Initial Project Report

Mingyang Wu and Guangyu Lin

April 2023

0.1 Introduction

The data analysis task aims to predict masked words within a paragraph and reveal the hidden text using the Text8 dataset, which comprises words from the English Wiki. For instance, the task might involve predicting the actual word ”lesion” from its masked form ”_e__on”. This probabilistic modeling technique improves performance while maintaining practicality, offering state-of-the-art likelihoods in arbitrary conditional modeling for text, image, and continuous data domains.

0.2 Data and Analysis Plan

The Text8 dataset is a compilation of textual data extracted from English Wikipedia articles and is specifically tailored for natural language processing and language modeling tasks. With approximately 17 million words, the dataset has undergone preprocessing to retain only alphabetic characters and spaces, resulting in a clean and well-structured text corpus for research purposes. Although the dataset doesn't have explicit features, tokenization or word embedding techniques can be employed to derive the desired features. Each observation is a word represented as a character sequence, and there are no missing values since the dataset is preprocessed. The Text8 dataset was introduced by A. Radford, R. Child, D. Luan, D. Amodei, and I. Sutskever in their 2019 paper titled ”Language Models are Unsupervised Multitask Learners.”

We will use bits per dimension (bpd) to compare our model with baseline methods. Bpd is a valuable performance metric for the Text8 dataset as it quantifies the uncertainty of a language model in predicting the next token. Lower bpd values indicate better compression and more accurate predictions, making it an effective measure for comparing different models. Bpd is useful for assessing the quality of language models on any text corpus, including Text8.

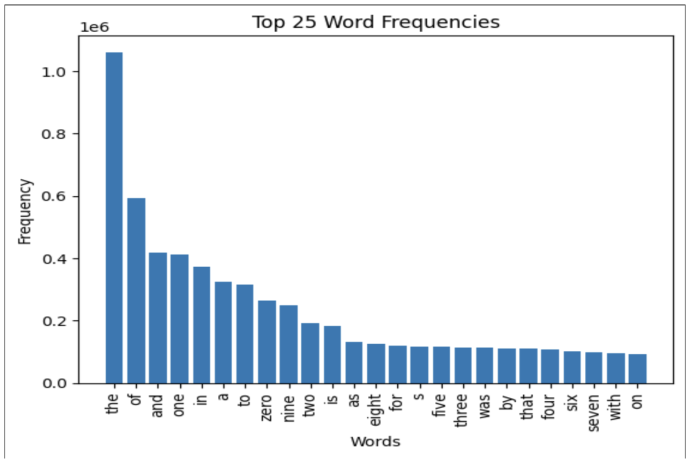


Figure 1: Top 25 word frequencies

As shown in Figure 1, the prevalence of common English words and numerals in word form suggests that some patterns or redundancy can be exploited by the model to achieve better compression and encoding performance. It is important to note that our models aiming for a lower Bpd value will likely benefit from capturing these commonalities and exploiting the dataset's inherent structure. This understanding can guide us in selecting and optimizing models that take advantage of the redundancy found in the dataset to minimize the number of bits needed to represent each dimension effectively. For this task, we have decided to allocate 90% of the dataset for training, 5% for validation, and 5% for testing. This split is sensible since the dataset is large, and 5% is sufficient for both validation and testing. Using a substantial portion of the dataset for training allows the model to learn diverse language patterns, improving generalization capabilities. A separate validation set is vital for model selection, hyperparameter tuning, and monitoring overfitting during the training process. Finally, a test set is essential for evaluating the model's performance on unseen data, giving an estimate of how well the model will perform in real-world scenarios. In summary, this train/validation/test split strategy is well-suited for language modeling tasks using the Text8 dataset, as it ensures sufficient data for learning complex patterns while providing separate sets for model tuning and unbiased performance evaluation.

0.3 Baseline Method

For the Text8 dataset, we are interested in creating a language model using an any-order autoregressive model (AO-ARM). Let's represent the dataset as a sequence of tokens $X = (x_1, x_2, \ldots, x_N)$, where N is the total number of tokens in the dataset, and x_i represents the i-th token.

By modeling $\prod_{t=1}^{N} p(x_{\sigma(t)}|x_{\sigma(<t)};\sigma)$ for all orderings σ, we can choose an order σ that is compatible with e and we can know $\sigma(\leq |e|) = e$, where $|e|$ is the cardinality of e. Then, we can evaluate each of the univariate conditionals as:

$$p(x_e) = \prod_{t=1}^{|e|} p(x_{\sigma(t)}|x_{\sigma(<t)};\sigma) \tag{1}$$

Our goal is to find the optimal parameters θ that maximize the likelihood of the observed data. We can train the AO-ARM to maximize the joint likelihood of a datapoint x under the expectation over the uniform distribution $\mathcal{U}_\sigma$ of orders, and by Jensen inequality we mentioned in $hw4$, we can get:

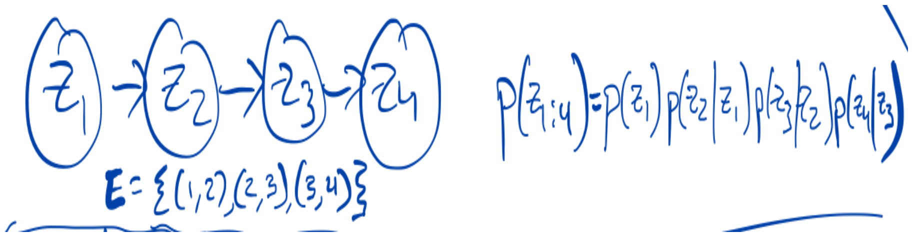
$$\text{log}p(x) = \text{log}\mathbb{E}_{\sigma\sim\mathcal{U}_\sigma} \sum_{t=1}^{N} p(x_{\sigma(t)}|x_{\sigma(<t)};\sigma) \geq \mathbb{E}_{\sigma\sim\mathcal{U}_\sigma} \sum_{t=1}^{N} \text{log}p(x_{\sigma(t)}|x_{\sigma(<t)};\sigma) \tag{2}$$

This objective can be simplified by treating t as a random variable with a uniform distribution $\mathcal{U}_t$ over cardinalities 1 to N. We let $M_{\text{card-edge}}$ be the distribution over the tuple $(\sigma(t), \sigma(<t))$ where $\sigma \sim \mathcal{U}_\sigma$ and $t \sim \mathcal{U}_t$, giving the following loss function for an AO-ARM parameterized by θ.

$$\mathcal{L}(\theta) = -\mathbb{E}_{\sigma\sim\mathcal{U}_\sigma} \sum_{t=1}^{N} \text{log}p(x_{\sigma(t)}|x_{\sigma(<t)};\sigma) = -N \cdot \mathbb{E}_{i,e\sim M_{\text{card-edge}}} \text{log}p_\theta(x_i|x_e) \tag{3}$$

By optimizing the log-likelihood, we aim to use the gradient descent to find the minimum loss value from the loss function.

Computing arbitrary marginals and conditionals has been a long-standing challenge in probabilistic inference, with a rich history of techniques, such as MCMC which we have discussed during class. Unlike the Markov model, through AO-ARM, we do not need to know the current state.



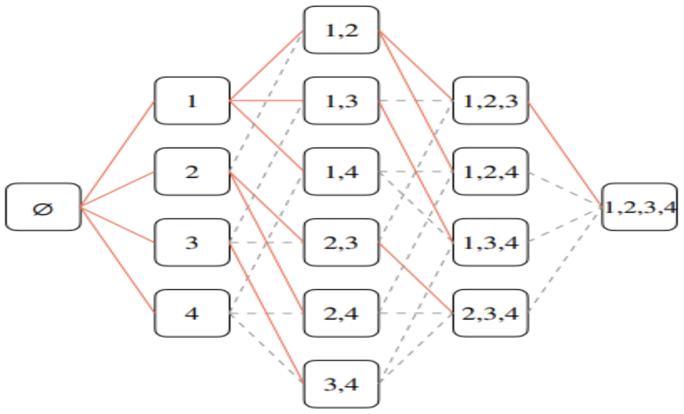
z_1
z_2
z_3
z_4
$E = \{(1,2),(2,3),(3,4)\}$
$p(z_{1:4}) = p(z_1)p(z_2|z_1)p(z_3|z_2)p(z_4|z_3)$

Instead, we can just use the conditional probability by some ordering σ and find the joint probability by chain rule, which we believe is more accurate and rigorous when we are doing calculations.

$$p(x) = \prod_{t=1}^{N} p(x_{\sigma(t)}|x_{\sigma(<t)};\sigma) \tag{4}$$

0.4 Proposed Upgrade

We will change the model in our upgrade. We don't need to learn all univariate conditionals (edges) to compute arbitrary marginals (nodes). It suffices to learn one left-going edge for every node besides the $\emptyset$ node, just like what red line in the figure is doing.



In this case, it will reduce edge redundancy from the AO-ARM, which we believe is a more effective way to do the calculation.