Abstract

We present a language model that combines a large parametric neural network (i.e., a transformer) with a non-parametric episodic memory component in an integrated architecture. Our model uses extended short-term context by caching local hidden states—similar to transformer-XL—and global long-term memory by retrieving a set of nearest neighbor tokens at each timestep. We design a gating function to adaptively combine multiple information sources to make a prediction. This mechanism allows the model to use either local context, short-term memory, or long-term memory (or any combination of them) on an ad hoc basis depending on the context. Experiments on word-based and character-based language modeling datasets demonstrate the efficacy of our proposed method compared to strong baselines.

1 Introduction

Human language processing is facilitated by complex systems interacting together. A core component that enables such a process is human memory. Memory in humans consists of specialized systems, which form a basis for intelligent behaviors (Tulving, 1985; Rolls, 2000; Eichenbaum, 2012). For language processing, working (short-term) memory is a temporary storage that can be used to comprehend sentences and follow conversations. Episodic (long-term) memory stores individual experience and events. Semantic memory stores facts and knowledge about words and concepts.\footnote{We refer readers to Nematzadeh et al. (2020) for discussions on human and artificial language processing memory systems.}

In artificial language processing systems (e.g., language models), a popular approach to design a better model is by encoding all of the desired knowledge (to produce grammatical sentences, process long text, remember events, etc.) in the weights of a large parametric neural network via end-to-end training. We see an increasingly larger transformer become a better language model (Radford et al., 2018, 2019; Shoeybi et al., 2019; Brown et al., 2020). In this \textit{scale} approach, the knowledge is implicitly represented in the weights of a parametric neural network, and it is not straightforward to interpret whether a model contains a particular knowledge without asking the model to produce a response—for example, via a cloze-style question (Petroni et al., 2020) or a prompt (Brown et al., 2020).

An alternative strategy is to design a \textit{modular} architecture that separates memory storage and computational processing, where each module has a clear purpose. Recent progress in memory-augmented neural networks has given rise to many variants of memory-augmented transformer language models that fall under this category. For example, attempts to incorporate extended local context to a neural network—such as those found in neural cache (Grave et al., 2017c), transformer-XL (Dai et al., 2019), compressive transformer (Rae et al., 2020), performers (Choromanski et al., 2021), longformer (Beltagy et al., 2020), and reformer (Kitaev et al., 2020)—can be seen as models of working memory. Models of episodic memory include $k$NN-LM (Khandelwal et al., 2020) and architectures that are designed for more complicated tasks such as question answering (de Masson d’Autume et al., 2019; Guu et al., 2020) and machine translation (Khandelwal et al., 2021). In machine learning and natural language processing, memory-augmented neural networks is used to refer to all types of memory systems.

In this paper, inspired by the modular design of human memory systems, we present a language model architecture (\textsc{Spalm}) with storage modules that resemble working and episodic memory systems, which we combine with a large parametric neural network that is responsible for
computation (§2). Our hypothesis is that encouraging each component to focus on a specific function (e.g., storing long-term information, capturing extended context, modeling local information) facilitates easier training that produces an overall better language model.²

Specifically, we follow transformer-XL (Dai et al., 2019) to capture extended context by caching hidden states in a temporary short-term memory. For long-term context, we use a persistent key-value database and perform sparse retrieval with (approximate) \( k \)-nearest neighbors. In contrast to previous language models that either interpolate output probabilities (Merity et al., 2017; Grave et al., 2017c; Khandelwal et al., 2020; Kassner and Schutze, 2020) or use input concatenation (Guu et al., 2020; Xu et al., 2020) to combine information from different sources, we design a context-dependent gating mechanism to incorporate local, extended, and global context. We discuss similarities and differences to related work in §3.

In language modeling, many tokens can be predicted from their local context without requiring long-term information. Our model can adaptively decide whether the current (local) context is enough, or whether it needs to use information from the short-term and/or long-term memory.

In §4, we compare SPALM with strong baselines—including transformer-XL and \( k \)NN-LM—on word-based and character-based language modeling. Our positive results establish the benefit of the proposed architecture. They also indicate the generality of our approach and its potential applicability to other sequence modeling tasks.

We analyze how SPALM uses long vs. short-term context (§5) to better understand how the model operates when making predictions. We conclude by discussing limitations and future directions (§6).

2 Model

We consider a language model that takes as input a sequence of words \( x_{\leq t} = \{x_0, \ldots, x_t\} \) and outputs a probability distribution of the next word

\[
p(x_{t+1} \mid x_{\leq t}; W).\]

Given a corpus of \( T \) words, the log likelihood of the corpus is:

\[
\mathcal{L} = \sum_{t=0}^{T} \log p(x_{t+1} \mid x_{\leq t}; W),
\]

where \( x_0 \) is the start of sentence symbol.

SPALM consists of three main components: (i) a large parametric neural network in the form of a transformer to process local context, (ii) a short-term memory to store extended context, and (ii) a non-parametric episodic memory module that stores information from long-term context. We integrate these components in a single architecture with a gating mechanism. Figure 1 shows an illustration of our model, which we discuss in detail below.

²We note that SPALM is not intended to be a model of human language processing system. We merely take inspirations from human memory systems to design a better artificial language model.
2.1 Base Model

We use transformer (Vaswani et al., 2017) as our base model. Given the input sequence \( x_{\leq t} \), transformer performs multiple layers of self-attention between every pair of tokens in the input sequence to produce token representations.

A core limitation of transformer is that its computational complexity is quadratic in the input sequence length. As a result, instead of considering all previous tokens \( x_{\leq t} \), transformer truncates the input to be the most recent \( N \) words \( \tilde{x}_{\leq t} = \{x_{t-N+1}, \ldots, x_t\} \) and only operates on this fixed-length window in practice. A large transformer, no matter how many parameters it has, is limited by the input sequence length.

2.2 Short-term Memory

We use transformer-XL (Dai et al., 2019) as our working memory model. Given the current context \( \tilde{x}_{\leq t} \), denote the extended context of length \( M \) by \( \tilde{x}_{\leq t-N} = \{x_{t-N-M+1}, \ldots, x_{t-N}\} \). In other words, the extended context is the \( M \) tokens prior to the current context. In transformer-XL, hidden states for \( \tilde{x}_{\leq t-N} \) (obtained from a previous computation when predicting \( x_{t-N+1} \)) are cached. They are then used as additional states that can be attended to during the forward pass when computing hidden states for the current context \( \tilde{x}_{\leq t} \), but the values of the states are not updated during the backward pass to save computation time.

Formally, denote the hidden state for \( x_t \) at layer \( r \) by \( h_t^r \). Denote the hidden states associated with the current (truncated) context \( \tilde{x}_{\leq t} \) by \( H^r = [h_{t-N}^r, \ldots, h_t^r] \) and the hidden states associated with the extended context \( \tilde{x}_{\leq t-N} \) by \( E^r = [S_G(h_{t-N-M+1}^r), \ldots, S_G(h_{t-N}^r)] \), where \( S_G \) is the stop gradient function. Together, \( H^r \) and \( E^r \) are used as an input to an attention function (with relative positional encodings) where each vector is transformed into a key, value, query triplet which are used to produce \( H^{r+1} \) (i.e., hidden states for the next layer).

Note that while transformer-XL extends the context window, the extra information is still “local” with respect to the sequence.

2.3 Long-term Memory

We design a long-term episodic memory module that allows our language model to retrieve “global” information. The long-term memory module is implemented as a key-value database. The key is a vector representation of a context \( \tilde{x}_{\leq t} \) (i.e., we compress \( \tilde{x}_{\leq t} \) into a vector). Each context is paired with the output token for that context \( x_{t+1} \), which is stored as the value. In our experiments, we store a key-value entry for each context-token pair in the training corpus, so the number of entries is equal to the number of tokens in the training corpus.

There are many choices that can be used for the key representation, which we denote by \( d_i \). For example, we can use \( h_t^R \) or a separate pretrained encoder such as BERT (Devlin et al., 2018). We pretrain a vanilla transformer language model and use the final-layer hidden state for \( d_i \).

For predicting a new token \( x_{t+1} \) given \( \tilde{x}_{\leq t} \), we first obtain \( d_i \) from the separate pretrained language model. We then use \( d_i \) to do a \( k \)-nearest neighbor search on the database. Since \( d_i \) is a contextual representation, this search finds contexts that are similar to \( \tilde{x}_{\leq t} \) in the database. For the top \( k \) such contexts, we retrieve the values associated with those contexts, which are the output (next) tokens when those contexts are encountered in the past. Denote the output tokens retrieved from the database by \( y_1, \ldots, y_K \).

For each \( y_k \), we create a vector representation \( y_k \) by using the same word embedding matrix that is used in our base model. We then combine the long-term memory information obtained from the database with the extended local context with a gating mechanism as follows:

\[
\begin{align*}
    m_i &= \sum_{k=1}^{K} \frac{\exp y_k^\top h_t^R}{\sum_{j=1}^{K} \exp y_j^\top h_t^R} y_k \\
    g_t &= \sigma (Vh_t^R) \\
    z_t &= (1 - g_t) \odot m_t + g_t \odot h_t^R \\
    p(x_{t+1} \mid x_{\leq t}) &= \text{softmax}(z_t; W),
\end{align*}
\]

where \( V \) is a parameter matrix, \( \sigma \) is the sigmoid function, and \( W \) is the word embedding matrix that is shared for input and output word embeddings (Inan et al., 2017).\(^3\)

In the above formulation, we first aggregate information from \( y_1, \ldots, y_K \) with a simple attention mechanism using \( h_t^R \) as the attention query.\(^4\)

\(^3\)In a preliminary experiment, we incorporate the nearest neighbor distance as a bias term in the computation of \( m_t \). However, this does not improve performance, so we use the above equation in the final model.

\(^4\)It is possible to first transform \( h_t^R \) (e.g., by doing a linear projection) before using it as an attention query. We choose
We then use a context-dependent gate $g_t$ that decides how much the model needs to use local information ($h^R_t$) versus long-term information ($m_t$) for making the current prediction based on the current context. Note that given the database, the only additional parameter that needs to be trained is $V$. The result is a language model that is able to rely on short-term context for “easy” predictions while using long-term context for “hard” predictions by adaptively combining short-term and long-term memory at the architectural level.

### 2.4 Training Details

As discussed previously, we first train a standard transformer language model and use it as an encoder to compute key representations $d_i$ for the episodic memory database. Since our training datasets contain hundreds of millions of tokens, for computational considerations, we do not update the key representations when training the overall model. This allows us to fix the set of nearest neighbors for each token, making training of the overall model to be almost as fast as a vanilla transformer-XL in terms of wall-clock time after we precompute neighbors for each token. The value encoder, on the other hand, is updated during training since we use the word embedding matrix to represent $y_k$.

$k$-nearest neighbors on hundreds of millions of tokens can be computationally expensive. We use the publicly available ScANN\(^5\) (Guo et al., 2020) to do this efficiently, which is a quantization-based technique to do fast and accurate maximum inner product search.

We note that it is conceptually possible to train all components of our model in an end-to-end manner. However, we leave end-to-end training to future work. In addition, while it is possible to continually grow the long-term memory module by storing new tokens from evaluation data, we choose to do a static evaluation. Therefore, we do not compare with dynamic evaluation models which adapt language models to evaluation data. We next discuss comparisons to existing nearest neighbor retrieval-based language models.

### 3 Comparisons to Previous Work

**kNN-LM.** There are several language models that are related to our proposed method. The closest one is $k$NN-LM (Khandelwal et al., 2020), which is another language model that is augmented with a nearest neighbor retrieval mechanism. $k$NN-LM is an ensemble technique that is designed to be used only at evaluation time. In $k$NN-LM, a pretrained language model (e.g., a transformer) is combined with another retrieval-based language model by interpolating their probabilities:

$$
p(x_{t+1} | x_{<t}) = \lambda p_{LM}(x_{t+1} | x_{<t}) + (1 - \lambda) p_{kNN}(x_{t+1} | x_{<t}).
$$

The interpolation weight $\lambda$ is tuned at the corpus level on a development set.

While this post hoc integration method used by kNN-LM has its merits (e.g., very practical, fast to incorporate to any model since it does not require additional training), our focus is on designing a model that combines short-term and long-term memory at the architecture level. Our motivation is twofold. First, interpolating the language model weights at the corpus level forces the model to use the same interpolation weight $\lambda$ for $p_{LM}$ and $p_{kNN}$ for each token in the corpus. It cannot adaptively combine short-term and long-term information at the token level based on the context. In addition, $\lambda$ needs to be tuned on an extra development set.\(^\text{6}\)

$P_{ALM}$, on the other hand, is able to adjust the weights placed on $m_t$ and $h_t^R$ when constructing $z_t$ differently for different tokens. Second, we believe that integration of different memory modules at the architectural level is a more natural approach that could help pave the way for applications with other memory sources (e.g., knowledge bases, images, videos)—where the memory output is not in the same space as the prediction output (i.e., words) and an interpolation technique cannot be used.

We compare with $k$NN-LM in our experiments. Since interpolating model probabilities is an ensembling technique that is independent of the architecture, we also show that our language model can be further ensembled with $p_{kNN}$ if necessary.

\(^6\)We note that it is possible to incorporate this interpolation technique during the training phase of a language model as well to avoid having to tune $\lambda$ on a development set. For example, Neubig and Dyer (2016) show how to train a mixture of experts language models, where the mixture weights are inferred. However, the efficacy of this approach as a memory-augmented language model has not been explored.

\(^5\)https://github.com/google-research/google-research/tree/master/scann.

\(^6\)An untransformed version in our experiments to minimize the number of new parameters in $P_{ALM}$. We leave explorations on the best transformation of $h_t^R$ to future work.
Cache-based Language Models and Pointer Networks. Cache-based language models (Grave et al., 2017c; Merity et al., 2017) store pairs of hidden states and output tokens from previously seen tokens (within a limited context length) in a cache. The best variant of the method uses an interpolation (ensemble) method similar to \( k_{\text{NN-LM}} \) to combine information from the cache and the backbone language model. This class of models temporarily stores \( M \) past hidden states (typically, in the order of thousands), so it is a working-memory model as opposed to long-term memory. In addition, they also rely on interpolating probabilities of a backbone language model and a cache component (similar to \( k_{\text{NN-LN}} \) when the cache size is unbounded).

Other Retrieval Augmented Methods. An early version of a neural language model that includes a retrieval component is presented in Guu et al. (2018). They follow a retrieve-then-edit approach to generate a sentence, which requires approximating an expectation over an edit prior.

Outside language modeling, there are several recent retrieval-augmented methods that have been used for question answering (de Masson d’Autume et al., 2019; Guu et al., 2020; Xiong et al., 2021; Kassner and Schutze, 2020), controllable generation (Xu et al., 2020), machine translation (Bapna and Firat, 2019; Khandelwal et al., 2021), and one-shot learning (Kaiser et al., 2017). These methods share some similarities with our proposed model since it involves a retrieval component. However, the difference in the downstream tasks (language modeling vs. question answering vs. machine translation), results in different items that are stored in and retrieved from the key-value database. For example, de Masson d’Autume et al. (2019) store and retrieve question-answer pairs, Guu et al. (2020) have a database of passages of an article, and Khandelwal et al. (2021) use source and target sentences. Our gating mechanism resembles the gate that is used to incorporate information from a non-parametric memory component to a machine translation model in Bapna and Firat (2019), although the memory entries, the decoder architecture, and the downstream task are different.

In addition, these models are only models of long-term memory. Their evaluation tasks often do not need working memory because the entire input sequence is short enough that it can be fed as an input to a transformer as a whole.

4 Experiments

We use word-based and character-based English language model datasets—WikiText 103, WMT, and enwik8—to evaluate our proposed method. We provide descriptive statistics in Table 1 and discuss each dataset in the respective section below.

4.1 Implementation Details

We use Adam (Kingma and Ba, 2015) as our optimizer. For word-based language modeling, we use adaptive softmax (Grave et al., 2017b). We apply dropout with a rate of 0.25. All models are trained on 128 Tensor Processing Units until convergence with batch size 256.

4.2 WikiText-103

Our first dataset is WikiText-103 (Merity et al., 2017). We compare four models: vanilla transformer, transformer-XL, \( k_{\text{NN-LM}} \), and \( S_{\text{PLM}} \). For WikiText-103, all of our models have 18 layers and 512 hidden dimension size with a total of 142M parameters. We set the sequence length to 512. For transformer-XL, we set the short-term memory length to 512 during training and 512 or 3072 at test time. We use 4 nearest neighbors for \( k_{\text{NN-LM}} \) and \( S_{\text{PLM}} \) and analyze the effect of varying the number of neighbors in §5.4. For \( k_{\text{NN-LM}} \), we use the transformer-XL model to obtain \( p_{\text{LM}} \), compute \( p_{\text{NN}} \) based on the nearest neighbor distance similar to Khandelwal et al. (2020), and tune \( \lambda \) from \{0.05, 0.1, 0.2, 0.3, 0.4\} on the development set.

Table 2 shows perplexity on WikiText103. Our implementation produces results that are in the same range as state-of-the-art numbers, demonstrating the strength of our baselines. Transformer-XL outperforms transformer, and interpolating the probability of transformer-XL.
with \( kNN \) (i.e., \( kNN-LM \)) improves the result further. This is true both with transformer-XL (short-term) memory length of 512 and 3072. Comparing \( kNN-LM \) with \( SPALM \), \( kNN-LM \) is marginally better on the test set even though \( SPALM \) is marginally better on the development set.

We observe further improvements in \( SPALM \) by interpolating its output probability with the output probability from \( p_{kNN} \) which is used by \( kNN-LM \), resulting in the best model with a perplexity of 17.6. We find this interesting since \( SPALM \) and \( p_{kNN} \) use the exact same four neighbors for each token. It indicates that there are some complementary benefits in incorporating long-term memory into training and interpolating probabilities at test time.

### 4.3 WMT

In the second experiment, our goal is to evaluate on a much larger dataset. We construct a language modeling dataset from the English portion of the WMT 2019 dataset, publicly available at http://www.statmt.org/wmt19/. WMT contains news articles from different months. We use articles from January to October for training, a portion of articles in November for development, and a portion of articles in December for test.\(^7\)

The resulting WMT dataset is approximately ten times larger than the WikiText-103 dataset.

Similar to the previous experiment, we evaluate models with 18 layers and 512 hidden dimension size with a total of 148 million parameters. We set the sequence length to 512, the transformer-XL short-term memory length to 512 for training and evaluation, and the number of neighbors for \( SPALM \) and \( kNN-LM \) to 4.

Table 3 shows results on this dataset. Consistent with the previous experiment, \( kNN-LM \) outperforms transformer-XL and transformer. \( SPALM \) outperforms all of them by a considerable margin on the test set. Unlike WikiText-103, we observe no further improvement interpolating the probabilities of \( SPALM \) with \( p_{kNN} \). The results also indicate that when the distributions of the dev and test sets can be different (e.g., articles from different months), \( kNN-LM \) that relies on tuning \( \lambda \) on the dev set is more sensitive to performance discrepancy between the dev and test sets.

#### 4.4 enwik8

In the third experiment, we evaluate our models on character-level language modeling. Compared to word-level language modeling, character-level has a much smaller output space (in the order of hundreds instead of tens of thousands) and has a different characteristic in how much local vs. global contexts are needed to make a good prediction.

The enwik8 dataset (Hutter, 2012) is a benchmark for character-level language modeling.

---

\(^7\)We sample articles written in November and December in chronological order to create development and test sets of approximately 1 million tokens (there are almost 100 million tokens if we use all of the articles in each month).
Table 4: Bits per character (BPC) on enwik8. The top rows contain results taken from other papers: (a) transformer-XL (Dai et al., 2019), (b) longformer (Beltagy et al., 2020), and (c) compressive transformer (Rae et al., 2020). The (log likelihood) difference between SPALM and transformer-XL on the test set is statistically significant (Wilcoxon signed-rank test, \( p < 0.05 \)).

We use a 24-layer model with 512 hidden size. In total, our model has 100 million parameters. We set the sequence length to 768, the transformer-XL short-term memory length to 1536 for training and 4096 for evaluation. Since character-level language models has a much smaller output space, we only retrieve two neighbors per character.

We show the results in Table 4. Unlike the previous two word-level language modeling results, kNN-LM underperforms transformer-XL. However, SPALM outperforms all other models. We note that a decrease of 0.01 is considerable on this dataset under the BPC metric. Similar to WMT, interpolating the probabilities of SPALM with \( p_{kNN} \) does not improve performance. These results highlight a major strength of our proposed model: Uniformly setting interpolation weights at the corpus level decreases performance (i.e., kNN-LM), but allowing the model to flexibly decide when to use long-term vs. short-term memory is beneficial.

Since character-level and word-based language modeling are characteristically different, the success of our model on this dataset indicates its applicability to other sequence modeling problems. We leave such explorations to future work.

5 Analysis

We have demonstrated the efficacy of our proposed method on three language modeling tasks. In this section, we analyze the model to gain more insights into how it works.

5.1 Examples of Neighbors

We inspect the neighbor tokens that are retrieved from the long-term memory for news articles in the WMT development dataset. We provide a cherry-picked example in Figure 2. As the model sees more tokens in a sequence, the long-term memory model becomes more accurate. We observe interesting cases such as when predicting a named entity (e.g., Elizabeth Warren), even if the long-term memory model fails to retrieve the correct first name, it usually is able to retrieve the correct last name after seeing the first name (because the entity exists in the training corpus). We observe this phenomenon in many other examples as well. We can also see that the retrieved neighbors are generally relevant even when they do not match a target word exactly—for example, when predicting names of days, dollar amounts, time quantifiers, and common phrases.

We next investigate neighbors on enwik8 development set (Figure 3). We observe that information from the long-term memory helps when completing common words (e.g., before and invasion), named entities (e.g., Soviet), and corpus-specific formats (e.g., double square brackets).

We note that the above examples are only provided to give a better insight into our model. It is entirely plausible that a baseline parametric model is already able to predict correctly from the local context. Nonetheless, directly providing this information as a long-term context helps our model learn better, as evident from the superior performance of SPALM on our three evaluation datasets.

5.2 Output Analysis

We search for predictions where SPALM significantly outperforms transformer-XL and transformer to understand when modeling local information is sufficient (i.e., vanilla transformer), when adding extended context helps (i.e., transformer-XL), and when storing long-term information is useful (i.e., SPALM). We show three examples from the WMT test set in Figure 4.
For Warren & Wednesday briefly a 5 billion to equity
 Warren may Tuesday praised wiping 16 trillion in funding
 Perhaps Warren has Sunday stood breaking 10 billion for federal
 Like Warren , Monday defended using 166 trillion in spending
 Elizabeth Warren on Friday proposed $ 20 trillion in federal

gets in 10 course eight . fight even care for
funding over the next three . upgrade them coverage for
funds over 10 next five in improve American - to
, over a next 10 , invest a insurance services
spending over the next decade to provide health care to

more community as the rates . the middle class
everyone child , a taxes on the wealthy class
some baby , co taxes . the middle class
every American by triggering taxes on all middle class
every American without raising taxes on the middle class

Figure 2: A sequence of words from WMT and its four nearest neighbors at each position. We break down the sequence into four blocks. The bottom row of each block in blue represents the original sequence, which is Elizabeth Warren on Friday ... the middle class. Each row above it represents a nearest neighbor token (starting from the first neighbor at the second-bottom to the fourth neighbor at the top) that is used when predicting that particular word. We highlight matching neighbor–target words in green. We provide a more detailed discussion in §5.1.

U o e h at fore t hi d en iet - U nta
er h e re b e f or e t he f av iet ' Unva
Even b e f o r e t he S ovi e t i nva
s i o n , b n t he [ n d of f t [ 1 6 7 5 ] ]
s i o n a nt A he e nd of f t [ 4 3 9 9 ] ]
s i o n at t he e nd of [ [ 1 9 7 9 ] ]

Figure 3: A sequence of characters from enwik8 and its two nearest neighbors at each position. We break down the sequence into two blocks. The bottom row of each block in blue represents the original character sequence, which is Even before ... [[1979]]. The two rows above it represent the nearest neighbors (the first nearest neighbors at the second bottom row and the second nearest neighbors at the top row) that are used when predicting that particular character. We highlight matching neighbor–target characters in green. We provide a more detailed discussion in §5.1.

... Several companies have pulled their advertising from the TV show following the revelations ... 
... Liberal Democrat leader Jo Swinson has said she would work with Donald Trump in government as ...
... Additionally , the airline has purchased six Boeing 787 - 9 Dreamliner aircraft that are scheduled ...

Figure 4: Three example sequences from the WMT test set. We highlight words where both \( p_{TXL} \) and \( p_{SPALM} \) are larger than \( p_{transformer} + 0.1 \) in green and \( p_{SPALM} > p_{TXL} + 0.1 \) in blue. See §5.2 for details.

While it is difficult to find consistent patterns, we observe that SPALM is generally better than both transformer and transformer-XL for predicting (completing) common phrases and named entities (that exist in the training set), especially when they are encountered for the first time and have not appeared in the extended context (e.g., pulled their advertising from, Liberal Democrat, Jo Swinson, Boeing 787-9 Dreamliner).

On the other hand, we also see a few cases when transformer-XL outperforms SPALM. These are usually associated with scenarios where the same word has appeared in the extended context. While SPALM uses information from the extended context as well, the probability is smoothed over by
5.3 Gate Vectors

Our model has a gating mechanism to regulate information flow from the current context, short-term, and long-term memory. We analyze the values of the gate for tokens in WMT and enwik8. Figure 5 shows histograms of the distribution of gate values.

We observe different characteristics for WMT and enwik8. On enwik8, the gate values are concentrated around 1. This indicates that the model relies on local context most of the time. This can explain why kNN-LM does not work well on this dataset. On WMT, the values are less concentrated around 1. This suggests that the model uses long-term memory more than on enwik8. SPA LM is able to learn when the long-term memory is needed and when it is not in both cases.

We next look into the value of the gates for a specific sequence in the development set in Figure 6. We note that we only show a small dimension subset from the gate vector for readability, so we caution against drawing a conclusion about how the model works from this. Our goal is only to get a better understanding of what happens when the model makes predictions. Comparing WMT and enwik8, we see that in general on WMT the model tends to reserve some dimensions to propagate information from the long-term memory, as indicated by vertical red lines. On enwik8, the model relies on long-term information when completing a known word such as Egypt, as shown by more horizontal red patterns when forming this word. For other characters, the value of the gates are closer to one, which shows that the model relies more on local and extended short-term context.

5.4 Number of Neighbors

We use four neighbors for our word-based and two neighbors for our character-based language models. These values are chosen from preliminary experiments on a small subset of the datasets.

We show SPA LM perplexity on development set for WikiText-103 when we vary the number of neighbors in Table 5. We see that using one nearest neighbor is enough to obtain good performance, with a slight advantage when we use four neighbors. The performance starts to degrade as we use 8 and 16 neighbors. We choose to use four neighbors in our experiments since kNN-LM—which also uses the same set of neighbors—performs better with four neighbors instead of one, and we want to keep the comparison as fair as possible.

One notable difference between our neighbors and those that are used in kNN-LM (Khandelwal et al., 2020) is that we do not limit the search of the neighbors to the same token as the current input token (\( I(x_i = x_t) \)). While this allows the model to combine information from related words (not constrained to an exact match), it could introduce noise when the number of neighbors is large.

We observe that our representation learning model (i.e., the baseline transformer) is able to retrieve relevant neighbors most of the time. It retrieves the exact output token as the first neighbor 33%, 44%, and 70% on WikiText-103, WMT, and enwik8 development sets, respectively.

6 Discussion

Summary of Contributions. We present a semiparametric language model (SPA LM) that combines local context, short-term memory, and long-term memory to make predictions. Experiments on word-based and character-based language models demonstrate the benefit of our proposed method.

Limitations. The biggest limitation is the necessity to retrieve neighbors for each training token. Such a process—even though can be fully parallelized—is time consuming. In our experiments, it takes 6–8 hours to obtain neighbors for WikiText-103 and enwik8 with 1,000 CPUs and 18 hours for WMT with 9,000 CPUs.

Future Directions. Our modular approach that combines multiple memory systems at the
architectural level opens up the possibility to incorporate additional memory from other modalities (e.g., images) or structured knowledge bases. We also envision a next-generation model that does not have to retrieve information from long-term memory for every token and only does it for those that require global context. A model that learns how to do this would save a considerable amount of training and test time—since it would significantly reduce the number of search that needs to be performed. Our language model that integrates retrieval into training is a first step in this direction.

Acknowledgments

We thank the action editor (Mihai Surdeanu) and three anonymous reviewers for helpful comments on an earlier draft of this article.

References


