
Learning Sparse Prototypes for Text Generation

Junxian He[†], Taylor Berg-Kirkpatrick[‡], Graham Neubig[†]

[†]Carnegie Mellon University; [‡]University of California San Diego
`{junxianh,gneubig}@cs.cmu.edu, tberg@eng.ucsd.edu`

Abstract

Prototype-driven text generation uses non-parametric models that first choose from a library of sentence “prototypes” and then modify the prototype to generate the output text. While effective, these methods are inefficient at test time as a result of needing to store and index the entire training corpus. Further, existing methods often require heuristics to identify which prototypes to reference at training time. In this paper, we propose a novel generative model that automatically learns a *sparse* prototype support set that, nonetheless, achieves strong language modeling performance. This is achieved by (1) imposing a sparsity-inducing prior on the prototype selection distribution, and (2) utilizing amortized variational inference to *learn* a prototype retrieval function. In experiments, our model outperforms previous prototype-driven language models while achieving up to a 1000x memory reduction, as well as a 1000x speed-up at test time. More interestingly, we show that the learned prototypes are able to capture semantics and syntax at different granularity as we vary the sparsity of prototype selection, and that certain sentence attributes can be controlled by specifying the prototype for generation.¹

1 Introduction

Language models (LMs) predict a probability distribution over text, and are a fundamental technology widely studied in the natural language processing (NLP) community (Bengio et al., 2003; Merity et al., 2018; Dai et al., 2019). Modern LMs are almost exclusively based on *parametric* recurrent (Mikolov et al., 2010; Sundermeyer et al., 2012) or self-attentional (Vaswani et al., 2017; Al-Rfou et al., 2019) neural networks. These models are of interest scientifically as one of the purest tests of our ability to capture the intricacies of human language mathematically (Linzen et al., 2016; Kuncoro et al., 2017; Petroni et al., 2019). They also have broad downstream applications in generating text in systems such as machine translation (Bahdanau et al., 2015), summarization (Rush et al., 2015), or dialog generation (Sordoni et al., 2015), as well as in the unsupervised representation learners that now power many applications in NLP (Devlin et al., 2018; Liu et al., 2019; Yang et al., 2019).

However, there has been a recent move towards *non-parametric* neural LMs (Guu et al., 2018; Khandelwal et al., 2020b) that generate sentences by first selecting examples from an external datastore. For instance, Khandelwal et al. (2020b) model the token-level probability at test time by interpolating the language model with a kNN distribution from the nearest context-token pairs in the datastore, while Guu et al. (2018) store external memories on sentence level and feature a prototype-then-edit process of (1) selecting a *prototype* sentence from a the prototype datastore, and (2) editing this prototype to the final desired output. In this paper, we focus on the prototype-then-edit model family which is a lot lighter relatively in terms of memory and time cost at test time.

Intuitively, these non-parametric LMs are attractive because they help remove some of the pressure on the parametric model to memorize the entirety of the language it must model. These intuitive advantages are also borne out in superior performance on language modeling tasks (Guu et al.,

¹Code is available at <https://github.com/jxhe/sparse-text-prototype>.

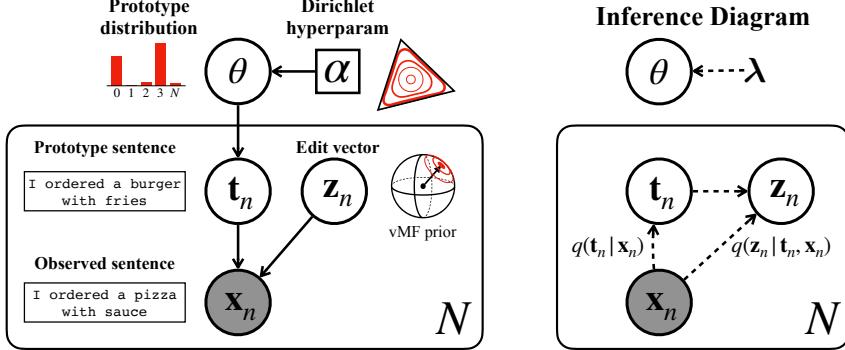


Figure 1: **Left:** the proposed generative model to generate data by editing prototypes. Shaded circles denote the observed variables and unshaded denote the latents. Prototypes are sampled from a sparse prototype distribution which itself is a random variable sampled from a Dirichlet prior distribution. **Right:** the inference diagram of the model, with $q(t_n|x_n)$ being the prototype retriever and $q(z_n|t_n, x_n)$ being the inverse editor.

2018; Khandelwal et al., 2020b), as well as down-stream applications such as dialogue response generation (Weston et al., 2018; Wu et al., 2019), machine translation (Gu et al., 2018; Bapna & Firat, 2019; Khandelwal et al., 2020a), and code generation (Hashimoto et al., 2018; Hayati et al., 2018). In addition, the prototypes and continuous representations of the edits in prototype-based models lend an element of interpretability to the modeling process. On the down side, however, previous prototype-driven generation methods usually need to store and index a large prototype candidate pool (in general the whole training dataset), leading to significant issues with memory and speed efficiency at test time.

In this paper, we hypothesize that, in fact, a *small* set of prototypes is sufficient to achieve the great majority of the gains afforded by such non-parametric models. Intuitively, in a large corpus many sentences look very similar and may be represented by minor transformations of a single prototype sentence. For example, the sentence “I ordered a burger with fries” can serve as the prototype for data samples with the form “I ordered [NOUN PHRASE] with [NOUN PHRASE]”. This is evidenced by Guu et al. (2018)’s observation that 70% of the test set in the Yelp restaurant review corpus (Yelp, 2017) is within word-token Jaccard distance 0.5 of one training sentence.

To take advantage of this intuition, we propose a novel generative model that samples prototypes from a *latent* prototype distribution, which itself is sampled from a symmetric Dirichlet prior, as shown in Figure 1 (Section 3.1). The Dirichlet prior with appropriate hyperparameters is able to encourage a *sparse* prototype selection distribution, allowing us to reduce the prototype support set at test time to greatly improve efficiency. Moreover, we utilize amortized variational inference (Kingma & Welling, 2013) to train our model, which introduces a learnable *prototype retriever* to identify prototypes useful for generating each sentence (Section 3.2). This is different from (Guu et al., 2018) where prototypes for each sentence are fixed before training through edit distance heuristics.

We evaluate our approach on the MSCOCO (Lin et al., 2014) and Yelp restaurant review (Yelp, 2017) corpora. Our method is able to improve perplexity over the neural language model baseline by up to 14 points and previous neural editor model by 6 points while achieving over 1000x memory savings and a 1000x speedup at test time (Section 4.2). Interestingly, we find that the learned prototypes are able to represent different features when varying sparsity levels – a strong sparsity prior forces the model to share prototypes and the induced prototypes turn out to represent more generic features (e.g. syntactic form of the sentence). On the text generation side, our model is able to generate sentences that resemble the given prototype while allowing for smooth interpolation on the edit space as well (Section 4.3).

2 Background

The prototype-then-edit framework defines a non-parametric way to augment text generation models. Formally, given a corpus $X = \{x_n\}_{n=1}^N$,² the model generates each observed example x_n by: (1) retrieving a prototype sequence t_n , (2) generating a continuous edit representation z_n , and (3)

²Below, we sometimes ignore the subscript to simplify notation when there is no confusion.

generating \mathbf{x}_n conditioned on \mathbf{t}_n and \mathbf{z}_n . These intermediate prototype and edit representation variables can depend on extra context in conditional generation tasks (Hodosh et al., 2013; Gu et al., 2018), or are randomly sampled in unconditioned language modeling. In this paper, we focus on the latter, but our methods could likely be applied to the former as well.

For unconditioned language modeling, Guu et al. (2018) define the data likelihood as:

$$p(\mathbf{X}) = \prod_n \sum_{\mathbf{t}_n} \int_{\mathbf{z}_n} p(\mathbf{z}_n) p(\mathbf{t}_n) p_\gamma(\mathbf{x}_n | \mathbf{t}_n, \mathbf{z}_n) d\mathbf{z}_n, \quad (1)$$

where $p(\mathbf{t}_n)$ is the prior distribution over prototypes and defined as a uniform distribution over *all* training examples, $p(\mathbf{z}_n)$ is a continuous distribution over the edit vector, and $p_\gamma(\mathbf{x}_n | \mathbf{t}_n, \mathbf{z}_n)$ represents a sequence-to-sequence model parameterized by γ . This model is referred to as the *neural editor*. Guu et al. (2018)'s stated goal of the neural editor is to take direct advantage of training examples to improve language modeling performance while capturing interpretable semantic or syntactic properties in the latent prototype and edit vector variables. However, because the prototypes are selected from the *entire* training dataset, such a formulation sacrifices memory and speed efficiency due to the necessity of indexing and searching every training example at test time. In the following section, we detail our approach to mitigate this issue through the learning of *sparse* prototypes.

3 Method

First we present our proposed generative model, then we describe the learning and inference techniques for this model class.

3.1 Model Structure

In the previous formulation, Eq. 1, maintaining the entire training dataset at test time is necessary due to assuming a uniform prior over prototypes $p(\mathbf{t})$. Motivated by the hypothesis that a much smaller prototype set would suffice to achieve comparable performance, however, we believe that $p(\mathbf{t})$ can be a sparse distribution where the probability mass concentrates on only a few representative prototypes. Since which training examples are representative as prototypes is unknown in advance, we propose to model the prototype distribution $p(\mathbf{t})$ as a latent variable, endowing the model with freedom to infer a sparse prototype posterior automatically. We define $\theta \equiv p(\mathbf{t})$ and further assume that the latent prototype distribution θ is sampled from a prior distribution $p_\alpha(\theta)$ (detailed below) which is able to encourage a sparse probability distribution, given appropriate hyperparameters. The graphical model is depicted in Figure 1, which gives the following joint likelihood:

$$p(\{\mathbf{x}_n, \mathbf{t}_n, \mathbf{z}_n\}_{n=1}^N, \theta; \alpha, \gamma) = p_\alpha(\theta) \prod_n p(\mathbf{t}_n | \theta) p(\mathbf{z}_n) p_\gamma(\mathbf{x}_n | \mathbf{t}_n, \mathbf{z}_n). \quad (2)$$

The log marginal likelihood of the data, which we will approximate during training is:

$$\log p(\mathbf{X}; \gamma, \alpha) = \log \int_\theta p_\alpha(\theta) \left[\prod_n \sum_{\mathbf{t}_n} \int_{\mathbf{z}_n} p(\mathbf{t}_n | \theta) p(\mathbf{z}_n) p_\gamma(\mathbf{x}_n | \mathbf{t}_n, \mathbf{z}_n) d\mathbf{z}_n \right] d\theta. \quad (3)$$

This is a general framework for learning sparse prototypes that we refer to as the *sparse neural editor*, and in this work we specifically experiment with the following parameterization to instantiate this model class:

Prior over prototype distribution $p_\alpha(\theta)$: We employ the Dirichlet distribution as $p_\alpha(\theta) \propto \prod_{k=1}^N \theta_k^{\alpha_k - 1}$. The support of Dirichlet distribution is the standard $N - 1$ probability simplex. Here we use the *symmetric* Dirichlet distribution which has the same α value for all components since we have no prior knowledge favoring one component over another. α is positive and also referred to as the concentration parameter, where smaller α prefers a sparser prototype distribution θ , with $\alpha = 1$ equivalent to a uniform distribution over the probability simplex. In our experiments, we often choose $\alpha < 1$ to encourage sparsity.

Prior over edit vector $p(\mathbf{z})$: We follow Guu et al. (2018) and utilize a von-Mises Fisher (vMF) distribution to model $p(\mathbf{z})$. The vMF distribution places mass on the surface of the unit sphere, and is parameterized by the mean direction vector μ and concentration parameter κ as $\text{vMF}(\cdot | \mu, \kappa)$. Thus,

information about the edit is captured through the directions of different unit vector samples. Xu & Durrett (2018) empirically show that the vMF distribution has the advantage of overcoming posterior collapse that plagues a large amount of previous work in latent variable models of text (Bowman et al., 2016; He et al., 2019). While Guu et al. (2018) add additional randomness on the norm of edit vectors by multiplying the vMF distribution with another uniform distribution, we sample edit vectors from a uniform vMF distribution directly, which simplifies the model but we nonetheless found sufficient to obtain competitive results. Formally, we define $p(\mathbf{z}) = \text{vMF}(\mathbf{z}|\cdot, 0)$.

The editor $p_\gamma(\mathbf{x}|\mathbf{t}, \mathbf{z})$: Generally $p_\gamma(\mathbf{x}|\mathbf{t}, \mathbf{z})$ can be parameterized by any standard Seq2Seq model with the edit vector \mathbf{z} incorporated. To compare with Guu et al. (2018) directly in the experiments, in this work we adopt the same attentional LSTM architecture (Hochreiter & Schmidhuber, 1997). \mathbf{z} is utilized to predict the initial hidden state of the decoder and concatenated to the input for the decoder.

3.2 Learning and Inference

Ideally the log marginal likelihood in Eq. 3 should be optimized during training. However, computation is intractable due to marginalization of latent variables, and we resort to amortized variational inference (Kingma & Welling, 2013), optimizing its evidence lower bound (ELBO) instead:

$$\begin{aligned} \log p(\mathbf{X}; \boldsymbol{\gamma}, \alpha) &\geq \mathcal{L}_{\text{ELBO}}(\mathbf{X}; \boldsymbol{\gamma}, \alpha, \phi_{t|x}, \phi_{z|t,x}, \boldsymbol{\lambda}) \\ &= \sum_n \left\{ \underbrace{\mathbb{E}_{q_{\phi_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)q_{\phi_{z|t,x}}(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n)}[\log p_\gamma(\mathbf{x}_n|\mathbf{t}_n, \mathbf{z}_n)]}_{\text{reconstruction log likelihood } \mathcal{L}_{\text{rec}}} \right. \\ &\quad - \mathbb{E}_{q_{\phi_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)}[D_{\text{KL}}(q_{\phi_{z|t,x}}(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n)||p(\mathbf{z}_n))] \\ &\quad \left. - \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\text{KL}}(q_{\phi_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))] \right\} - D_{\text{KL}}(q_{\boldsymbol{\lambda}}(\theta)||p_{\alpha}(\theta)), \end{aligned} \quad (4)$$

where q represents the variational distribution to approximate the model posterior distribution and admits the following factorization form:

$$q(\theta, \{\mathbf{t}_n, \mathbf{z}_n\}_{n=1}^N | \mathbf{X}; \boldsymbol{\lambda}, \phi_{t|x}, \phi_{z|t,x}) = q_{\boldsymbol{\lambda}}(\theta) \prod_n q_{\phi_{t|x}}(\mathbf{t}_n|\mathbf{x}_n) q_{\phi_{z|t,x}}(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n). \quad (5)$$

Note that we make conditional independence assumption between θ and other latent variables in q to simplify the approximate posterior, following common practice in traditional mean field variational inference. The inference diagram is depicted in Figure 1. The optimal $q_{\boldsymbol{\lambda}}(\theta)$ to maximize Eq. 4 is a Dirichlet distribution parameterized by $\boldsymbol{\lambda} \in \mathbb{R}_+^N$ (proof is in Appendix A), i.e., $q_{\boldsymbol{\lambda}}(\theta) = \text{Dir}(\theta; \boldsymbol{\lambda})$.³ And $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x}) = \text{Cat}(\mathbf{t}; f_{\phi_{t|x}}(\mathbf{x}))$, the *prototype retriever*, is a categorical distribution over training examples parameterized by a neural network $f_{\phi_{t|x}}(\mathbf{x})$. We assume $q_{\phi_{z|t,x}}(\mathbf{z}|\mathbf{t}, \mathbf{x})$, the *inverse neural editor*, is a vMF distribution $q_{\phi_{z|t,x}}(\mathbf{z}|\mathbf{t}, \mathbf{x}) = \text{vMF}(g_{\phi_{z|t,x}}(\mathbf{t}, \mathbf{x}), \kappa)$ where the mean direction parameter is from an encoder g that encodes \mathbf{t} and \mathbf{x} parameterized by $\phi_{z|t,x}$, and the scalar concentration parameter κ is a hyperparameter. Pre-fixing κ results in a constant KL divergence term associated with \mathbf{z} and proves to be effective to mitigate the posterior collapse issue (Xu & Durrett, 2018) where \mathbf{x} and \mathbf{z} become independent. Yet there might be still posterior collapse on \mathbf{t} where, for example, the prototype retriever always predicts a uniform distribution or a degenerate distribution concentrated on a certain prototype regardless of \mathbf{x} . To overcome this issue, we follow (Li et al., 2019) to combine annealing (Bowman et al., 2016) and free-bits techniques (Kingma et al., 2016) and apply them to the term $\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\text{KL}}(q_{\phi_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))]$.

Notably, the variational distribution family defined in Eq. 5 admits tractable closed-form expressions of all three KL divergence terms in Eq. 4 (detailed derivations and expressions are in Appendix A). To compute the reconstruction log likelihood \mathcal{L}_{rec} , expectations over \mathbf{z} can be efficiently approximated by the reparameterization trick for the vMF distribution (Guu et al., 2018). However, the prototype \mathbf{t} is discrete and non-differentiable, and summing over all prototypes \mathbf{t} to compute \mathcal{L}_{rec} is infeasible due to the evaluation burden of $\log p_\gamma(\mathbf{x}|\mathbf{t}, \mathbf{z})$. Thus, we use the REINFORCE algorithm (Williams, 1992) to compute the gradients of $\phi_{t|x}$ contributed from \mathcal{L}_{rec} as:

$$\frac{\partial \mathcal{L}_{\text{rec}}}{\partial \phi_{t|x}} = \frac{1}{L} \sum_{l=1}^L \left(\mathbb{E}_{q_{\phi_{z|t,x}}(\mathbf{z}|\mathbf{t}^{(l)}, \mathbf{x})} [\log p_\gamma(\mathbf{x}|\mathbf{t}^{(l)}, \mathbf{z})] - b \right) \frac{\partial \log q_{\phi_{t|x}}(\mathbf{t}^{(l)}|\mathbf{x})}{\partial \phi_{t|x}}, \quad (6)$$

³ $q_{\boldsymbol{\lambda}}(\theta)$ is not symmetric and $\boldsymbol{\lambda}$ is a vector.

where $\mathbf{t}^{(l)}$ are samples from $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$. We use an average reward from L samples as the baseline b . The neural parameters $\gamma, \phi_{t|x}, \phi_{z|x}$ are updated with stochastic gradient descent to maximize Eq. 4. With respect to the posterior Dirichlet parameter λ , we found in preliminary experiments that classic gradient descent was unable to update it effectively – λ was updated too slowly and the Dirichlet prior became decoupled with the model. Thus, we instead update λ with stochastic variational inference (SVI, Hoffman et al. (2013)) based on the formula of the optimal λ^* given $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ (derivations can be found in Appendix A):

$$\lambda_k^* = \alpha + \sum_{n=1}^N q_{\phi_{t|x}}(\mathbf{t}_n = \mathbf{x}_k | \mathbf{x}_n). \quad (7)$$

It is infeasibly expensive to keep λ optimal under current $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ at each training step, as it would involve summing over all training examples. Thus we perform SVI, which uses a batch of examples to approximate Eq. 7, leading to the following update form:

$$\lambda_k^{(t)} = (1 - \rho_t) \lambda_k^{(t-1)} + \rho_t \left(\alpha + \frac{N}{B} \sum_{i=1}^B q_{\phi_{t|x}}(\mathbf{t}_i = \mathbf{x}_k | \mathbf{x}_i) \right), \quad \rho_t = (t + \sigma)^{-\tau}, \quad (8)$$

where B is the batch size, ρ_t is the step-size at iteration t , $\tau \in (0.5, 1]$ is the forgetting rate, and $\sigma \geq 0$ is the delay parameter to down-weight early iterations.

We note that our training algorithm is different from Guu et al. (2018) in that we use a learnable prototype retriever $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ to derive a lower bound as the objective while Guu et al. (2018) directly approximate marginalization over \mathbf{t} . They use heuristics to fix the prototype set for each \mathbf{x} to be examples similar to \mathbf{x} in terms of edit distance, which might produce suboptimal prototypes for the generative model and also does not permit the learning of sparse prototype support.

Sparsity and scalability: After training we expect to be able to infer a *sparse* prototype distribution θ with most components being almost zero, based on which we can prune and store the entries over a particular probability threshold only, improving memory- and time-efficiency at test time. Specifically, we compute mean of θ under the Dirichlet posterior: $\mathbb{E}_{q_{\lambda}(\theta)}[\theta_k] = \lambda_k / \sum_i \lambda_i$, and then take the largest M entries that occupy 90% of the probability mass. At test time, we only maintain these M prototypes and the prototype retriever $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ is re-normalized accordingly. One issue present during training is that $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ cannot fit into memory when dealing with large datasets since it is a categorical distribution over all training examples. In this work, we randomly downsample a subset of training data as our prototype library before training if memory is unable to fit all training examples, and learn the sparse prototypes on top of this subsampled corpus. This acts like a rough pre-filtering and in Section 4 we show that it suffices to learn good prototypes and achieve competitive language modeling performance. We leave more advanced techniques to address this issue (e.g. dynamically updating the prototype library) as future work.

Architectures: We now describe the neural architectures we use for the prototype retriever $q(\mathbf{t}|\mathbf{x})$ and inverse neural editor $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$. $q(\mathbf{t}|\mathbf{x})$ is defined as:

$$q(\mathbf{t} = \mathbf{x}_k | \mathbf{x}) \propto \begin{cases} \exp(h(\mathbf{x}_k, \mathbf{x})/\mu) & \text{if } \mathbf{x}_k \neq \mathbf{x} \\ 0 & \text{if } \mathbf{x}_k = \mathbf{x}, \end{cases} \quad h(\mathbf{x}_k, \mathbf{x}) = \text{Embed}(\mathbf{x}_k)^\top \mathbf{W} \text{Embed}(\mathbf{x}), \quad (9)$$

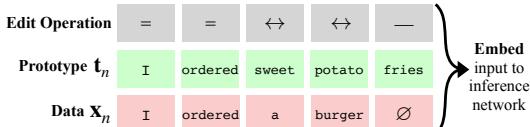


Figure 2: Example of aligned sequences.

where we prevent selecting the data example itself as the prototype during training to avoid overfitting. $\text{Embed}(\cdot)$ is a pretrained sentence encoder, \mathbf{W} is a linear transformation matrix, and μ is a temperature hyperparameter to control the entropy of $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$, which is critical to

stabilize the Reinforce algorithm at the initial training stage. To ease the computation of encoding all training examples at each update step, we fix the parameters of $\text{Embed}(\cdot)$ and update \mathbf{W} only, which proves to be sufficient in our experiments. We use the average embeddings of the last layer in pretrained BERT (Devlin et al. 2018)⁴ as our sentence encoder.

While Guu et al. (2018) uses sum of inserted/deleted word vectors as the mean direction parameter of vMF distribution $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$, we choose a more powerful encoder following recent advances on

⁴We use pretrained uncased BERT base from the transformers library (Wolf et al., 2019).

representing edits (Yin et al., 2019). Specifically, a standard diffing algorithm is run to compute an alignment of tokens in \mathbf{t} and \mathbf{x} , and produces two aligned sequences \mathbf{t}' , \mathbf{x}' and an additional edit sequence that indicates edit operation + (insertion), - (deletion), \leftrightarrow (substitution), and = (equal) at each position. We use \emptyset to denote padding. This is illustrated in Figure 2. Word embeddings of all three sequences are concatenated and fed into a single-layer LSTM to obtain the edit representation.

4 Experiments

Our experiments below are designed to (1) examine the efficacy of the proposed sparse neural editor on language modeling, (2) examine the efficiency of sparse neural editor on memory savings and speed-up at test time, and (3) demonstrate the interpretable semantics/syntax captured by prototypes and edit vectors.

4.1 Setup

We perform experiments on three different-scale datasets to test our method in different scenarios:

- **MSCOCO** (Lin et al., 2014): MSCOCO is an image caption dataset and we only focus on its captions as our data. The average length of captions is 12.6. The sentences do not have complex variations and are easy to find similar sentences as prototypes. We randomly sample 40K sentences as our training data and 4K as validation and test set respectively. This dataset represents a simple and small-scale setting to test our approach.
- **Yelp Medium/Yelp Large**: These datasets consist of sentences from Yelp restaurant reviews (Yelp, 2017) preprocessed by Guu et al. (2018), allowing us to perform a direct comparison with their method. The medium and large datasets consist of 1.5M and 17M sentences respectfully, allowing us to test in moderate and relatively large data settings respectively. Note that Yelp Medium is obtained by further filtering Yelp Large to keep sentences that are generally shorter and have less variations, but the test sets for these two are the same.

Baselines: We mainly consider neural language models (NLM) and the neural editor (Guu et al., 2018) as our baseline. It is worth noting that the neural editor model does not have likelihood defined on test sentences that are not similar to any example in the prototype library, thus it is necessary to interpolate with another NLM at test time for smoothing purposes, while our model is able to be used on its own. Note that we only report the neural editor baseline on Yelp Large since their public code is not ready to run on other datasets due to the required pre-built index to retrieve prototypes.

Evaluations: We evaluate language modeling performance with perplexity (PPL). For our model, we approximate the log marginal data likelihood through 1000 importance-weighted latent variable samples (Burda et al., 2015) and compute PPL based on this likelihood. At test time our approach prunes and has access to only M prototypes that occupy 90% probability mass of the posterior prototype distribution as described in Section 3.2. We report M as “#prototypes”. To have an intuitive notion about how similar the prototype and data examples are, we compute average of smoothed sentence-level BLEU scores (Lin & Och, 2004) of the data examples on the validation set with their most likely prototype as the reference. We also report BLEU scores based on part-of-speech sequences (POS-BLEU)⁵ to view the similarity from a more syntactic perspective. Test speed is evaluated on a single Nvidia 1080 Ti GPU.

Hyperparameters: We try different Dirichlet prior parameters α to control the sparsity and report different sparsity settings for all the datasets. The temperature parameter μ in the prototype retriever $q(\mathbf{t}|\mathbf{x})$ is tuned on the MSCOCO validation data and set as 0.3 for all datasets. The concentration parameter κ of the vMF distribution is tuned and set as 30 for MSCOCO and Yelp Medium and 40 for Yelp Large. The number of Reinforce samples L is set as 10 across all datasets. We sample 50K examples for Yelp Medium and 100K examples for Yelp Large as our training prototype library to address the training memory issue discussed in Section 3.2. We employ the same attentional LSTM Seq2Seq model as in Guu et al. (2018) to parameterize $p_\gamma(\mathbf{x}|\mathbf{t}, \mathbf{z})$ for a direct comparison. Our implementation is based on the fairseq toolkit (Ott et al., 2019) and complete hyperparameter settings can be found in Appendix B.

⁵POS tagging is performed using the Stanza library (Qi et al., 2020).

Table 1: Results on three datasets. Numbers in the parentheses indicate the percentage of prototypes over all training examples. BLEU score is computed by comparing validation examples against their most likely prototypes. POS-BLEU represents the BLEU score on part-of-speech sequences. We also list BLEU scores from random prototype retrieval as a reference point. Results in the starred entry (*) are obtained by running the public code of the neural editor.

Dataset	Model	PPL \downarrow	#prototypes	test speed (sent/s) \uparrow	BLEU	POS-BLEU																				
MSCOCO	random retrieval	—	—	—	10.9	31.6																				
	NLM	20.0	—	3714	—	—																				
	Sparse Neural Editor ($\alpha = 10^{-3}$)	18.9	25 (0%)	388	13.2	38.8																				
	Sparse Neural Editor ($\alpha = 0.1$)	18.6	778 (2%)	313	17.2	42.5																				
	Sparse Neural Editor ($\alpha = 0.2$)	19.0	16K (40%)	250	20.9	46.7																				
Yelp Medium	Sparse Neural Editor ($\alpha = 0.3$)	19.2	22K (56%)	217	22.2	47.9																				
	random retrieval	—	—	—	8.1	17.8																				
	NLM	74.7	—	236	—	—																				
	Sparse Neural Editor ($\alpha = 10^{-3}$)	63.6	77 (0%)	157	12.3	24.7																				
	Sparse Neural Editor ($\alpha = 0.5$)	61.9	1.5K (0.1%)	107	21.6	38.4																				
Yelp Large	Sparse Neural Editor ($\alpha = 10$)	63.2	31K (2.1%)	95	29.9	48.3																				
	random retrieval	—	—	—	6.6	16.0																				
	NLM	34.2	—	272	—	—																				
	Sparse Neural Editor ($\alpha = 0.7$)	30.2	2K (0.01%)	108	10.5	24.8																				
	Sparse Neural Editor ($\alpha = 10$)	30.3	5.5K (0.03%)	98	10.8	25.3																				
Interpolated w/ NLM																										
<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;">Neural Editor [Guu et al., 2018]</td> <td style="width: 10%;">26.9</td> <td style="width: 30%;">17M (100%)</td> <td style="width: 10%;">—</td> <td style="width: 10%;">—</td> <td style="width: 10%;">—</td> <td style="width: 10%;">—</td> </tr> <tr> <td>Neural Editor (our runs)*</td> <td>31.2</td> <td>17M (100%)</td> <td>0.1</td> <td>—</td> <td>—</td> <td>—</td> </tr> <tr> <td>Sparse Neural Editor ($\alpha = 0.7$)</td> <td>20.2</td> <td>2K (0.01%)</td> <td>108</td> <td>10.5</td> <td>24.8</td> <td>—</td> </tr> </table>						Neural Editor [Guu et al., 2018]	26.9	17M (100%)	—	—	—	—	Neural Editor (our runs)*	31.2	17M (100%)	0.1	—	—	—	Sparse Neural Editor ($\alpha = 0.7$)	20.2	2K (0.01%)	108	10.5	24.8	—
Neural Editor [Guu et al., 2018]	26.9	17M (100%)	—	—	—	—																				
Neural Editor (our runs)*	31.2	17M (100%)	0.1	—	—	—																				
Sparse Neural Editor ($\alpha = 0.7$)	20.2	2K (0.01%)	108	10.5	24.8	—																				

4.2 Results

Results are shown in Table I. Our sparse neural editor outperforms the NLM baseline across all datasets in terms of PPL, often by a large margin. When interpolated with an NLM at test time, our method outperforms the NLM baseline by 14 PPL points and neural editor by 6.7 PPL points.⁶ This effect is also observed in (Guu et al., 2018) – prototype-driven language models are especially strong at modeling test sentences that have similar prototypes but relatively weak at modeling others, thus interpolation with a normal NLM is likely to help. Furthermore, in line with the goal of this work to learn a sparse prototype set, our method is able to achieve superior language modeling performance while utilizing only a small fraction of training examples as prototypes. This verifies our hypothesis that a sparse prototype set suffices for such non-parametric language modeling. Also, sparsity learned in our model allows for over a 1000x memory savings and 1000x speed-up⁷ at test time on Yelp Large compared with a previous neural editor that memorizes all training examples.

Table I demonstrates the trend that a smaller Dirichlet hyperparameter α leads to a sparser prototype set, which agrees with our expectation. BLEU scores are also improved as α increases, implying that the less sparse the prototype set the closer the match between the sentence and its prototype. Interestingly, the BLEU score on Yelp Large is a bit low and the model tends to generally favor sparse prototypes. We suspect that this is because it is difficult to learn prototypes that capture fine-grained semantic features and large sentence variations among 17M examples with a limited prototype memory budget, thus the model has to learn prototypes that represent more generic shared features among examples to reach an optimum – for example, the syntactic feature as somewhat reflected by the decent POS-BLEU scores.

We want to emphasize that different sparsity may lead to different notions of prototypes and it is hard to judge which one should be preferred – memorizing more prototypes pays cost on language modeling performance and does not necessarily produce better PPL. Also, prototypes that are “less similar” to the examples on the superficial level but capture coarse-grained features may have potentially interesting application on sentence generation since the model is able to generate more diverse output conditioned on prototypes.

⁶For a fair comparison, we interpolate with the same pretrained NLM from (Guu et al., 2018).

⁷We include the time to retrieve prototypes for new test sentences when computing test speed. We use Guu et al. (2018)'s public implementation of the neural editor, where the computation of edit distance between all training examples and test sentences to find nearest neighbors accounts for much of the runtime. More efficient implementation of this operation, for example through tries, may speed this to some extent.

Table 2: Number of matching tokens between examples and their prototypes on the Yelp Medium validation set. Results are reported in cluster of POS tags. Relative changes that are larger than the overall change are bolded.

Model	Overall	NOUN	DET	AUX	PRON	ADJ	VERB	CCONJ
Sparse Neural Editor (31K prototypes)	91.2K	14.4K	9.6K	9.3K	9.0K	7.2K	6.4K	5.5K
Sparse Neural Editor (1.5K prototypes)	74.7K	9.9K	8.5K	8.2K	7.3K	5.6K	4.4K	5.0K
Relative Change	-18.1%	-31.3%	-11.5%	-11.8%	-18.9%	-22.2%	-31.3%	-9.1%

Table 3: Qualitative examples of prototypes when using denser and sparser prototype supports.

Data Examples	Prototypes
the best corned beef hash i 've ever had !	(dense) the best real corned beef hash i 've had . (sparse) the chicken satay is the best i 've ever had .
the grilled chicken was flavorful , but too flavorful .	(dense) the chicken was moist but it lacked flavor . (sparse) my sandwich was good but the chicken was a little plain .
i asked her what time they close and she said <cardinal> o'clock .	(dense) i asked what time they closed <date> , and was told <cardinal> . (sparse) we asked how long the wait was and we were informed it would be <time> .

Table 4: Qualitative examples from the MSCOCO dataset on interpolated sentence generation given the prototype. The first row is the given prototype, the second-row and the last-row sentences are obtained by sampling edit vectors from the prior, the rest three sentences are generated by interpolating between the two edit vectors.

Prototype: A man is using a small laptop computer	Prototype: A cat sitting on a sidewalk behind a bush
A man is using his laptop computer with his hands on the keyboard	A cat laying on top of a wooden bench
A man is using a laptop computer with his hands on the keyboard	A cat standing next to a tree in a park
A man is using a laptop computer while sitting on a bench	Two cats sitting on a bench near a park bench
A man is using a laptop computer in the middle of a room	A dog sitting on a bench near a park bench
A young man is using a laptop computer in the middle of a room	A dog sitting on a bench near a park bench

4.3 Analysis

How do prototypes change when they grow sparser? In Table 1 we notice the BLEU scores usually drop when the prototype set is sparser, implying the learned prototypes change in some way under different sparsity settings. Here we take the Yelp Medium dataset as an example to analyze how prototypes are trained to capture sentence attributes differently in the sparse ($\alpha = 0.5$) and relatively dense ($\alpha = 10$) situations. Specifically, we align prototype and example sequences to minimize edit distance, and focus on words that were aligned between the two sequences. This allows us to obtain a notion of what kind of words are more likely to be kept the same as the prototype during the model’s editing process and how this pattern changes in different sparsity settings. We cluster these matched words in terms of POS tag and report the most common ones.⁸

Results are shown in Table 2. While the overall number of matching tokens decreases as the prototype set becomes sparser, the content words exhibit a more drastic change (e.g. the nouns, adjectives, and verbs). In contrast, the function words experience a moderate decrease only (e.g. the determiners, auxiliaries, and coordinating conjunctions). This shows that the model tends to learn prototypes that drop fine-grained semantic distinctions but keep the same general syntax when a sparsity constraint is enforced, which is not surprising since it is difficult for a limited number of prototypes to capture large semantic variations in a large dataset. We list qualitative examples in Table 3 to demonstrate this phenomenon, where some content words such as “beef” or “chicken” differ between data examples and prototypes in the sparser setting, yet these words do not change in the dense setting.

Ablation Study on Pre-clustering Prototypes: There is a simple method that is able to endow the neural editor baseline with sparse prototypes – pre-clustering the training examples and using only a subset of the them as the prototype pool during training and test. To compare the effectiveness of this heuristic sparse prototypes and our learned sparse prototypes, we experiment this baseline on MSCOCO dataset and pre-cluster 778 prototypes that correspond to our setting $\alpha = 0.1$. Concretely, we run k-means to obtain 778 clusters of training sentences embedded by Sentence-BERT (Reimers & Gurevych, 2019) which produces state-of-the-art semantic sentence embeddings. Then for each cluster we identify one prototype whose embedding is the closest to the cluster centroid. We use these 778 prototypes in the neural editor model (Guu et al., 2018) and train with the same objective

⁸Note that alignment is performed on word sequences instead of POS sequences.

Table 5: Comparison with Neural Editor that uses heuristic pre-clustering to select a sparse prototype support. The results are on MSCOCO dataset.

Model	PPL \downarrow	#prototypes	BLEU	POS-BLEU
NLM	20.0	—	—	—
Neural Editor (pre-cluster)	19.5	778	17.9	40.4
Sparse Neural Editor	18.6	778	17.2	42.5

as Guu et al. (2018)⁹. Results are shown in Table 5. Our method outperforms the neural editor with the same number of prototypes and presents similar BLEU scores, which demonstrates the superiority of learned prototypes over the heuristically selected prototypes.

Interpolation on the edit vector space: We take our model with 1.5K prototypes on the MSCOCO dataset (i.e. $\alpha = 0.1$) and perform sentence interpolation in edit vector space. Specifically, we sample two edit vectors from the uniform vMF prior to produce two sentences for each prototype with beam search decoding, then we perform spherical linear interpolation of the two edit vectors to generate interpolated sentences in-between. Qualitative examples in Table 4 (more examples in Appendix C) show that the edit vectors are able to *smoothly* capture minor edits over the given prototypes.

5 Conclusion

In this work, we propose a novel generative model that discovers a sparse prototype set automatically by optimizing a variational lower bound of the log marginal data likelihood. We demonstrate its effectiveness on language modeling and its efficiency advantages over previous prototype-driven generative models. The framework proposed here might be generalized to automatically discover salient prototypes from a large corpus. New kinds of prototype structure in text might be discovered through either injecting different biases into the model (e.g. sparsity biases in this paper), or incorporating prior knowledge into the prototype library before training. Finally, the approach might be easily extended to conditional generation (e.g. with the edit vectors depending on other data input), and we envision that inducing a sparse prototype set in this case may potentially facilitate controlling text generation through prototypes. We leave exploration in this direction as our future work.

Broader Impact

Our approach is likely to benefit researchers or practitioners who are interested in generating text through machines in practical scenarios such as writing news articles given facts, describing stock trends with stock data, generating headlines, etc. Many such applications have a small number of templates for generated text. On the one hand, our model is able to automatically induce those representative prototypes from a large corpus, helping knowing the salient prototypes or templates to help human writers to start with. On the other hand, our approach can be easily extended to these conditional text generation task directly, for example, with the edit vector depending on the input data instead of from a uniform distribution. Such way potentially allows our model to control the prototypes and directly generate text conditioned on the input data as well. Furthermore, the prototype library may be further explored in other formats in addition to training examples, opening a door to more flexible control under different notions of “prototypes”.

With respect to possible disadvantages from this research from a societal perspective, as a contribution to the widely studied field of language modeling, the proposed method inherits some of the risks of the field as a whole. These may include models being used maliciously to create fake content (Zellers et al., 2019), or models being used in earnest being manipulated through adversarial attacks to generate undesirable or defamatory content (Wallace et al., 2019). With respect to the latter, as our method is more interpretable due to its use of readable prototypes, we actually expect that it may be *more* robust to adversarial attacks, and these attacks may be easier for human auditors to detect or defuse when they occur. However, this is speculation, and would have to be confirmed by further experiments.

⁹The prior over edit vector $p(\mathbf{z})$ and the inverse neural editor $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$ in sparse neural editor is different from the original neural editor model. For a fair comparison we use $p(\mathbf{z})$ and $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$ described in this paper for the neural editor baseline here.

Acknowledgements

This work was supported in part by the DARPA GAILA project (award HR00111990063), and a gift of computation credits from Amazon AWS. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government or Amazon.

References

- Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. In *Proceedings of AAAI*, 2019.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.
- Ankur Bapna and Orhan Firat. Non-parametric adaptation for neural machine translation. In *Proceedings of NAACL*, 2019.
- Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
- Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of CoNLL*, 2016.
- Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.
- Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of ACL*, 2019.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. Search engine guided neural machine translation. In *Proceedings of AAAI*, 2018.
- Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450, 2018.
- Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. A retrieve-and-edit framework for predicting structured outputs. In *Proceedings of NeurIPS*, 2018.
- Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. Retrieval-based neural code generation. In *Proceedings EMNLP*, 2018.
- Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. Lagging inference networks and posterior collapse in variational autoencoders. In *Proceedings of ICLR*, 2019.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.
- Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.
- Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Nearest neighbor machine translation. *arXiv preprint arXiv:2010.00710*, 2020a.

- Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *Proceedings of ICLR*, 2020b.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Proceedings of NeurIPS*, 2016.
- Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. What do recurrent neural network grammars learn about syntax? In *Proceedings of EACL*, 2017.
- Bohan Li, Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick, and Yiming Yang. A surprisingly effective fix for deep latent variable modeling of text. In *Proceedings of EMNLP*, 2019.
- Chin-Yew Lin and Franz Josef Och. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of COLING*, 2004.
- Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of ECCV*, 2014.
- Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4: 521–535, 2016.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *Proceedings of ICLR*, 2018.
- Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.
- Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL (Demo Track)*, 2019.
- Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, 2014.
- Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of EMNLP*, 2019.
- Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of ACL (Demo Track)*, 2020.
- Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *Proceedings of EMNLP*, 2019.
- Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of EMNLP*, 2015.
- Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of NAACL*, 2015.

- Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of NeurIPS*, 2017.
- Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for nlp. *arXiv preprint arXiv:1908.07125*, 2019.
- Jason Weston, Emily Dinan, and Alexander Miller. Retrieve and refine: Improved sequence generation models for dialogue. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, 2018.
- Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierrick Cistac, Tim Rault, R’emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.
- Yu Wu, Furu Wei, Shaohan Huang, Yunli Wang, Zhoujun Li, and Ming Zhou. Response generation by context-aware prototype editing. In *Proceedings of AAAI*, 2019.
- Jiacheng Xu and Greg Durrett. Spherical latent spaces for stable variational autoencoders. In *Proceedings of EMNLP*, 2018.
- Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Proceedings of NeurIPS*, 2019.
- Yelp. Yelp dataset challenge, round 8. 2017. URL https://www.yelp.com/dataset_challenge.
- Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Learning to represent edits. In *Proceedings of ICLR*, 2019.
- Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. In *Advances in Neural Information Processing Systems*, pp. 9051–9062, 2019.