You can't pick your neighbors, or can you? When and how to rely on retrieval in the kNN-LM

Anonymous ACL submission

Abstract

Retrieval-augmented language models (LMs), which condition their predictions on text retrieved from large external datastores, have recently shown significant perplexity improvements compared to standard LMs. One such approach, the kNN-LM, interpolates any existing LM's predictions with the output of a k-nearest neighbors model and requires no additional training. In this paper, we explore the importance of lexical matching and vector similarity in the context of items retrieved by kNN-LM. We find two trends: (1) the presence of large overlapping n-grams between the datastore and evaluation set plays an important factor in strong performance, even when the datastore is derived from the training data; and (2) kNN-LM is more effective on average when the top retrieved items have high vector similarity with the query. Based on our analysis, we define a new formulation of the kNN-LM that uses an adaptive interpolation coefficient rather than a static value for all queries. We measure empirically the effectiveness of our approach on two English language modeling datasets, Wikitext-103 and PG-19. Our re-formulation of the kNN-LM is beneficial in both cases, and leads to nearly 4% improvement in perplexity on the Wikitext-103 test set.

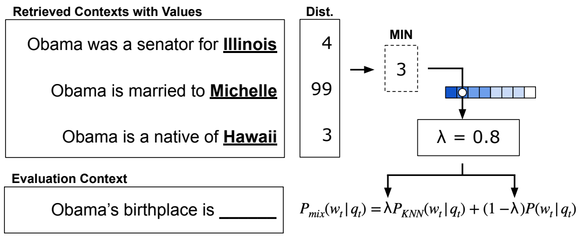

Figure 1: We present an extension to kNN-LM that conditions the interpolation coefficient (λ) on the vector distance of retrieved contexts.

1 Introduction

Recently, a new class of language models (LMs) that are augmented with *retrieval* capabilities have led to substantial improvements over standard neural LMs (Lewis et al., 2020; He et al., 2020; Yogatama et al., 2021; Borgeaud et al., 2021; Wu et al., 2022; Thoppilan et al., 2022, inter alia). Furthermore, LMs with retrieval warrant investigation as they provide benefits for many tasks (Zamani et al., 2022). These approaches generally involve a backbone neural LM that interacts with a retrieval component of varying complexity to find relevant documents. In this work, we analyze and improve a specific and simple type of retrieval-augmented language model, the kNN-LM originally proposed by Khandelwal et al. (2020).

The kNN-LM is non-parametric — it works by retrieving instances from an external datastore at each decoding timestep, and it improves language model performance without requiring additional training. In essence, the kNN-LM interpolates a base LM's predicted probability distribution of the next word with a distribution formed by *retrieving* vectors similar to the current hidden state. kNN-LM includes two tunable hyperparameters: the number of items to retrieve (k) and an interpolation coefficient (λ). The method's effectiveness depends crucially on source and size of the retrieval datastore: it is most effective when using a very large datastore with orders of magnitude more tokens than seen in the training corpus, but Khandelwal et al. (2020) also observe improvements with smaller datastores.

Modern neural models have massive capacity to memorize their training data (Zhang et al., 2017). Nonetheless, simply using an LM's training corpus as the source for the datastore works well for kNN-LM, as test perplexity on the Wikitext-103 dataset decreases substantially from 18.65 to 16.12. However, it remains unclear how and why the kNN-LM achieves these improvements. Which types of tokens and contexts does it improve most on? In §3, we analyze the kNN-LM's behavior with respect to parts of speech, vector distance between context and retrievals, and lexical overlap.

Our analysis reveals the syntactic categories kNN-LM is most helpful with (e.g. proper nouns), that low vector distance between query and context vectors correlates with kNN-LM improvements over the base language model, and that the kNN-LM is sensitive to lexical patterns (albeit the degree of sensitivity is domain specific). Next, we leverage our knowledge of when the kNN-LM helps to develop an improved reformulation. In particular, we develop a simple yet effective modification of the kNN-LM that employs an *adaptive* interpolation coefficient instead of a static one as in the original method. Our method chooses this coefficient (λ) conditioned on the query and its retrieved items (see Figure 1). While it introduces new hyperparameters, we show that the additional hyperparameter tuning comes at negligible cost. Importantly, our empirical results demonstrate that our newly introduced reformulation of kNN-LM is beneficial for both encylopedic text and book data, and leads to an improvement of nearly 4% perplexity over the the vanilla kNN-LM, measured on the English language modeling Wikitext-103 test set. Broadly, we hope that our insights and methods can help facilitate future development of retrieval-augmented LMs.

2 Language Modeling with kNN-LM

The kNN-LM improves over a base language model by explicitly *memorizing* the LM's training data. It stores exact sentences from the training data in its datastore that can be accessed during language model inference to produce a k-nearest neighbor next word distribution that is interpolated with the base model's prediction. Interpolation is preferred for similar reasons as approximate matrix factorization in collaborative filtering — the universe of text patterns is sparse and lossless compression of the training data alone is not sufficient to model new patterns. In this section, we explain the specifics of the kNN-LM's inner workings in order to guide our analysis.

2.1 General Approach

The kNN-LM (Khandelwal et al., 2020) is a language model with a retrieval component. Like all language models, it predicts the the word at time step t conditioned on the history of words: $P(w_t|w_0, w_1, \ldots, w_{t-1})$. Neural language models encode the history of words using a vector h: $P(w_t|h_{t-1})$. What makes the kNN-LM novel is that it uses a pre-trained language model to encode a collection of documents, and then retrieves documents from this collection based on vector similarity in order to improve its next word prediction. Notably, the retrieval is completely latent — no supervised ranking information is used and documents are simply retrieved using vector similarity.

The kNN-LM follows a particular way of encoding the collection of documents into a datastore. Consider document x_i consisting of n words. The kNN-LM encodes the first $n-1$ words as a vector and this becomes the **key** of document x_i, referred to as k_i. The n-th word is saved as the **value** v_i. In practice, and since kNN-LM is used for language modeling, a sequence with n words is recorded as $n-1$ documents: for any $t \leq n$, a document whose key is words w_1 to w_{t-1} and value is w_t is built.

After the datastore is built, the kNN-LM is evaluated on a dataset with m words, predicting words from left-to-right. Retrieval in kNN-LM is done by measuring Euclidean distance $d(.,.)$ between vector encodings of the **query** q_j (corresponding to the context of the j-th word in the evaluation data) and the keys in the datastore. The values from retrieved documents define a new distribution of the next word:

$$P_{KNN}(w_t|q_t) \propto \sum_{(k_i,v_i)} \mathbb{1}_{w_t=v_i} \exp(-d(k_i, q_t)) \tag{1}$$

The best performance typically involves mixing the original and kNN-based word distributions using a tunable hyperparameter λ:

$$P_{mix}(w_t|q_t) = \lambda P_{KNN}(w_t|q_t) + (1-\lambda)P(w_t|q_t) \tag{2}$$

3 Analysis: When is kNN-LM effective?

In the original kNN-LM work, the authors point out that the model generally helps for rare patterns, factual knowledge, and names (Khandelwal et al., 2020). These observations are primarily qualitative and anecdotal. In this section we perform automated analysis to more specifically understand when kNN-LM is beneficial, with the aim to uncover systematic behavior that can be leveraged to extend kNN-LM and improve its effectiveness at next word prediction.

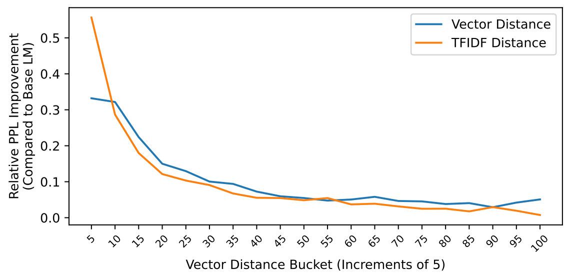

Figure 2: Relative perplexity improvement of kNN-LM compared to the base language model measured on the Wikitext-103 validation set. Query contexts are bucketed by vector distance of the closest retrieved item.

	Dev	Dev-8	Test	Test-8
Wikitext				
BaseLM	17.96	17.96	18.65	18.65
kNN-LM	16.06	17.28	16.12	18.05
Ours	15.72	17.26	15.50	18.03
PG-19				
BaseLM	60.83	60.83	50.95	50.95
kNN-LM	52.49	53.34	43.93	44.97
Ours	52.08	53.06	43.58	44.78

Table 1: Perplexity on Wikitext-103 and PG-19 datasets. Dev-8 and Test-8 contain the same data as Dev and Test, but overlapping n-grams ($n \geq 8$) with the evaluation data have been removed from the kNN-LM datastore. The adaptive coefficient (Ours) is described in §4.

3.1 Distance between Latent Vectors

The kNN-LM encodes the context into a fixed-length query vector and uses this to retrieve similar contexts from the datastore. Not every retrieved context will have a key whose associated value matches the ground-truth next word, and a priori, it is difficult to know when a retrieved context is helpful. Nonetheless, kNN-LM is typically helpful when at least 1 of the retrieved keys corresponds to the true next word.

Figure 2 examines this intuition a posteriori on the Wikitext-103 validation set. First, we bucket query contexts according to the vector distance of their top retrieved item. The buckets are non-overlapping and each contain 5% of the query contexts — the first contains the 5% of queries that have lowest vector distance with their top retrieved item, the next contains the next 5% lowest, and so on. Plotted in the figure is the relative perplexity improvement of kNN-LM compared to the base language model. It becomes obvious that the buckets with lower vector distance are the ones where kNN-LM is more beneficial, supporting the hypothesis that vector distance is a proxy for relevance.

3.2 Lexical Overlap

Another possible proxy for relevance is lexical overlap. Rather than directly using neural network hidden representations to form buckets, we first convert contexts into TFIDF vectors (using 32-token trailing window), which are a popular and effective bag-of-words representation (Chen et al., 2017). We use these representations to measure vector distance solely for bucketing (retrieval is still done using the kNN-LM's neural representations). The relative perplexity for this setting is reported in Figure 2, and aligns well with the bucketing reported in the previous subsection. This suggests that kNN-LM is beneficial when the current context has high lexical overlap with a context saved in the datastore.

To further examine the relationship between kNN-LM performance and lexical overlap between contexts, we rebuild the index that kNN-LM uses to retrieve items in a way to minimize lexical overlap. We run the base language model over the training corpus, but ignore tokens that correspond to large overlapping n-grams ($n \geq 8$) with the evaluation data.[1] The Wikitext-103 perplexity for kNN-LM compared to the base language model using the original and restricted datastore is shown in Table 1. Trend-wise, the kNN-LM works better when there is high lexical overlap between query and its retrieved contexts, but when large overlapping n-grams are removed from the datastore, the benefit of the kNN-LM substantially diminishes.

3.3 Part-of-Speech Tags

Another lens, syntax, can shed light on kNN-LM performance outside of document relevance. To further understand which types of words benefit most from kNN-LM, we group tokens by their part-of-speech. Then we compute validation perplexity separately for each group using both the base language model and the kNN-LM. To get part-of-speech tags, we segment the data into sentences and label words using the tagger from Stanza[2] with the universal dependencies output space. We in-

[1]To ensure the n-gram context does not leak into the datastore, we follow Brown et al. (2020) and ignore tokens corresponding to a 200-token window centered around the n-gram.

[2]https://stanfordnlp.github.io/stanza/

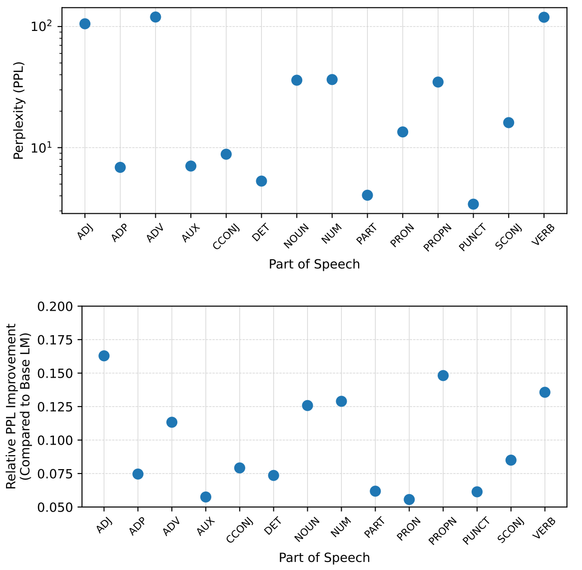

Figure 3: Perplexity of the base language model grouped by part-of-speech (top), and relative improvement of the kNN-LM (bottom).

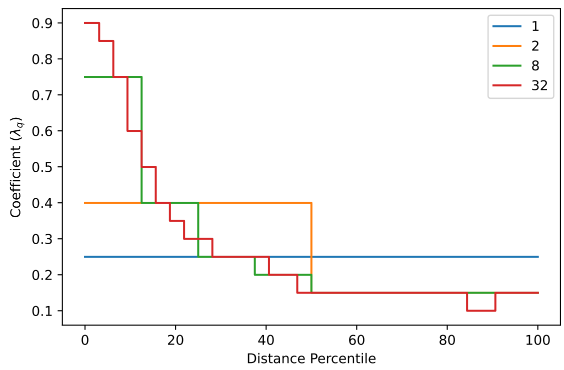

Figure 4: Sets of coefficients (λ_q) tuned on the Wikitext validation set for different bucket sizes (1, 2, 8, 32). The buckets are balanced.

clude categories with frequency greater than 1K in the Wikitext-103 validation data.

The results are included in Figure 3. We find that kNN-LM is most helpful for syntactic categories where the base language model most struggles, e.g. the original perplexity for adjectives (ADJ) is 105.37 and the kNN-LM improves perplexity by 16.3% for this category. The five other categories that had worst perplexity (ADV, NOUN, NUM, PROPN, VERB) are also where kNN-LM works best. It's satisfying to see proper noun (PROPN) included here, as it reflects intuition of kNN-LM assisting with factual knowledge.

4 A New Formulation for kNN-LM

In the previous section, we analysed when kNN-LM is most helpful. We use this information to design a new formulation of kNN-LM that can exploit this behavior. The original kNN-LM uses the same interpolation coefficient (λ) for every example, which may not be desirable. As our analysis reveals, we can predict when the kNN-LM is most beneficial, which naturally leads us to a new formulation with an *adaptive* λ:

$$P'_{mix}(w_t|.) = \lambda_q P_{KNN}(w_t|.) + (1-\lambda_q)P(w_t|.) \tag{3}$$

where λ_q is a function of the query and its retrieved documents rather than constant for all queries.

Using the same λ for all examples is limiting — it does not effectively leverage retrieval when neighboring keys are clearly relevant (like shown in Figure 1), nor does it trust the base language model when retrieved items are dissimilar from the query context. Of course, the critical decision here is how to define partitions. One option that we explore is the "Distance Percentile" (measured with "minimum vector distance") covered in the previous section. If we partition the data into two balanced buckets, we would define the first bucket as queries with "minimum vector distance" within the [0, 50] percentile range and the second bucket including those in the (50, 100] range. For each bucket we perform the same hyperparameter search over coefficients as in kNN-LM.[3] Sets of coefficients using different bucket sizes and tuned for Wikitext-103 are shown in Figure 4.

5 Experiments and Results

To evaluate the effectiveness of our new formulation of kNN-LM we measure perplexity on two English language modeling datasets. The first is the Wikitext-103 corpus (Merity et al., 2016) used by Khandelwal et al. (2020). The second is PG-19 (Rae et al., 2020), which we include because it consists of books and is thematically distinct from the encyclopedic documents in Wikitext-103.

The target baseline we aim to improve on is the original formulation of kNN-LM. As described in §2.1, the datastore is built by encoding a large text corpus, in this case the training set. The kNN-LM is already a substantial improvement over the backbone language model (Baevski and Auli, 2019).

[3]See Khandelwal et al. 2020 Figure 5.

b	Dev_0	Dev_1	Dev
1	17.091	14.989	16.091
2	16.909	14.854	15.933
4	16.763	14.767	15.815
8	16.665	14.727	15.743
16	16.637	14.722	15.727
32	16.629	14.722	15.721
64	16.622	14.724	15.719
128	16.619	14.724	15.715

Table 2: Validation perplexity on Wikitext-103. Used for hyperparameter tuning.

By using an adaptive interpolation coefficient we improve the performance even further.

5.1 Experimental Setup and Pre-processing

Wikitext-103 The data is split 103M/217K/245K tokens for training, validation, and test. The vocabulary is at the word-level and includes 267K tokens. We use the already trained model from Khandelwal et al. (2020).

PG-19 To understand when the adaptive coefficient is desirable compared with a static coefficient, we include PG-19 in our experiments. PG-19 consists of books and is thematically distinct form the encyclopedic douments in the Wikitext-103 data. We sample 2,000 books from the training corpus, which gives approximately 150M tokens and is close in size to Wikitext-103. We use the standard validation split (50 books) and test split (100 books). We use word-level tokenization with a 300K vocabulary derived from our constructed training split. We train our own model using the same architecture and hyperparameters from Khandelwal et al. (2020).

5.2 Tuning kNN-LM Hyperparameters

For the original formulation of kNN-LM there are two hyperparameters to tune: the number of items to retrieve (k) and the interpolation coefficient (λ). These are tuned on the validation set. We introduce an important hyperparameter for the number of buckets to use (b)[4] and tune a new interpolation coefficient (λ_q) separately for each bucket. Since each bucket is assigned its own coefficient, the total number of hyperparameters grows with the number of buckets. Even so, hyperparameter tuning is only required for the validation data and we cache expensive computation so that the cost of hyperparameter search is negligible (see §5.2.1 for more details).

To select the number of buckets (b), we use the first half of the validation data (Dev_0) to define partition boundaries, and find the best performing interpolation coefficient for each partition separately. Then we measure perplexity on the second half of the validation data (Dev_1) using those partition boundaries and coefficients. The choice of b that gives the best perplexity on Dev_1 is the one we ultimately use. With b chosen, we then re-compute the partition boundaries and corresponding coefficients using the full validation data (Dev), which will be used to evaluate against the test data.

An example of tuning for b on Wikitext-103 is shown in Table 2. On Dev_0, increasing b always leads to better perplexity, albeit with diminishing returns. On the held-out data (Dev_1), since the partition boundaries and coefficients are chosen using Dev_0, it is not guaranteed that increasing b improves perplexity. Similarly, tuning the partition boundaries and coefficients on the validation data does not guarantee improvement on the test data. Even so, empirically we find that our adaptive coefficient is always at least as effective as the static coefficient.

5.2.1 Computational Cost of Tuning

Using the adaptive coefficient, the number of hyperparameters scales with the size of b and is more than order of magnitude more than what is used for kNN-LM. That said, by effectively caching query vectors, retrieved items, and associated vector distances the cost associated with hyperparameter search is negligible. The initial time to compute these values takes hours and is the same as with kNN-LM, but after computed it takes less than 5 minutes to perform the hyperparameter search for the adaptive coefficient on the Wikitext-103 data.[5] We release our implementation with value caching on github: github.com/anonymous/submit.

5.3 Perplexity on WikiText-103

Table 3 reports the perplexity from our approach and various baselines on the Wikitext-103 validation and test sets. Our approach scores 15.50 per-

[4]The number of buckets divides the data evenly, and each is associated with partition boundaries. The queries with a minimum vector distance greater than the lower boundary and less than the upper boundary are included in the bucket.

[5]All experiments are run on a single Titan X GPU with 256GB CPU memory.

	λ	b	k	Dev	Test
Base LM	-	-	-	17.96	18.65
kNN-LM	0.25	1	1024	16.06	16.12
+CCache	0.25	1	1024	15.81	15.79
Ours (TFIDF)	λ_q	32	1024	15.76	15.54
Ours	λ_q	32	1024	15.72	**15.50**

Table 3: Test and validation perplexity on Wikitext-103. This is our main result and demonstrates that our new formulation with adaptive coefficient (λ_q) substantially improves over kNN-LM.

plexity on the test set. This is a 16.9% improvement over the base language model and a 3.8% improvement over the original kNN-LM formulation.

For the number of buckets (b) we found 32 to work best (see Table 2), and the set of coefficients are the same as shown in Figure 4. Our search space includes $b \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ and $\lambda_q \in \{0.05, 0.1, 0.15, \ldots, 0.9, 0.95\}$.

Khandelwal et al. (2020) find that retrieving from recent history using the continuous cache model (CCache; Grave et al. 2017) is complementary to retrieving from the datastore, improving perplexity when combined with kNN-LM. This type of caching is out of scope of this paper, and our approach already outperforms the combined model.

5.4 Perplexity on PG-19

To further understand how lexical overlap influences kNN-LM performance we evaluate using the PG-19 dataset. Compared to Wikipedia, text across books has much less repetition, so text retrieved from the datastore is less likely to overlap with n-grams in the evaluation data.

We train our own model using the same architecture and hyperparams for Wikitext-103, and report perplexity in Table 1. We found $b = 32$ works best. Despite the challenging properties of the book data, kNN-LM is still effective. Our re-formulation is marginally beneficial here.

5.5 Filtering n-grams from the Datastore

Our analysis thus far indicates that lexical overlap is important for strong kNN-LM performance. To test this directly for the adaptive coefficient, we follow the procedure described in §3.2 rebuild the datastore but remove from the index large n-grams ($n \geq 8$) and their surrounding tokens that also appear in the evaluation data.

	λ	b	k	Dev
Dense	0.25	1	1024	16.06
Dense	λ_q	32	1024	15.72
TFIDF	λ_q	32	1024	15.76
Dense	0.05	1	1	17.10
Dense	0.15	1	8	16.66
Dense	0.25	1	64	16.31
Dense	λ_q	16	1	16.63
Dense	λ_q	128	8	16.19
Dense	λ_q	16	64	15.90
TFIDF	λ_q	32	1	16.38
TFIDF	λ_q	64	8	16.06
TFIDF	λ_q	16	64	15.87

Table 4: Validation perplexity on Wikitext-103 used for ablation analysis. The kNN-LM uses a single static value for the interpolation coefficient (λ), our method uses an adaptive coefficient (λ_q). This table includes our approach when using the learned vector distance (Dense) or bag-of-words representation (TFIDF). Based on how many items are retrieved (k), our approach works best with a different amount of buckets (b).

The results for this experiment on both Wikitext-103 and PG-19 are shown in Table 1. Most of kNN-LM's improvements on Wikitext-103 come from retrieving contexts with overlapping n-grams,[6] which could motivate simpler and faster retrieval functions. On the other hand, the cases in which n-gram overlap does not play a major role require further analysis.

6 Behavior of Adaptive Coefficient

In §3, we inspect cases to understand when kNN-LM is most effective compared to the base language model. In this section, we instead analyze the behavior of our new formulation that uses an adaptive coefficient, with the goal of understanding its relative performance compared to the original kNN-LM.

6.1 How to define partition boundaries?

In §3.1 we establish that kNN-LM performs similarly per bucket when partition boundaries are established through latent vectors or bag-of-words

[6] As others have previously noted, Wikitext-103 contains considerable amounts of duplicate text (McCoy et al., 2021). Deduplicating the training data can be helpful for language modeling (Lee et al., 2022; Kandpal et al., 2022), and sometimes other tasks (Schofield et al., 2017), but we completely remove text that overlaps with the evaluation data.

representation. The question is, does this also hold when using an adaptive coefficient? We tune the adaptive coefficient using different values of k and report the results for Wikitext-103 in Table 4.

In general, we find that both the dense learned vectors and TFIDF bag-of-word vectors work similarly well for establishing partition boundaries. For the best setting, when $k = 1024$, the learned vectors work better, reflecting recent findings that dense vectors outperform sparse representations for various retrieval-related tasks (Lee et al., 2019). Hence, throughout this paper we use the adaptive coefficient with learned vectors and $k = 1024$ unless otherwise specified. Interestingly, for lower values of k the bag-of-words representation has an edge over the learned vectors. If there is a budget on how many items can be retrieved, then it can be especially helpful to use our adaptive coefficient with partition boundaries from TFIDF rather than the original kNN-LM.

6.2 Does the adaptive coefficient bolster the trends from kNN-LM?

We repeat the syntactic analysis from §3.3 using our adaptive coefficient and include PG-19 as an additional dataset.[7] The corresponding plots are shown in Figure 5.

For the base language model, the relative perplexity in each syntactic bucket is similar between Wikitext-103 and PG-19. The relative perplexity of kNN-LM is mostly similar in each dataset, except for two important categories, adjective (ADJ) and verb (VERB), which kNN-LM helps with more for Wikitext-103 than PG-19.

For Wikitext-103, our adaptive coefficient is universally helpful across all syntactic categories. This behavior changes for PG-19. The adaptive coefficient is more helpful than static coefficient for most categories, but has negligible impact on ADP, PRON, and PUNCT. This is surprising as PRON is a category where kNN-LM gives an outsized improvement on Wikitext-103. Although there are categories where the adaptive coefficient hurts (CCONJ and DET), this is outweighed by the improvement on ADJ and VERB. As previously mentioned, ADJ and VERB are more challenging for kNN-LM on PG-19 than Wikitext-103, so the benefit provided by the adaptive coefficient is appealing.

[7]We only include the first 500K tokens from PG-19 validation data, as this is already more than twice the size of Wikitext-103 validation data.

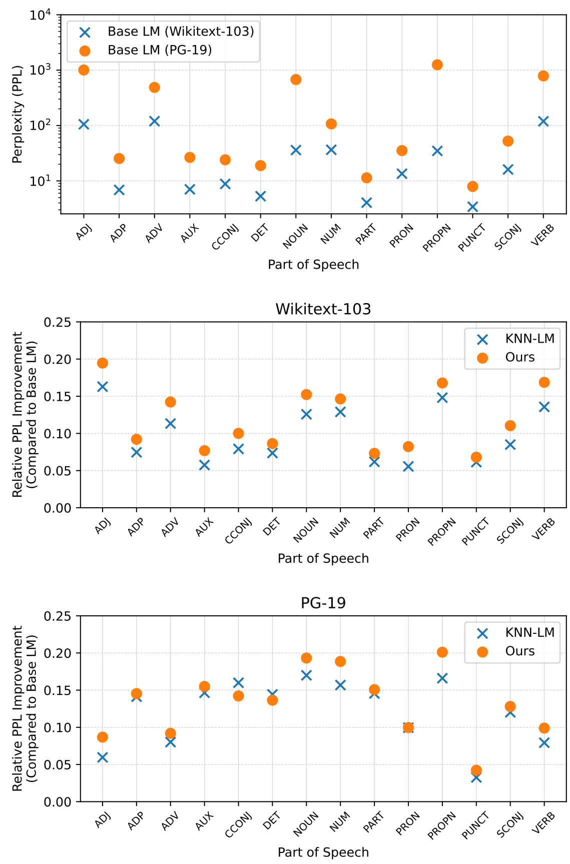

Figure 5: Perplexity of the base language model (top), grouped by part-of-speech. Relative perplexity improvement by kNN-LM approches on Wikitext-103 (center) and PG-19 (bottom). The lines corresponding kNN-LM match Figure 3 — they are included here to emphasize the difference to our new formulation.

6.3 Is the adaptive coefficient helpful without highly relevant items in the datastore?

As we established in §3.2, the lexical overlap between a query and a retrieved context is a reasonable proxy for relevance. In Table 1, we report the perplexity of our adaptive coefficient when removing large n-grams from the datastore that overlap with the evaluation data. With these highly *relevant* contexts removed, we observe that the kNN-LM shows substantially worse test perplexity on Wikitext-103, 18.05 instead of 16.12. On the contrary, for PG-19 the change in perplexity is minimal. This suggests that kNN-LM can be helpful even when there are not large overlapping n-grams between the datastore and evaluation corpus — such cases occur frequently in PG-19, and we visualize these in Table 5.

The benefit from the adaptive coefficient is also substantially diminished for Wikitext-103, but less

Book	Context	r
The Unbearable Bassington, Saki (1912)	My dear Francesca , he said soothingly , laying his hand **affectionately**	q
FLORA, A.L.O.E. (1860)	My dear madam , said Mr. Ward earnestly , laying his hand *on*	1
Peter, Smith (1908)	this young man 's uncle , said Peter , laying his hand **affectionately**	11
Life of Napoleon Bonaparte, Sloane (1896)	during the worst periods of terror , were thronged from pit to **gallery**	q
Sketches of Reforms—, Stanton (1849)	For weeks , that theater was crowded from pit to *dome*	1
Farquharson of Glune, Bateman (1908)	The storm of feeling swept alike from stall to **gallery**	6
Walking, Thoreau (1851)	like a dream of the Middle Ages . I floated down its historic **stream**	q
The Automobilist Abroad, Mansfield (1907)	France is a pleasure , a voyage up a picturesque and historic *French*	1
Canadian Notabilities, Dent (1880)	two small sailing craft slowly making their way up the majestic **stream**	42

Table 5: Examples from PG-19 where relevant contexts are found even with large n-grams removed from the datastore. There can be overlap in small n-grams (top), local structure (center), or semantics (bottom). The contexts are shown with their corresponding book. Rank (r) is shown except for queries (q). Values are bolded or italicized.

so for PG-19. This suggests that the partitions capture similarity akin to lexical overlap for Wikitext-103, but perhaps encode other properties, such as semantic similarity, for PG-19. Alternatively, it could be that short n-grams are helpful in Wikitext-103, despite Khandelwal et al. (2020) reporting that interpolating the base language model with an n-gram model was not very effective.

It is worth noting that even with the relevant items removed from the datastore, the adaptive coefficient is robust and provides just as good performance as the original kNN-LM. While kNN-LM does not provide as much of a benefit, it still improves over the base language model. These findings suggest alternative definitions of partition boundaries may be worth exploring besides those derived from learned vector distance or TFIDF.

7 Related Work

Our adaptive coefficient improves perplexity on the language modeling task without additional training. In contrast, several works extend kNN-LM with a tradeoff between task performance and speed. These techniques include conditioning retrieval on previously retrieved documents (Alon et al., 2022), dimensionality reduction of the datastore (He et al., 2021), or retrieving phrases instead of single tokens (Martins et al., 2022). In similar spirit, Zheng et al. (2021) train a separate component to predict k based on the input and retrieved items — in our case, we did not find any benefit to tuning k separately for each bucket. Other extensions inject document categories (Xu et al., 2022) or add retrieval-specific model layers (Meng et al., 2022), and are not directly comparable with our results.

Researchers have explored fundamental extensions to kNN that are agnostic to language data. Wettschereck and Dietterich (1993) spatially partition the datastore, adapting the value of k for each region. Keeping k fixed, Hastie and Tibshirani (1995) instead adapt the shape of the neighborhood based on local information.

There are many recent works that use retrieval components for language tasks besides next word prediction, such as question answering (Godbole et al., 2019; Guu et al., 2020; Kassner and Schütze, 2020), dialogue generation (Fan et al., 2021), conversational search (Hashemi et al., 2020), semantic parsing (Gupta et al., 2021), and data augmentation (Du et al., 2021). The kNN-MT used for machine translation (Khandelwal et al., 2021) is an extension of kNN-LM for conditional text generation — it adds a new hyperparameter T to control softmax temperature in the interpolated probability.

8 Conclusion

In this paper, we have proposed a novel and effective re-formulation of the kNN-LM. Our approach uses an adaptive interpolation coefficient conditioned on the query and retrieved documents — queries that retrieve at least one highly similar item assign a higher weight to the kNN probability. Although this adds many new hyperparameters to tune, the additional computational cost is negligible. Our analysis supports the intuition behind the adaptive coefficient and provides insights on which types of tokens kNN-LM is most helpful for. Importantly, our experimental results verify the effectiveness of our new method, which provides nearly 4% improvement in test perplexity on the Wikitext-103 language modeling corpus.

Limitations

The kNN-LM leverages a datastore, and when populated with text relevant for the task domain, can be used to improve language modeling performance. The benefits of this procedure are data dependent and domain-specific, and the same applies to the adaptive coefficient technique that we introduce.

The adaptive coefficient requires many more tunable hyperparameters. To address this, we release an optimized codebase to perform this hyperparameter search in neglible time compared with the original kNN-LM.

Ethical Concerns and Impact

Even when used with the best intentions language models can produce malicious or harmful text, and guards are typically used to account for inherent bias or undesirable output. In our case, we do not generate text and simply use the model to evaluate perplexity on existing data, so effectiveness of safety guards and their limitations is not a relevant concern in this work.

References

Uri Alon, Frank F. Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. 2022. Neuro-symbolic language modeling with automaton-augmented retrieval. *ArXiv*, abs/2201.12431.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, T. W. Hennigan, Saffron Huang, Lorenzo Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and L. Sifre. 2021. Improving language models by retrieving from trillions of tokens. In *ICML*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

Jingfei Du, Edouard Grave, Beliz Gunel, Vishrav Chaudhary, Onur Çelebi, Michael Auli, Ves Stoyanov, and Alexis Conneau. 2021. Self-training improves pre-training for natural language understanding. In *NAACL*.

Angela Fan, Claire Gardent, Chloé Braud, and Antoine Bordes. 2021. Augmenting transformers with knn-based composite memory for dialog. *Transactions of the Association for Computational Linguistics*, 9:82–99.

Ameya Godbole, Dilip Chakravarthy Kavarthapu, Rajarshi Das, Zhiyu Gong, Abhishek Singhal, Hamed Zamani, Mo Yu, Tian Gao, Xiaoxiao Guo, Manzil Zaheer, and Andrew McCallum. 2019. Multi-step entity-centric information retrieval for multi-hop question answering. *ArXiv*, abs/1909.07598.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. In *International Conference on Learning Representations*.

Vivek Gupta, Akshat Shrivastava, Adithya Sagar, Armen Aghajanyan, and Denis Savenkov. 2021. Retronlu: Retrieval augmented task-oriented semantic parsing. *ArXiv*, abs/2109.10410.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *ArXiv*, abs/2002.08909.

Helia Hashemi, Hamed Zamani, and W. Bruce Croft. 2020. Guided transformer: Leveraging multiple external sources for representation learning in conversational search. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Trevor Hastie and Robert Tibshirani. 1995. Discriminant adaptive nearest neighbor classification and regression. In *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

Junxian He, Taylor Berg-Kirkpatrick, and Graham Neubig. 2020. Learning sparse prototypes for text generation. In *NeurIPS*.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient nearest neighbor language models. In *EMNLP*.

Nikhil Kandpal, Eric Wallace, and Colin Raffel. 2022. Deduplicating training data mitigates privacy risks in language models. *ArXiv*, abs/2202.06539.

Nora Kassner and Hinrich Schütze. 2020. BERT-kNN: Adding a kNN search component to pretrained language models for better QA. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3424–3430, Online. Association for Computational Linguistics.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *International Conference on Learning Representations*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations (ICLR)*.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*.

Pedro Henrique Martins, Zita Marinho, and André F. T. Martins. 2022. Chunk-based nearest neighbor machine translation. *ArXiv*, abs/2205.12230.

R. Thomas McCoy, Paul Smolensky, Tal Linzen, Jianfeng Gao, and Asli Celikyilmaz. 2021. How much do language models copy from their training data? evaluating linguistic novelty in text generation using raven. *ArXiv*, abs/2111.09509.

Yuxian Meng, Shi Zong, Xiaoya Li, Xiaofei Sun, Tianwei Zhang, Fei Wu, and Jiwei Li. 2022. GNN-LM: Language modeling based on global contexts via GNN. In *International Conference on Learning Representations*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

Alexandra Schofield, Laure Thompson, and David Mimno. 2017. Quantifying the effects of text duplication on semantic models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2737–2747, Copenhagen, Denmark. Association for Computational Linguistics.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam M. Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, Yaguang Li, Hongrae Lee, Huaixiu Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, I. A. Krivokon, Willard James Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Hartz Søraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Díaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravindran Rajakumar, Alena Butryna, Matthew Lamm, V. O. Kuzmina, Joseph Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. 2022. Lamda: Language models for dialog applications. *ArXiv*, abs/2201.08239.

Dietrich Wettschereck and Thomas Dietterich. 1993. Locally adaptive nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann.

Yuhuai Wu, Markus N. Rabe, DeLesley S. Hutchins, and Christian Szegedy. 2022. Memorizing transformers. In *ICLR*.

Frank F. Xu, Junxian He, Graham Neubig, and Vincent J. Hellendoorn. 2022. Capturing structural locality in non-parametric language models. In *ICLR*.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373.

Hamed Zamani, Fernando Diaz, Mostafa Dehghani, Donald Metzler, and Michael Bendersky. 2022. Retrieval-enhanced machine learning. In *SIGIR '22*.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. In *ICLR*.

Xin Zheng, Zhirui Zhang, Junliang Guo, Shujian Huang, Boxing Chen, Weihua Luo, and Jiajun Chen. 2021. Adaptive nearest neighbor machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 368–374, Online. Association for Computational Linguistics.