Regularized Training of Nearest Neighbor Language Models

Jean-Francois Ton* University of Oxford
Walter Talbott Apple
Shuangfei Zhai Apple
Joshua Susskind Apple

Abstract

Including memory banks in a natural language processing, architecture increases model capacity by equipping it with additional data at inference time. In this paper, we build upon kNN-LM (Khandelwal et al., 2020), which uses a pre-trained language model together with an exhaustive kNN search through the training data (memory bank) to achieve state-of-the-art results. We investigate whether we can improve the kNN-LM performance by instead training a LM with the knowledge that we will be using a kNN post-hoc. We achieved significant improvement using our method on language modeling tasks on `WIKI-2` and `WIKI-103`. The main phenomenon that we encounter is that adding a simple L2 regularization on the activations (not weights) of the model, a transformer (Vaswani et al., 2017), improves the post-hoc kNN classification performance. We explore some possible reasons for this improvement. In particular, we find that the added L2 regularization seems to improve the performance for high-frequency words without deteriorating the performance for low-frequency ones.

1 Introduction

The problem of language modeling (LM) usually consists of two main challenges. Firstly, mapping the context, i.e. the sentence prefixes, to a vector representation, and secondly using this representation to predict the subsequent word. In Khandelwal et al. (2020), the authors claim that the first problem is much easier to solve. Hence, given a pre-trained LM, they post-hoc modify the representation using a k-nearest neighbor scheme (kNN) and achieve significant improvements on challenging datasets, such as `WIKI-103`.

Given that kNN improves the overall language modeling of a pre-trained network, we examine training strategies that can make the underlying network's representations more amenable to the kNN step. Our results show improvements over applying kNN to a generic LM network.

We first explore a simple learning scheme for the language model, where during training we intentionally push representations that predict the same word closer together in the L2 sense, using a Momentum Contrastive (MOCO) (He et al., 2020) style implementation. We go on to note that this MOCO style learning can be replaced by simply adding L2 regularization to the activation of the layer used for kNN, eliminating implementation complexity. Lastly, we present some initial experiments toward understanding why this L2 regularization brings improved performance.

2 Background

Our work builds upon kNN-LM (Khandelwal et al., 2020). In essence, kNN-LM tackles the problem of how to improve a trained LM's representations, and how to adapt LMs to capture non-frequent sentences that are usually forgotten by the model during training. kNN-LMs achieve significantly higher performance through a simple interpolation between the original LM predictions and the kNN predictions.

At inference time, given a new context sentence, kNN-LM works as follows:

1. The context sentence c_i is passed through the pre-trained network to produce a representation $r_i^{context} \in \mathbf{R}^d$ as well as the corresponding logits y_i^{LM} to predict the next word.

2. $r_i^{context}$ is used to find the k-nearest neighbors in the training data. The logits y^{kNN} are computed by a weighted average of the neighbors' labels, using the inverse exponential distance as the weight for each neighbor.

3. The logits are interpolated to give the final prediction:

$$y_{final} = \lambda y^{kNN} + (1 - \lambda) y^{LM},$$

*Work done as an intern at Apple

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop, pages 25 - 30

where λ is the interpolation parameter that can be tuned on validation data.

This simple post-hoc implementation allows Khandelwal et al. (2020) to improve upon the SOTA in LM by a significant margin. One thing to note about kNN-LM is that they do not need to retrain the LM and hence the whole algorithm can be run on CPU only. Furthermore, kNN-LM uses *FAISS*, which is an efficient library that allows them to quickly find kNNs.

One detail to note in (Khandelwal et al., 2020) which was crucial for this work was that the authors tried both the inner product and the L2 for their distance metric in kNN. They concluded that L2 worked significantly better. This observation implies the fact that the default training recipe of LMs *implicitly* prefers one distance over the other. Given that we know that a post-hoc kNN adaptation significantly improves the performance, it is natural to ask whether we could train a LM with this in mind. In the next section, we describe how to adapt the training of the LM for this purpose.

3 Proposed Method

In our initial attempt, we experimented with the idea of explicitly minimizing the L2 distance between context vectors that predict the same target word. This strategy directly mirrors the use of context vectors at the kNN step, and we hoped that training the representations in a way similar to testing will further improve the effectiveness of kNN LM. However, a naïve implementation of it is infeasible due to having to store all the representation in memory. We then resorted to a MOCO-(He et al., 2020) style training scheme. Specifically, for each target word w, we construct a queue Q of fixed length L, which stores the recent L context representations for w. During training, we optimize a regularized objective as follows:

$$\mathcal{L}_{new} = L_{CE} + \omega \sum_{j=1}^{\mathrm{N}} \sum_{i=1}^{\mathrm{L}} ||sg(Q_i^{w_j}) - r_j||^2, \tag{1}$$

where N is the batch size, r_j is the context representation of the jth word, $Q_I^{w_j}$ is the ith item in the queue corresponding to the jth target word w_j; ω is the regularization parameter; $sg(\cdot)$ is the stop gradient operator. Specifically, Q is updated with a momentum target encoding network which is initialized with the same parameters of the LM, similar to MOCO (He et al., 2020).

Empirically, we found that Equation 1 provides a practical solution and yields improved representations for the kNN LM, as shown in Fig 1. In particular, we see from the figure that there is an optimal value for ω for which the added regularization seems to improve the kNN LM model perplexity significantly i.e. from 76 to 70 at $\omega = 2$ (orange line). The interesting part to note in this case, is the fact that the standard LM (without post-hoc kNN) does not vary much up to $\omega = 5$, leading us to conclude that the added regularization has changed the representation in a way that kNN can more effectively exploit the neighbors.

However, the use of the queue and momentum target network still adds overhead to a large-scale model training as we are required to access the queue for each batch. Hence we tried to decrease Q and L, which interestingly did not decrease the performance at all and therefore, to promote efficiency, we tested an even simpler formulation, where we replace Q with all **zero** vectors. This eliminates the need to explicitly construct and update the queue, while instead encouraging the model to learn conservative representations w.r.t. the L2 norms of its context representations. The corresponding loss is as follows:

$$\mathcal{L}_{new} = L_{CE} + \omega \sum_{j=1}^{\mathrm{N}} ||r_j||^2. \tag{2}$$

To our surprise, Equation 2 yields similar performance to Equation 1 in practice see Table 1, while being much easier to implement and tune. This is a new interesting finding that we will try to explain in the ablation study below. We thus use Equation 2 as the default loss function in our experiments unless otherwise mentioned.

4 Experiments

We tested our method on the `WIKI-2` and `WIKI-103` datasets, which are widely used benchmarks for language modeling. We are interested in demonstrating two empirical results: improved performance using our approach over that of kNN-LM, and exploring a possible mechanism for this improved performance.

4.1 Experimental setup

Dataset `WIKI-2` is a benchmark with 30k word vocabulary and consists of 2M tokens. `WIKI-103`

	kNN-LM ($\omega=0$)	$\omega=0.1$	$\omega=1.0$	$\omega=10.0$
Train Ppl. LM	19.99	20.05	20.11	21.37
Valid Ppl. LM	75.96	75.68	76.37	81.29
Valid Ppl. kNN-LM	74.11	73.13	**70.63**	80.52

Table 1: Experiments on `WIKI-2` with corresponding validation perplexity using L2 regularization. We see that a weighting of $\omega = 1$ yields the best performance for our method

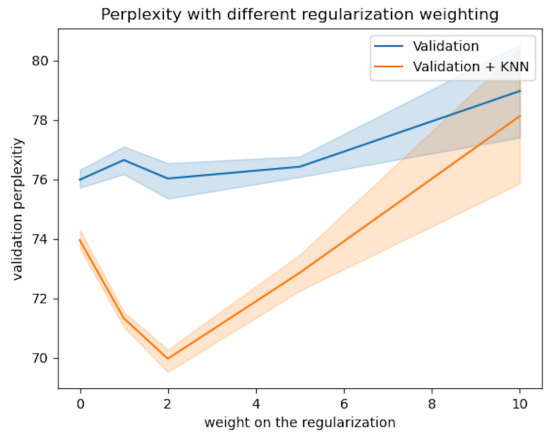

Figure 1: Validation perplexity on `WIKI-2` of the LM before (blue) and after (orange) adding kNN. NOTE: `weight=0` corresponds to the standard version that does not include our added MOCO-style regularization term i.e. kNN-LM from Khandelwal et al. (2020)

is a benchmark with 250k word vocabulary and consisting of 103M tokens (Merity et al., 2016).

Language Model Architecture For the language model architecture, we will be using the exact setup as described in (Khandelwal et al., 2020). This setup consists of the language model (Baevski and Auli, 2018), which consists of 16 layers, each with 16 self-attention heads, 1024 dimensional hidden states, and 4096 dimensional feedforward layers. Thus, following (Baevski and Auli, 2018), this LM has adaptive inputs and an adaptive softmax (Joulin et al., 2017) with tied weights (Press and Wolf, 2016) for all our experiments. We trained the each language model on a `Tesla V100` with `40GB` of RAM.

In addition, we follow the exact same training procedure as in (Khandelwal et al., 2020) and refer to their paper for further details on the training parameters. The only difference in terms of implementation is the MOCO style learner as well as the L2 regularization added to the final layer. Lastly, we would like to note that while crossvalidating though the interpolation parameter λ we note that for all models, $\lambda = 0.3$ works the best which is similar to the finding in (Khandelwal et al., 2020).

4.2 Experiments on `WIKI-2`

We first apply our proposed method on the standard `WIKI-2` dataset, where we run each configuration 5 times and plot the standard deviation, as seen in Figure 1. Note that $\omega = 0$ in Figure 1 corresponds to the standard kNN-LM version, i.e. without the added term in the loss. Comparing Figure 1 and Table 1, we see that the MOCO and L2 approaches produce similar results. From these results, we note the following phenomena:

1. A clear "U"-shape demonstrating the added benefit of our loss term on the validation perplexity of the LM for moderate values of ω.

2. Training performance does not decrease for moderate values of ω, showing that the extra term does not destroy training and generalization of the standard LM.

3. There is no difference in terms of validation perplexity between the standard LM and our version **before** applying kNN, but there is a significant difference **after** applying kNN. Our approach likely finds a different local minimum for the language model that is better suited for kNN.

The above finding supports our belief that using our added regularization, we are able to find better representations, that can subsequently be used more efficiently when for kNN LM. Next, we apply our methods on the much bigger data `WIKI-103`.

4.3 Experiments on `WIKI-103`

We illustrate our findings on the more challenging `WIKI-103` dataset and demonstrate that our L2 fix significantly improves the performance of the LM. In the Table 2, we illustrate that when changing the regularization strength we again see a significant gain in performance when adding our regularization during training of the LM. Due to the computational costs when training these models, we resort to the same hyperparameters as in

	kNN-LM ($\omega = 0$)	kNN-LM ($\omega = 1$)	kNN-LM ($\omega = 10$)
Train Ppl. LM	11.31	11.24	**11.07**
Valid Ppl. LM	18.00	17.95	**17.71**
Valid Ppl. kNN-LM	16.09	**15.89**	17.46

Table 2: Experiments on `WIKI-103`. We report the training and validation perplexities for standard kNN-LM i.e. ($\omega = 0$) as well as our weighted versions. Here we show that our method is much better once we apply kNN

the `WIKI-2` dataset and hence present fair comparisons of the different variants of the model.

Note that again, we see significant improvements in terms of validation perplexity when using the kNN-LM scheme by simply adding an L2 regularization when training the language model.

On another note, when taking a closer look at the validation perplexity before applying kNN, we note that $\omega = 10$ seems to have the lowest validation perplexity. This better generalization phenomenon is interesting and has recently been noted in the machine vision community in the context of investigating the regularization effects of batch normalization in classification settings (Dauphin and Cubuk, 2021). This also relates to the findings of (Merity et al., 2017), those who used L2 regularization in LSTMs. In this paper, we found initial indications that the L2 regularization on the activations might be useful for Transformer models.

Finally, we believe that these two standard benchmark datasets in language modeling are sufficient evidence to demonstrate the merit our of findings. Further studies with more hyperparameters could be done on `WIKI-103`, however, due to computational costs, we leave this for future work.

4.4 Further investigations into the representations and possible explanations

To get a better understanding of why the L2 regularization on the activations seems to improve the performance of kNN-LM, we looked closer at the learned representations for `WIKI-02`.

Effect of the target word frequency on the loss: Figure 2 shows a histogram of word frequency, where the color represents the respective losses each word incurred. More concretely, each bar represents the number of words with a given frequency. For a given histogram bar, we compute the loss for each of these corresponding words. The colors represent the loss i.e. if we have a darker violet color, we incurred a higher loss for these words, and the lighter color the bar the smaller the error. Note that firstly, there is little difference in the loss for the less frequent words (right tail end of the histogram). If we shift our attention to the more frequent words (left side of the histogram) however, we see a different picture. Looking at our L2 regularized model, we note that for the most frequent words, our model seems to incur lower loss (see the brighter colors bars at the peak of the histograms) compared to the standard LM with kNN. This observation suggests that the main differences in terms of representations come from the frequent words rather than rare ones. This is an indication that L2 regularization helps representations cluster and hence when performing the interpolation between the predictions of the LM and kNN, the resulting kNN LM is more confident in these predictions hence leading us to obtain lower perplexities for common words.

Secondly, knowing that the main differences are within the words that are most frequent, we investigated these representations in more detail. In particular, we analyzed the most frequent words and divided the data into "*high loss/score* i.e. loss > -10" meaning they contributed a lot to the loss (bad predictions) and "*low loss/score* i.e. loss < -10" meaning they are good predictions i.e. they contributed a little to the loss.

We employed a simple mixture of Gaussians model (GMM) ($m = 10$) and used the log-likelihood as an indicator for how well the data are clustered. GMMs allow us to put probability mass on each of the representations and given that we are using a mixture of Gaussians, we inherently capture clusters. Intuitively, this means that if the likelihood of the GMM is high, the representations can be easily captured using a mixture of Gaussians, which is indicative of being more clustered i.e. close to one of the gaussian mixture means.

In Figure 3 we compare the distributions of the loglikelihoods for the representations that have been trained using the standard LM and our modified L2 regularization. In particular, for each representation, we obtain the corresponding likelihood from the GMM (x-axis on Figure 3). As mentioned before, we split the words into "*high loss/scores*" and "*low loss/scores*" and plot their histograms in

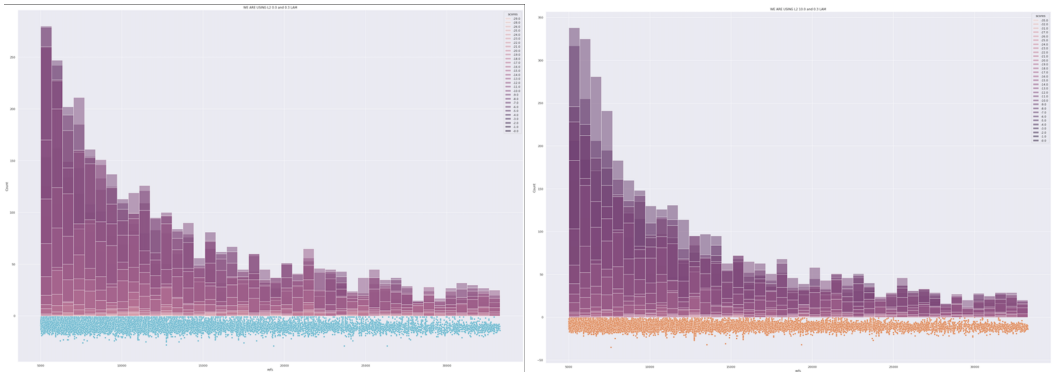

Figure 2: Frequency/Loss histograms. The x-axis denotes the frequency of the word with high-frequency words to the left. The y-axis denotes the number of words with x frequency and the colors of each bar represent the loss accumulated. (LEFT) Standard LM after kNN, (RIGHT) Our L2 regularized LM after kNN.

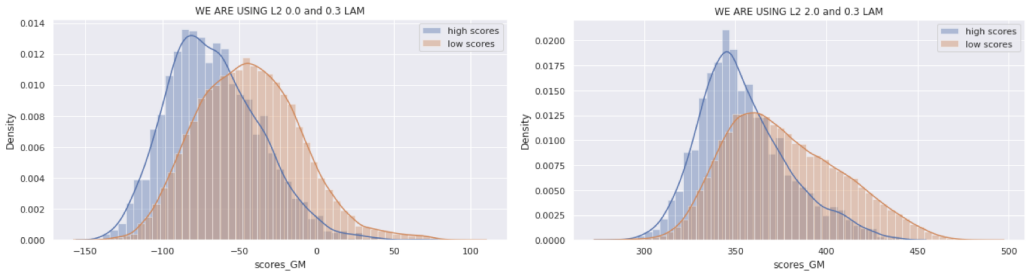

Figure 3: x-axis denotes the Loglikelihood under the Gaussian Mixture. y-axis denotes the normalized histogram. (LEFT) Standard training of the Language Model (RIGHT) using an L2 regularization for the Language Model.

blue and orange respectively. Fig. 3 demonstrates one key finding, which is that the difference between the likelihoods of the "*high loss/scores*" and "*low loss/scores*" varies much more dramatically in the L2 regularized case. Recall that the higher the likelihood, the higher the "clusterness" is. By noting that the likelihood differs much more in the L2 regularized case, we can conclude that the representations in the latter are more clustered (for the low scores) due to the regularization, which could be one potential explanation why kNN LM is improved. Hence, one of our hypotheses is that kNN-LM improves the classification accuracy mostly for the non-frequent words (Khandelwal et al., 2020), whereas our proposed method with L2 regularization, **in addition**, also improves the classification accuracy of the frequent words by clustering them closer together and hence improving kNN-LM.

5 Conclusion

In conclusion, we propose a useful training mechanism that is inspired by the fact that the post-hoc application of kNN seems to significantly improve the performance of standard LMs. We have found that training a LM with L2 regularization at the final layer, i.e. layer which is used for the post-hoc kNN search, improves validation performance. We have also found initial indications that the L2 regularization mostly improves performance for the most frequent, lower-loss words. In addition, we have found further evidence for the hypothesis proposed (Dauphin and Cubuk, 2021) which states that L2 regularization helps generalization in vision tasks. This paper found similar results when working with Transformer models in NLP tasks.

There are, however, some shortcomings in our work. Firstly, we have only given a preliminary explanation for why the added L2 regularization significantly improves upon standard kNN LM, but we believe that we have given sufficient evidence that our proposed method promotes clustering of the representations which subsequently improves the kNN. Secondly, even though we have found great and promising improvement using our findings on `WIKI-2`, further work with more compute should be done on `WIKI-103`. We however leave this for future work due to computational constraints. Lastly, we believe that training models with post-hoc kNN in mind is a promising area and hence future work will consider more diverse datasets from the NLP literature. These findings motivate exploring various regularizations in different Transformer architectures and LM tasks.

References

Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*.

Yann N Dauphin and Ekin D Cubuk. 2021. Deconstructing the regularization of batch-norm. In *International Conference on Learning Representations (ICLR)*.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738.

Armand Joulin, Moustapha Cissé, David Grangier, Hervé Jégou, et al. 2017. Efficient softmax approximation for gpus. In *International Conference on Machine Learning*, pages 1302–1310. PMLR.

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)*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.