Building Personalized Language Models through Language Model Interpolation

Milton King and Paul Cook

University of New Brunswick
{milton.king, paul.cook}@unb.ca

Abstract. Social media users differ in how they write such as writing style and topics. This suggests that personalized language models — language models tailored to a specific person — could outperform a single generic language model. One challenge, however, is that language models typically require a large volume of text to train on, but for many people such a volume of text is not available. In this paper, we train n-gram and neural language models on relatively large in-domain background corpora, and on relatively small amounts of text from individual social media users, specifically authors of blogs. In experiments with interpolated language models, we find that, although user-specific language models trained on a small amount of text from a user perform relatively poorly, they can be interpolated with language models trained on a large background corpus to give improvements over either approach on its own. We further find that n-gram and neural language models are complementary, and can be interpolated to give improvements over either approach used individually. Our evaluation considers perplexity, and two evaluation measures motivated by next word suggestion on smart-phones. We find that although perplexity is widely used for intrinsic evaluation of language models, it is a poor indicator of performance in terms of these other measures.

Keywords: Language models · User-specific language model · Personalization.

1 Introduction

The difference in the writing styles of people on social media platforms, such as blogs, suggests that language models tailored towards individuals can outperform generic language models. However, language models often require a large amount of text to train on, which is often not available for many people. Therefore, we interpolate language models that we trained on a large in-domain corpus with language models that are trained on far less text from a single person by averaging the probabilities from each model. Our evaluation considers perplexity, and two other evaluation measures motivated by next word suggestion on smartphones. We find that although perplexity is widely used for intrinsic evaluation of language models, it is a poor indicator of performance in terms of these other

measures. We show that even though the language model that was trained on a single user performs fairly poorly on its own, it can increase the performance of the generic language model that was trained on a large background corpus. We look at two types of language models, a long short-term memory neural network (LSTM) and an n-gram language model which uses Kneser-Ney smoothing [3] and found that using an LSTM as a background model outperformed all other models by themselves. However, we achieve better performance when interpolating an LSTM-based background model with a user-level model that was trained on only 1000 tokens for two of our three evaluation metrics. Furthermore, we show that the type of model used as the the background model largely affects the performance of the interpolated models. Nevertheless, we find that both types of background language models achieve better results when interpolated with a user-level language model. We also look at the effects that the volume of text from a user has on the language models' performance.

2 Related work

Personalizing language models can be viewed as a domain adaptation problem, which often involves models that train on a large out-of-domain corpus and then adapted to a specific domain. Such approaches for language modeling often involve a recurrent neural network at its base with a method to perform the adaptation such as [11], who used a topic distribution vector fed into the hidden and output layers of their network. Another common way to achieve domain adaptation is to initially train on a background corpus and then continue training on text from the target domain [8]. A model can also leverage metadata about the author such as age, gender, and personality, to assist in personalized NLP tasks [9]. [7] used metadata about the author as well, along with two LSTM models — with one model being used to model an author and another model being used to model how the author's text changes given a specific addressee. Although metadata can be helpful, it is often not available and therefore we only consider text from authors in our following experiments.

[5] achieved domain adaptation, with the domains being categories of YouTube videos, by combining multiple LSTMs, where each LSTM represents a specific domain and are then combined with a "mixer" LSTM that determines how much weight to give to each domain-specific LSTM. Similarly, we combine the output of multiple language models with the difference that we average our outputs across all language models instead of using an LSTM to combine them and are using text from individual users. Language models are also commonly used in an extrinsic evaluation such as text classification. This includes domain-adapted language models in sentiment analysis, question classification, and topic classification [4], or determining if a tweet is relevant to a natural disaster [2].

There have been many different domains that have been approached with domain adaptation techniques such as Youtube speech recognition [5] and newspaper sections [6]. Our work differs from many of the previous works because our background corpus and test set are within the same domain and therefore,

we are adapting to the user's writing style and not the domain itself. This is a more subtle change, where the only thing that differs is the authors themselves.

3 Data and evaluation

3.1 Dataset

The dataset that we use contains blogs and was used in [14], which contains 19,320 users. We use the sentence splitter from [10] to generate one sentence per line and prepended each sentence with a start-of-sentence token and append each sentence with an end-of-sentence token. We hold out the 10 users with the most tokens and divide their text into *USER* — a set of 10 user-specific corpora, which will be used to train our user-level models — and *TEST*, which will be our test set. The partition is approximately a 25/75 split. *TEST* consists of a 38,219 sentences, containing a total of 765,614 tokens with the number of tokens from a single author ranging from 52,633 to 112,049. We hold out the next 20 users with the most tokens for future analysis. We take a maximum of 30,000 tokens (not including start and end-of-sentence tokens) from each of the other 19,290 user to form our background corpus *BACKGROUND*. The 30,000 threshold is implemented to avoid biases and promotes a more evenly distributed corpus across multiple users and is based on the token count for each user. We further modify *BACKGROUND* by replacing all words that occur less that 10 times with the unknown token *unk*. This gives us a background corpus of 6,668,281 sentences containing 129,549,606 tokens (including start and end-of-sentence tokens) with a vocabulary of 92,578 types. The number of sentences from any single user ranges from 1 to 4024 and number of tokens from a user ranging from 112 to 38053, including start and end-of-sentence tokens. All corpora have numerals replaced with a *num* token and were casefolded and tokenized by Stanford Core NLP [10].

3.2 Evaluation

One of the most common language models used in language modelling is an LSTM due to its often superior performance [15] and therefore the two types of language models that we look at include an LSTM implemented in Pytorch[1] and an n-gram model that uses Kneser-Ney smoothing known as Kenlm [3]. We trained each type of language model exclusively on either a single user's corpus from *USER* or on the collection of blog posts that we call *BACKGROUND* and will be denoted as *language_model-corpus* from hereon. For example, an LSTM trained on *USER* would be *LSTM-USER*. The language models are trained across sentence boundaries. The language models are tested on *TEST*, with the models trained on *USER* only testing on their corresponding test sentences from the same user.

[1] Implementation is based off the code from `https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/02-intermediate/language_model/main.py#L30-L50`

3.3 Evaluation metrics

There are a few common metrics that are used to evaluate the performance of a language model and some may be preferred over others depending on the target application for the language model. We use three different evaluation metrics in order to achieve a better understanding of how our models compare. In this section, we discuss the evaluation metrics that we use to evaluate our models.

Adjusted perplexity Perplexity is defined below in Equation 1

$$\text{perp} = -\frac{1}{N}\sum_{i=1}^{N}\log(\text{p}(\text{w}_i)) \tag{1}$$

with N being the size of the vocabulary. It is one of the most common intrinsic evaluation metrics used to evaluate language models that are trained on the same corpus, but does not give a fair comparison of language models that have different vocabulary. This is due to the fact that language models can artificially inflate their score by decreasing their vocabulary size and predict most words as *unk*. Adjusted perplexity was proposed by [12], which penalizes a language model that predicts *unk* and therefore allows us to fairly compare models that were trained on different corpora. The overall perplexity calculation does not change, but the probability of *unk* is calculated using Equation 2, defined below as

$$\text{p(unk)} = \frac{\text{p(unk)}}{|\text{UNK-TYPES}|} \tag{2}$$

with $UNK - TYPES$ being the set of types that are converted to unk in the test file. During our evaluation, the start-of-sentence type is never predicted as the target word but the LSTMs generate probabilities for all words in its vocabulary, including the start-of-sentence type and therefore we remove the probability given to it before applying our softmax function making the size of our output layer $v - 1$, where v is the size of the vocabulary. The *n-gram* models do not give a probability for the start-of-the-sentence type as a default. All models are still given the start-of-sentence token as the starting token. The models are expected to predict the end-of-sentence token for this metric.

Accuracy at k Accuracy at k is a metric that is closely associated with the down stream application of next word prediction on smart-phones where many soft keyboards provide three suggestions for the next word. Accuracy at k captures the number of times that a model predicts the target word in the top k words of its vocabulary. We look at k in a range from 1 to 5. Unlike adjusted perplexity, we do not evaluate a model on their ability to predict the end-of-sentence token. This is to reflect the use of word suggestion on smart-phones where the end-of-sentence is not useful to the user. The probabilities for *unk* and end-of-sentence are set to 0 because these will never be the target word.

Accuracy at k given c keystrokes We take the accuracy at k one step closer to how many smart-phones perform next word prediction by allowing the language model to look at the first c characters of the target word. This is similar to the task of query completion on the character level, which was approached by [6] and [13]. This evaluation metric simulates that the user types the first c characters of the target word. We set k to 3 because this is common for most smart-phones and look at c on a range from 0 to 3. If c is equal to or larger than the target word then we say the model predicted the word with a probability of 1 for that c. Similar to accuracy at k, we do not evaluate the model's ability to predict the end-of-sentence token. Again, the probabilities for *unk* and end-of-sentence are set to 0 because we never expect them to be the target word in a real-world environment, such as on a smart-phone.

4 Experimental results

In this section we first tune our LSTM models that are trained on *user* text. We then evaluate our models using our three evaluation metrics. We look at the models' performances by themselves — allowing *user* models to train on different volumes of text from a single user — followed by interpolations involving two models. The *user* models that we use for interpolations are trained on 1000 tokens. We then discuss the performance of interpolations involving three and four models.

4.1 Tuning user-trained language models

We use text from five random users from the *BACKGROUND* corpus to tune our parameters for our LSTM that is trained on a user via grid search. Each model is only given 1000 tokens from a single user. For example, we will have five models with the same parameters, but each one will be trained on text from a different user. The parameters for the LSTMs that we tune are number_of_layers $(1, 2)$; number_of_hidden_units $(128, 256, 512, 1024, 2048)$; embeddings_size $(64, 128, 512, 1024)$; number_training_epochs $(1, 2, 3)$; batch_size $(1, 2, 5, 15, 30, 454)$. Our final user-level LSTM models are single layer with 256 hidden units, an embedding size of 256, trained using a batch size of 2 for 1 epoch.

Our LSTM that was trained on *BACKGROUND* uses the default parameters of an embeddings size of 128, 1024 hidden units, 1 layer, a batch size of 45, and 1 training epoch. They were trained using a cross entropy loss function. We applied the same final parameters that we use for *LSTM-user*, but preliminary experiments showed that the default values achieved better results. Our n-gram model uses the trigram implementation of the KenLM model.

4.2 Impact of volume of data on user-level language models

In this section, we observe the effects of allowing the language models to have access to a larger amount of training data from a user. We randomly select sets

containing approximately 1,000; 10,000; 100,000; and 200,000 tokens from each user from *USER*. We train each type of model on each of these sets — with each model only training on text from one user — and evaluate them on *TEST*.

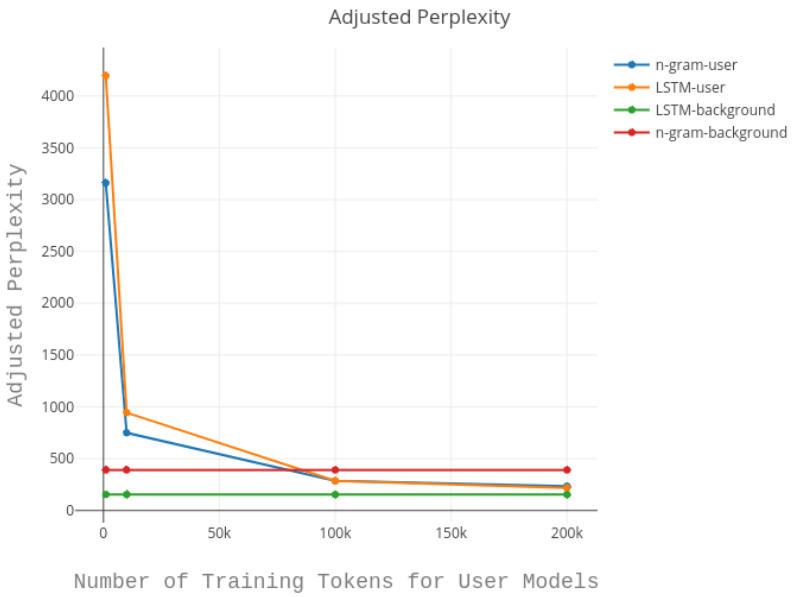

Fig. 1. Adjusted perplexity for different amounts of training text (lower is better).

We show the adjusted perplexity for each language model and the effects that the amount of training text has on our *user* models in Figure 1. The amount of training text for *LSTM-background* and *n-gram-background* does not change. We see that both *n-gram-user* and *LSTM-user* perform relatively poorly when given less than $100k$ tokens to train on but quickly achieve a better score than *n-gram-background* around $100k$ tokens and approaches *LSTM-background* at $200k$ tokens. The fact that the models achieve similar score with far less training text shows the importance of training on text from the user and not a generic corpus even when generic text is within domain.

Next, we compare the accuracy at k for the four different models. Again, *LSTM-background* achieves the highest score. We also see, that the *user* models outperform *n-gram-background* with only $10k$ tokens instead of the roughly $100k$ tokens needed for adjusted perplexity. The findings for accuracy at k show that *user* models do not compete with large models trained on in-domain background corpora, which is not in line with the findings when evaluating with adjusted perplexity.

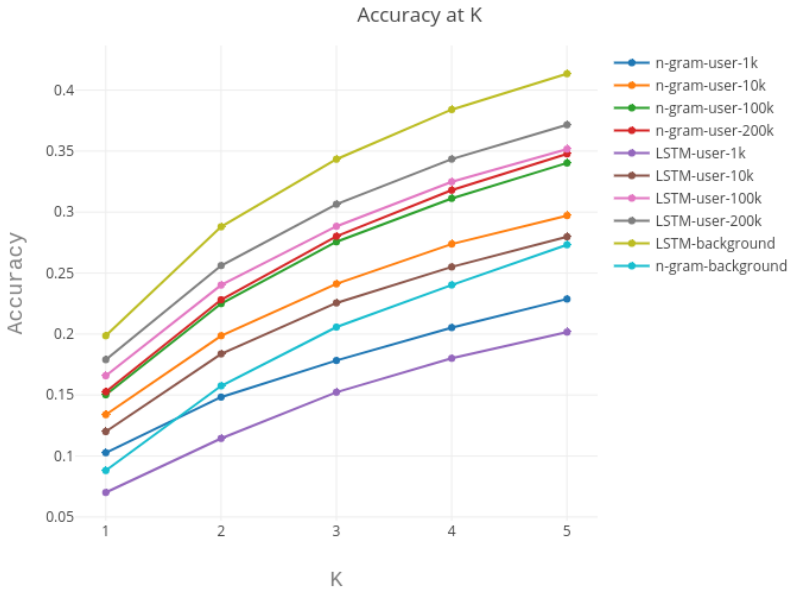

Fig. 2. Accuracy at k for different amounts of training text with k as the x-axis (higher is better).

4.3 Interpolation

In this section, we show the performance of our interpolated models using the different evaluation metrics. Since we are interested in users with a small amount of text, our *user* models that are involved with interpolations are trained on only 1000 tokens from a single user. As discussed in Section 3.2, we interpolate our models by averaging their probabilities.

We first look at the effects that two-way interpolation has on adjusted perplexity shown in Figure 3. We see that any interpolation involving a *background* model outperforms *n-gram-background* without interpolation. This result is interesting since both *user* models performed relatively poorly but are still able to assist the *n-gram-background* model. Furthermore, each *background* model improves its performance when interpolated with either a *user* model or another *background* model of a different type, with *LSTM-background* and *n-gram-background* achieving the best score. This supports the findings of [1], who found that neural language models and n-gram language models are complimentary. Although, *LSTM-user* interpolated *n-gram-user* does not outperform *n-gram-user* by itself.

Next, we look at interpolating models with respect to accuracy at k shown in Figure 4. For this metric, language models generate probabilities for each word in its vocabulary, but we only look at the top 50 probabilities — to reduce the computational cost. If a word that is present in the top 50 of one model but not

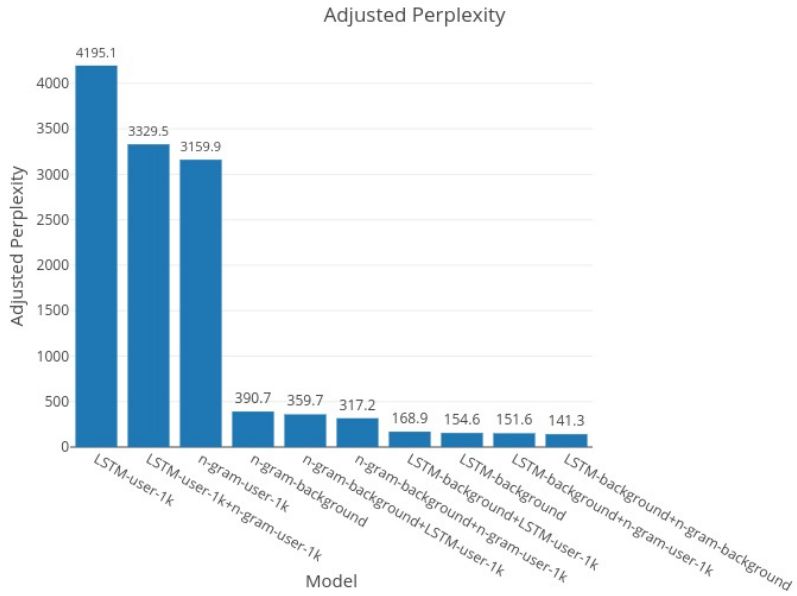

Fig. 3. Evaluating interpolated models using adjusted perplexity (lower is better).

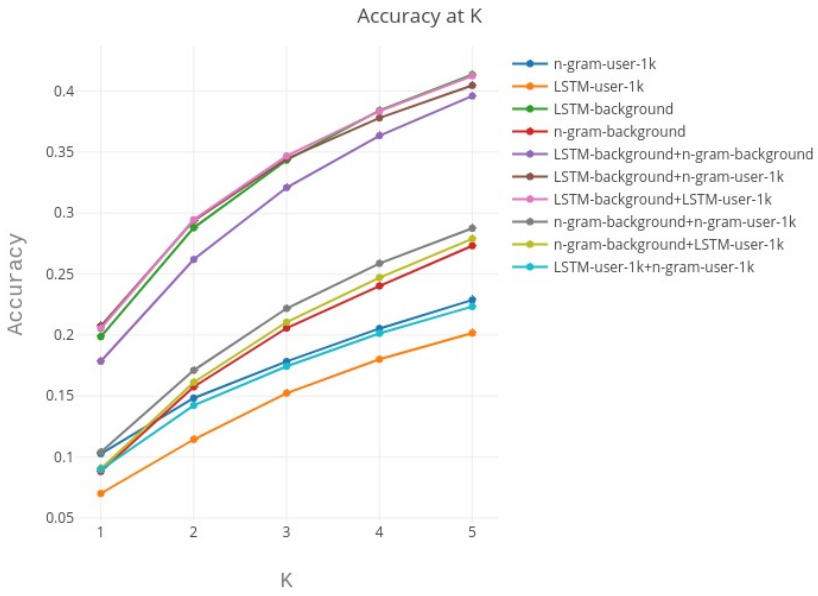

Fig. 4. Evaluating accuracy at k for interpolated models (higher is better).

in another, then the probability is set to 0 for the model that does not contain the word and the interpolation is performed the same as before. Here we see that any interpolation that involves a *background* model outperforms either model by itself, which contradicts the findings from adjusted perplexity in Figure 3. Furthermore, *LSTM-background* interpolated with *n-gram-background* achieves the best performance in terms of adjusted perplexity, but is outperformed by *LSTM-background* by itself or when it is interpolated with either *user* model when using the accuracy at k evaluation. We achieve our highest scores with *LSTM-background* interpolated with either *user* model. Interestingly, the two *user* models are not complimentary when evaluating with accuracy at k, which supports the findings when evaluating with adjusted perplexity.

4.4 Accuracy at 3 given c keystrokes evaluation

In this section, we evaluate our models using accuracy at 3 given c keystrokes, which was explained in Section 3.3. We selected 3 because it is common for smart-phones to suggest up to 3 words. Given c keystrokes, we normalize the model's probabilities across all word types that begin with the c keystrokes. Similar to accuracy at k, during testing we average the top 50 probabilities for any given c keystrokes with c being on a range from 0 to 3, inclusive. Again, we will first look at the models by themselves and then the interpolations.

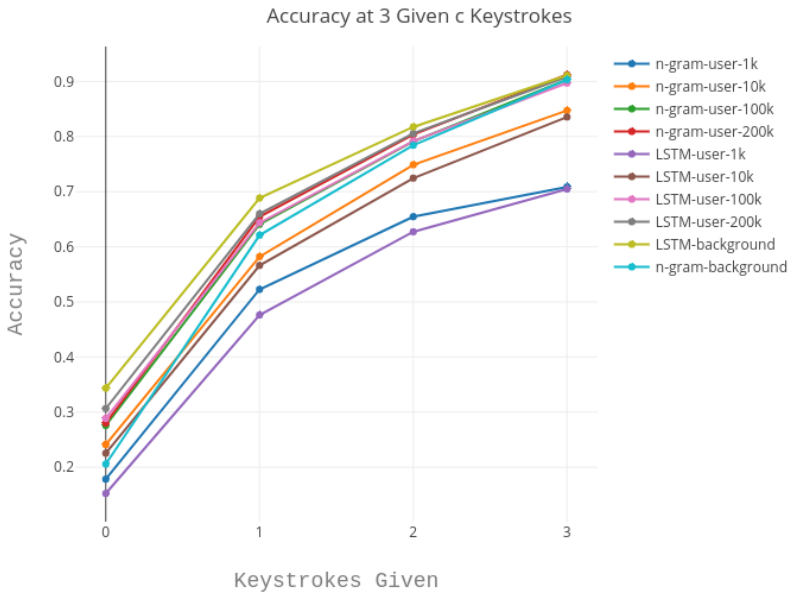

Fig. 5. Evaluating models using accuracy at 3 given c keystrokes (higher is better).

Figure 5 shows accuracy at 3 given c for models without interpolation while allowing *user* models to train on different amounts of text from a single user. We see similar findings to accuracy at k in Figure 2, where neither model by itself beats *LSTM-background* and either *user* model trained on at least $10k$ tokens outperforms *n-gram-background*.

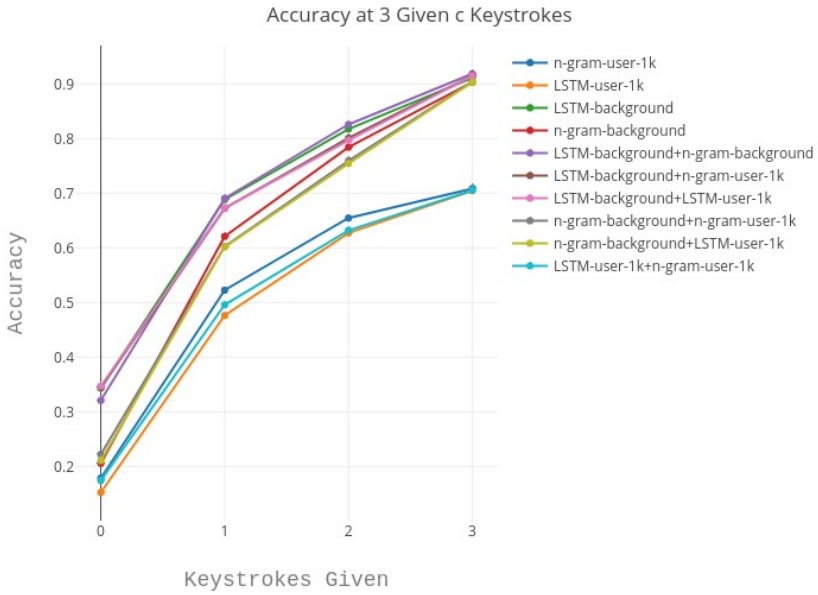

Fig. 6. Evaluating interpolated models using accuracy at 3 given c keystrokes (higher is better).

Next, we evaluate our interpolated models with respect to accuracy at 3 given c keystrokes shown in Figure 6. It shows that the only model to outperform *LSTM-background* by itself is *LSTM-background* interpolated with *n-gram-background* for $c \geq 1$. This finding is different than the findings shown when evaluating with accuracy at k where *LSTM-background* interpolated with either *user* model outperforms *LSTM-background* interpolated with *n-gram-background*.

Three-way and four-way interpolations In an attempt to better our results using our models that were trained on 1000 tokens from a user, we perform interpolations involving three and four models. We consider all possible three-way interpolations. However, none of the three-way or four-way interpolations outperform our previously best models for all our metrics, which were *LSTM-background* interpolated with *n-gram-background* for both adjusted perplexity and accuracy at 3 given c keystrokes, and *LSTM-background* interpolated with either *user* model for accuracy at k.

5 Conclusions

In this work, we looked at interpolating language models trained on an in-domain background corpus with language models trained on a small amount of text from a single user. We showed that we can outperform a large background model by interpolating it with a language model that was trained on as little as 1000 tokens from a user in terms of adjusted perplexity and accuracy at k. However, interpolating two models trained on two large background corpora achieved our best results for adjusted perplexity and accuracy at 3 given c keystrokes. We showed that adjusted perplexity does not reflect the performance of a language model's ability to perform next-word suggestion and does not give the same findings as both accuracy at k and accuracy at k given c keystrokes. For example, *LSTM-background* interpolated with *n-gram-background* outperforms all other models in terms of adjusted perplexity, but is outperformed by both *LSTM-background* by itself and *LSTM-background* interpolated with a *user* model in terms of accuracy at k. Our results show that under all metrics, interpolating either *background* model with a model trained on text from a user will outperform either model by themselves except in a few cases. We also found that the type of the language model used for the background corpus has a large impact on the final results, with models including *LSTM-background* outperforming models that include *n-gram-background*.

In future work, we would like to apply a weighted interpolation that determines how much input either model contributes to a final prediction. Also, our neural language models were trained using a cross entropy loss which is designed for a model to favour a lower perplexity, but we would like to train the neural language models with a loss function that takes into account the ranking — making it more suitable for accuracy at k and accuracy at k given c keystrokes evaluation metrics.

References

1. Adams, O., Makarucha, A., Neubig, G., Bird, S., Cohn, T.: Cross-lingual word embeddings for low-resource language modeling. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. pp. 937–947. Association for Computational Linguistics (2017), http://aclweb.org/anthology/E17-1088
2. Alam, F., Joty, S., Imran, M.: Domain adaptation with adversarial training and graph embeddings. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1077–1087. Association for Computational Linguistics (2018), http://aclweb.org/anthology/P18-1099
3. Heafield, K., Pouzyrevsky, I., Clark, J.H., Koehn, P.: Scalable modified Kneser-Ney language model estimation. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics. pp. 690–696. Sofia, Bulgaria (August 2013), https://kheafield.com/papers/edinburgh/estimate_paper.pdf
4. Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. In: Proceedings of the 56th Annual Meeting of the Association for Computational

Linguistics (Volume 1: Long Papers). pp. 328–339. Association for Computational Linguistics (2018), http://aclweb.org/anthology/P18-1031
5. Irie, K., Kumar, S., Nirschl, M., Liao, H.: Radmm: Recurrent adaptive mixture model with applications to domain robust language modeling. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) pp. 6079–6083 (2018)
6. Jaech, A., Ostendorf, M.: Personalized language model for query auto-completion. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 700–705. Association for Computational Linguistics (2018), http://aclweb.org/anthology/P18-2111
7. Li, J., Galley, M., Brockett, C., Spithourakis, G., Gao, J., Dolan, B.: A persona-based neural conversation model. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 994–1003. Association for Computational Linguistics (2016). https://doi.org/10.18653/v1/P16-1094, http://aclweb.org/anthology/P16-1094
8. Lin, Z.W., Sung, T.W., Lee, H.y., Lee, L.S.: Personalized word representations carrying personalized semantics learned from social network posts. pp. 533–540 (12 2017). https://doi.org/10.1109/ASRU.2017.8268982
9. Lynn, V., Son, Y., Kulkarni, V., Balasubramanian, N., Schwartz, H.A.: Human centered nlp with user-factor adaptation. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 1157–1166. Association for Computational Linguistics (2017), http://www.aclweb.org/anthology/D17-1120
10. Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. pp. 55–60. Baltimore, USA (2014)
11. Mikolov, T., Zweig, G.: Context dependent recurrent neural network language model. 2012 IEEE Spoken Language Technology Workshop (SLT) pp. 234–239 (2012)
12. P. Ueberla, J.: Analyzing and improving statistical language models for speech recognition (07 1994)
13. Park, D.H., Chiba, R.: A neural language model for query auto-completion. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1189–1192. SIGIR '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3077136.3080758, http://doi.acm.org/10.1145/3077136.3080758
14. Schler, J., Koppel, M., Argamon, S., Pennebaker, J.W.: Effects of age and gender on blogging. In: AAAI spring symposium: Computational approaches to analyzing weblogs. vol. 6, pp. 199–205 (2006)
15. Sundermeyer, M., Schlüter, R., Ney, H.: Lstm neural networks for language modeling. In: INTERSPEECH (2012)