to incrementally optimize a sentence towards desired attributes. This task serves as a useful measure of semantic coverage: if an edit model has high coverage over sentences that are semantically similar to a prototype, it should be able to satisfy the target attribute while deviating minimally from the prototype's original meaning.

We focus on controlling two simple attributes: compressing a sentence to below a desired length (e.g., 7 words), and inserting a target keyword into the sentence (e.g., "service" or "pizza").

Given a prototype sentence, we try to discover a semantically similar sentence satisfying the target

[6] 545 similarity assessments pairs were collected through Amazon Mechanical Turk following Agirre (2014), with the same scale and prompt. Similarity judgements were converted to descriptions by defining Paraphrase (5), Roughly Equivalent (4-3), Same Topic (2-1), Unrelated (0).

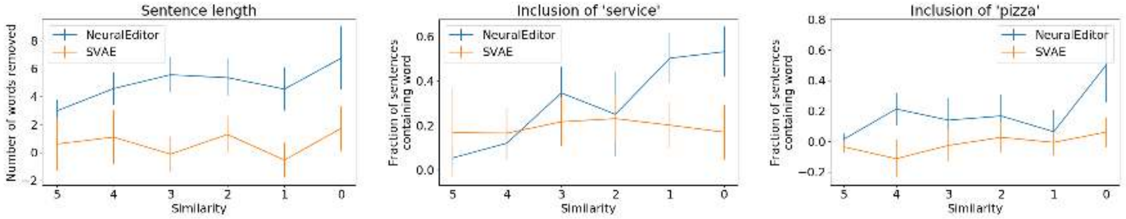

Figure 6: NEURALEDITOR can shorten sentences (left), include common words (center, the word 'service') and rarer words (right 'pizza') while maintaining similarity.

NEURALEDITOR	SVAE
the coffee ice cream was one of the best i've ever tried.	the coffee ice cream was one of the best i've ever tried.
some of the best ice cream we've ever had!	the <unk> was very good and the food was good.
just had the best ice - cream i've ever had!	the food was good, but not great.
some of the best **pizza** i've ever tasted!	the food was good, but not great.
that was some of the best **pizza** i've had in the area.	the food was good, but the service was n't bad.

Table 5: Examples of word inclusion trajectories for 'pizza'. NEURALEDITOR produces smooth chains that lead to word inclusion, but the SVAE gets stuck on generic sentences.

attribute using the following procedure: First, we generate 1,000 edit sequences using the procedure described earlier. Then, we select the sequence with highest likelihood whose endpoint possesses the target attribute. We repeat this process for a large number of prototypes.

We use almost the same procedure for the SVAE, but instead of selecting by highest likelihood, we select the sequence whose endpoint has shortest latent vector distance from the prototype (as this is the SVAE's metric of semantic similarity).

In Figure 6, we then aggregate the sentences from the collected edit sequences, and plot their *semantic similarity to the prototype* against their *success in satisfying the target attribute*. Not surprisingly, as target attribute satisfaction rises, semantic similarity drops. However, we also see that NEURALEDITOR sacrifices less semantic similarity to achieve the same level of attribute satisfaction as SVAE. SVAE is reasonable on tasks involving common words (such as the word *service*), but fails when the model is asked to generate rarer words such as *pizza*. Examples from these word inclusion problems show that SVAE often becomes stuck generating short, generic sentences (Table 5).

Consistent edit behavior: sentence analogies. In the previous results, we showed that edit models learn to generate semantically similar sentences. We now assess whether the edit vector possesses globally consistent semantics. Specifically, applying the same edit vector to different sentences should result in semantically analogous edits.

For example, if we have an edit vector which edits the sentence x_1 = "this was a good restaurant" into x_2 = "this was the best restaurant". Given a new sentence y_1 = "The cake was great", we expect applying the same edit vector to result in y_2 = "The cake was the greatest".

Formally, suppose we have two sentences, x_1 and x_2, which are related by some underlying semantic relation r. Given a new sentence y_1, we would like to find a y_2 such that the same relation r holds between y_1 and y_2.

Our approach is to estimate the edit vector between x_1 and x_2 as $\hat{z} = f(x_1, x_2)$ — the mode of the inverse neural editor q. We then apply this edit vector to y_1 using the neural editor to yield $\hat{y_2} = \text{argmax}_x p_{\text{edit}}(x \mid y_1, \hat{z})$.

Since it is difficult to output $\hat{y_2}$ exactly matching y_2, we take the top k candidate outputs of p_{edit} (using beam search) and evaluate whether the gold y_2 appears among the top k elements.

We generate the semantic relations r using prior evaluations for word analogies (Mikolov et al., 2013a; Mikolov et al., 2013b). We leverage these to generate a new dataset of sentence analogies, using a simple strategy: given an analogous word pair (w_1, w_2), we mine the Yelp corpus for sentence pairs

Google			Microsoft						Method
gram4-superlative	gram3-comparative	family	JJR_JJS	VB_VBD	VBD_VBZ	NN_NNS	VB_VBZ	JJ_JJR	
0.45	0.85	1.0	0.75	0.63	0.82	0.82	0.61	0.77	GloVE
0.75	0.75	0.29	0.79	0.57	0.60	0.58	0.41	0.24	Edit vector (top 10)
0.60	0.32	0.01	0.45	0.16	0.23	0.17	0.01	0.06	Edit vector (top 1)
0.10	0.10	0.09	0.10	0.08	0.14	0.15	0.05	0.03	Sampling (top 10)

Table 6: Edit vectors capture one-word sentence analogies with performance close to lexical analogies.

	Example 1	Example 2
Context	he comes home tired and happy .	i went with a larger group to <person> 's .
Edit	+ was - is	+ good - better
Result	= he came home happy and tired .	= i went to <person> 's with a large group .

Table 7: Examples of lexical analogies correctly answered by NEURALEDITOR. Sentence pairs generating the analogy relationship are shortened to only their lexical differences.

(x_1, x_2) such that x_1 is transformed into x_2 by inserting w_1 and removing w_2 (allowing for reordering and inclusion/exclusion of stop words).

For this task, we initially compared against the SVAE, but it had a top-k accuracy close to zero. Hence, we instead compare to SAMPLING which is a baseline which randomly samples an edit vector $\hat{z} \sim p(z)$, instead using $\hat{z}$ derived from $f(x_1, x_2)$.

We also compare our accuracies to the simpler task of solving the word, rather than sentence-level analogies in (Mikolov et al., 2013a) using GloVe. This task is substantially simpler, since the goal is to identify a single word (such as "good:better::bad:?") instead of an entire sentence. Despite this, the top-10 performance of our model in Table 6 is nearly as good as the performance of GloVe vectors on the simpler lexical analogy task. In some categories, NEURALEDITOR at top-10 actually performs better than word vectors, since NEURALEDITOR has an understanding of which words are likely to appear in the context of a Yelp review. Examples in Table 7 show the model is accurate and captures lexical analogies requiring word reorderings.

5 Related work and discussion

Our work connects with a broad literature on attention-based neural models, retrieval-augmented text generation, semantically meaningful representations, and nonparametric statistics.

Based upon recurrent neural networks and sequence-to-sequence architectures (Sutskever et al., 2014), neural language models (Bengio et al., 2003) have been widely used due to their flexibility and performance across a wide range of NLP tasks (Kalchbrenner and Blunsom, 2013; Hahn and Mani, 2000; Ritter et al., 2011). Our work is motivated by an emerging consensus that attention-based mechanisms (Bahdanau et al., 2015) can substantially improve performance on various sequence to sequence tasks by capturing more information from the input sequence (Vaswani et al., 2017). Our work extends the applicability of attention mechanisms beyond sequence-to-sequence models by allowing models to attend to randomly sampled sentences.

There is a growing literature on applying retrieval mechanisms to augment text generation models. For example, in the image captioning literature, Hodosh (2013), Kuznetsova (2013) and Mason (2014) proposed to generate image captions by first retrieving a prototype caption based on an image context, and then applying sentence compression to tailor the prototype to a particular image. More recently, Song (2016) ensembled a retrieval system and an NLM for dialogue, using the NLM to transform the retrieved utterance, and Gu (2017) used an off-the-shelf search engine system to retrieve and condition on training set examples. Although these approaches also edit text from the training set, these papers solve a fundamentally different problem since they solve conditional generation problems, and retrieve prototypes based on a context, where as our task is unconditional and thus there is no context which we can use to retrieve.

Our work treats the prototype x' as a latent variable rather than being given by a retrieval mechanism, and marginalizes over all possible prototypes — a challenge which motivates our new lexical similarity training method in Section 3.1. Practically, marginalization over x' makes our model attend to training examples based on similarity of *output sequences*, while prior retrieval models attend to examples based on similarity of the *input sequences*.

In terms of generation techniques that capture semantics, the sentence variational autoencoder (SVAE) (Bowman et al., 2016) is closest to our work in that it attempts to impose semantic structure on a latent vector space. However, the SVAE's latent vector is meant to represent the entire sentence, whereas the neural editor's latent vector represents an edit. Our results from Section 4.3 suggest that local variation over edits is easier to model than global variation over sentences.

Our use of lexical similarity neighborhoods is comparable to context windows in word vector training (Mikolov et al., 2013a). More generally, results in manifold learning demonstrate that a weak metric such as lexical similarity can be used to extract semantic similarity through distributional statistics (Tenenbaum et al., 2000; Hashimoto et al., 2016).

From a generative modeling perspective, editing randomly sampled training sentences closely resembles nonparametric kernel density estimation (Parzen, 1962) where one samples points from a training set, and adds noise to smooth the density. Our edit model is the text equivalent of Gaussian noise, and our training mechanism is a type of learned smoothing kernel.

Prototype-then-edit is a semi-parametric approach that remembers the entire training set and uses a neural editor to generalize meaningfully beyond the training set. The training set provides a strong inductive bias — that the corpus can be characterized by prototypes surrounded by semantically similar sentences reachable by edits. Beyond improvements on generation quality as measured by perplexity, the approach also reveals new semantic structures via the edit vector.

Reproducibility. All code, data and experiments are available on the CodaLab platform at `https://bit.ly/2rHsWAX`.

Acknowledgements. We thank the reviewers and editor for their insightful comments. This work was funded by DARPA CwC program under ARO prime contract no. W911NF-15-1-0462.

6 Appendix

Construction of the LSH. The LSH maps a sentence to lexically similar sentences in the corpus, representing a graph over sentences. We apply breadth-first search (BFS) over the LSH sentence graph started at randomly selected seed sentences and uniformly sample this set to form the training set.

Reparameterization trick for q. First, note that we can write $z_{\text{norm}} \sim q(z_{\text{norm}}|x', x)$ as $z_{\text{norm}} = h_{\text{norm}}(\alpha_{\text{norm}}) \overset{\text{def}}{=} \tilde{f}_{\text{norm}} + \alpha_{\text{norm}}$ where $\alpha_{\text{norm}} \sim \text{Unif}(0, \epsilon)$. Furthermore, Wood (1994) present a function h_{dir} and auxiliary random variable α_{dir}, such that $z_{\text{dir}} = h_{\text{dir}}(\alpha_{\text{dir}})$ is distributed according to a vMF with mean f and concentration κ. We can then define $z = h(\alpha) \overset{\text{def}}{=} h_{\text{dir}}(\alpha_{\text{dir}}) \cdot h_{\text{norm}}(\alpha_{\text{norm}})$.

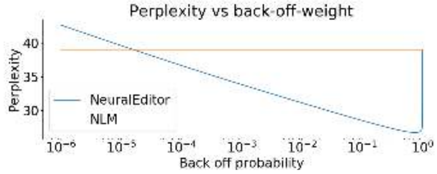

Figure 7: Small amounts of smoothing are sufficient to make NEURALEDITOR outperform the baseline NLM.

Smoothing for language models. As a language model, NEURALEDITOR does not place probability on any test sentence which is sufficiently dissimilar from all training set sentences. In order to avoid this problem, we can consider a special prototype sentence '$\emptyset$' which can be edited into any sentence, and draw this special prototype with probability $p_\emptyset$. Concretely, we write:

$$\begin{aligned} p(x) &= \sum_{x' \in \mathcal{X} \cup \{\emptyset\}} p_{\text{edit}}(x|x') p_{\text{prior}}(x') \\ &= (1 - p_\emptyset) \sum_{x' \in \mathcal{X}} \frac{1}{|\mathcal{X}|} p_{\text{edit}}(x|x') + p_\emptyset \, p_{\text{NLM}}(x). \end{aligned}$$

This linearly smoothes between our edit model (p_{edit}) and the NLM (p_{NLM}) since our decoder is identical to the NLM, and thus conditioning on the special $\emptyset$ token reduces to using a NLM.

Empirically, we observe that even small values of $p_\emptyset$ produces low perplexity (Figure 7) corresponding to the observation that smoothing of NEURALEDITOR is only necessary to avoid degenerate log-likelihoods on a very small subset of the test set.

References

E. Agirre, C. Banea, C. Cardie, D. M. Cer, M. T. Diab, A. Gonzalez-Agirre, W. Guo, R. Mihalcea, G. Rigau, and J. Wiebe. 2014. SemEval-2014 Task 10: Multilingual semantic textual similarity. In *International Conference on Computational Linguistics (COLING)*, pages 81–91.

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(0):1137–1155.

S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. 2016. Generating sentences from a continuous space. In *Computational Natural Language Learning (CoNLL)*, pages 10–21.

C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

C. Doersch. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

J. Gu, Y. Wang, K. Cho, and V. O. Li. 2017. Search engine guided non-parametric neural machine translation. *arXiv preprint arXiv:1705.07267*.

U. Hahn and I. Mani. 2000. The challenges of automatic summarization. *Computer*, 33.

T. B. Hashimoto, D. Alvarez-Melis, and T. S. Jaakkola. 2016. Word embeddings as metric recovery in semantic spaces. *Transactions of the Association for Computational Linguistics (TACL)*, 4:273–286.

J. R. Hayes and L. S. Flower. 1986. Writing research and the writer. *American psychologist*, 41(10):1106–1113.

H. He, A. G. II, J. Boyd-Graber, and H. D. III. 2015. Syntax-based rewriting for simultaneous machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 55–64.

K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Association for Computational Linguistics (ACL)*, pages 690–696.

M. Hodosh, P. Young, and J. Hockenmaier. 2013. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research (JAIR)*, 47:853–899.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. 1999. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233.

N. Kalchbrenner and P. Blunsom. 2013. Recurrent continuous translation models. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1700–1709.

D. P. Kingma and M. Welling. 2014. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

P. Kuznetsova, V. Ordonez, A. C. Berg, T. L. Berg, and Y. Choi. 2013. Generalizing image captions for image-text parallel corpus. In *Association for Computational Linguistics (ACL)*, pages 790–796.

J. Li, M. Galley, C. Brockett, J. Gao, and W. B. Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Human Language Technology and North American Association for Computational Linguistics (HLT/NAACL)*, pages 110–119.

R. Mason and E. Charniak. 2014. Domain-specific image captioning. In *Computational Natural Language Learning (CoNLL)*, pages 2–10.

T. Mikolov, K. Chen, G. Corrado, and Jeffrey. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

T. Mikolov, W. Yih, and G. Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Human Language Technology and North American Association for Computational Linguistics (HLT/NAACL)*, volume 13, pages 746–751.

E. Parzen. 1962. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

A. Ritter, C. Cherry, and W. B. Dolan. 2011. Data-driven response generation in social media. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 583–593.

L. Shao, S. Gouws, D. Britz, A. Goldie, B. Strope, and R. Kurzweil. 2017. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 2210–2219.

Y. Song, R. Yan, X. Li, D. Zhao, and M. Zhang. 2016. Two are better than one: An ensemble of retrieval- and generation-based dialog systems. *arXiv preprint arXiv:1610.07149*.

I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112.

J. B. Tenenbaum, V. D. Silva, and J. C. Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science*, pages 2319–2323.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin.

2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

A. T. Wood. 1994. Simulation of the von mises fisher distribution. *Communications in statistics-simulation and computation*, pages 157–164.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yelp. 2017. Yelp Dataset Challenge, Round 8. `https://www.yelp.com/dataset_challenge`.