Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 2461–2469
Marseille, 11–16 May 2020

Evaluating Approaches to Personalizing Language Models

Milton King, Paul Cook
University of New Brunswick
Fredericton, New Brunswick, Canada
milton.king@unb.ca, paul.cook@unb.ca

Abstract

In this work, we consider the problem of personalizing language models, that is, building language models that are tailored to the writing style of an individual. Because training language models requires a large amount of text, and individuals do not necessarily possess a large corpus of their writing that could be used for training, approaches to personalizing language models must be able to rely on only a small amount of text from any one user. In this work, we compare three approaches to personalizing a language model that was trained on a large background corpus using a relatively small amount of text from an individual user. We evaluate these approaches using perplexity, as well as two measures based on next word prediction for smartphone soft keyboards. Our results show that when only a small amount of user-specific text is available, an approach based on priming gives the most improvement, while when larger amounts of user-specific text are available, an approach based on language model interpolation performs best. We carry out further experiments to show that these approaches to personalization outperform language model adaptation based on demographic factors.

Keywords: Language modelling, Personalization, Domain adaptation

1. Introduction

In this work, we consider the problem of personalizing language models, that is, building language models that are tailored to the writing style of an individual. Language modelling requires a large amount of text to sufficiently train a language model. However, a large amount of text from any one user — for example social media users such as bloggers — is not necessarily available for training a user-specific language model. To overcome this issue, some techniques treat text from other users that share demographic characteristics, such as age or gender, as training data. Although this has been shown to improve the performance of language models (Lynn et al., 2017), this demographic information is typically obtained via document metadata, which is not available for all document types, or for all users. This suggests a need to personalize language models while only relying on a small amount of text from the user.

We propose and evaluate three different personalization techniques, while limiting the models' access to different amounts of text from the user. These techniques are applied to a language model that was trained on a large background corpus of in-domain text (specifically blog posts). The three techniques include continuing to train the background model on text from the user, interpolating the background model with a model that was trained on the user's text, and priming the state of the background model with text from the user. Our results show that we are able to improve the performance of a background long short-term memory (LSTM) neural network language model using any of the personalization techniques. The interpolated models perform best when a relatively large amount of text is available from the user, however, priming performs best when only a small amount of user text is available. We further apply the interpolation technique to an n-gram background language model and show that it improves its performance, similarly to how it improved the LSTM language model.

Furthermore, we benchmark our personalized models against models that use the same techniques, but that use text from users with the same demographic characteristics as the target user — specifically age and gender — or text from randomly-selected users, for model adaptation. Our findings indicate that all three of our proposed approaches to personalization out-perform model adaptation based on demographics, and that these improvements are not the result of the language model simply having access to more data.

2. Related Work

Language models are important components of systems for many downstream NLP tasks, such as machine translation (Och and Ney, 2004), speech recognition (Jelinek, 1976), handwriting recognition (Marti and Bunke, 2001), parsing (Choe and Charniak, 2016), and classification (Howard and Ruder, 2018). It is reasonable to expect that better performing language models for individual people will result in better performing systems for these downstream tasks for individual people. For example, a non-personalized language model might suggest words in speech recognition that would be correct for many people, but that have rarely been used by the user of the application. An ideal personalized language model would generate probabilities for words that the person is more likely to use, rather then what the majority of speakers would use.

In this work, we use both n-gram language models and neural language models. Neural language models have been shown to be superior to n-gram language models (Mikolov et al., 2010) but often require more text to train and require more computation time. Our neural language model is an LSTM neural network, which can handle an arbitrary length of text as input and can learn how much should be remembered from previous states (Sundermeyer et al., 2012). Prior research has increased the size of the training corpus for a user by including text from other users based on demographics. The assumption being made is that people of similar demographics tend to write similarly. Lynn et al. (2017) used metadata such as age, gender, and personality to adapt their models for tasks such as sentiment analysis and sarcasm detection. Such metadata can be useful (Lynn et al., 2017), although it is not always available due to a limitation of the collected data, or privacy concerns of the user. For example, the user's metadata may not have been recorded, or the user may not be willing to share this information. Also, in the case that a model can access metadata from a user, there is the possibility that the model does not have text from other users in this demographic. In this paper, we compare model personalization using text from a specific user, with model adaptation based on text from other users of the same demographic.

Language modeling has been applied to a variety of domains including text from Wikipedia (Merity et al., 2016), children's books (Hill et al., 2015), and newswire text (Mikolov et al., 2010). Personalizing language models can be seen as a domain adaptation problem where a language model that is trained on a large background corpus is adapted to text from a specific domain, which can be text from a single person. Domain adaptation has been applied for adapting one domain to another, such as adapting between newspaper sections (Jaech and Ostendorf, 2018) or YouTube video categories (Irie et al., 2018). Downstream applications that have benefited from domain adaptation include classifying the relevancy of tweets for a natural disaster (Alam et al., 2018), sentiment analysis, question classification, and topic classification (Howard and Ruder, 2018). Alam et al. (2018) adapted their model from one disaster to another with a series of neural networks and classifiers. Irie et al. (2018) associated each YouTube category with its own LSTM and connected each one using a 'mixer' LSTM, which generated weights for each YouTube category. Mikolov and Zweig (2012) adapted a recurrent neural network by using a vector of a distribution over topics as input to both the hidden layer and output layer. Lin et al. (2017) do not modify the neural network but allow the language model that was trained on the source domain to continue training on the target domain to form personalized word embeddings. The amount of text that they used for training ranged from 1,959 tokens per user to 32,690 tokens per user. We apply a similar technique, with the amount of text per user ranging from 100, to over 100k, tokens.

Radford et al. (2019) conditioned a transformer for text generation by priming it by exposing it to an input text, and then using the state of the transformer for text generation. Here we personalize a language model by priming an LSTM by updating its state while exposing it to text from a user.

3. Data and Evaluation

3.1. Dataset

The dataset that was used consists of blog posts from 19,320 users. It was originally used in Schler et al. (2006). We perform sentence splitting on each document, and add start-of-sentence and end-of-sentence tokens to each sentence.

We select the 10 users who have the largest amount of text to form our set of users that we use for evaluation, which we label *USER*. Our test users consist of 6 males and 4 females with ages ranging from 14 to 47 years old with a mean of approximately 28 years. We split up the text from each user in *USER* into a training set and a held out set, without splitting up blog posts. We do not split up blogs to ensure that we do not have parts of the same blog post in the training and testing data. We randomly select sentences from the training set so that each user has 6 corpora with the following numbers of tokens: 100, 500, 1000, 10k, 100k, and all of the text available for that user. We refer to these corpora as *USER-TRAIN*. The held out set consists of approximately 25% of the user's text, from which we randomly extract sentences until we have at least 10k tokens from each user. We refer to this test set as *USER-TEST*. We use *USER-TRAIN* to build personalized language models, and test them on *USER-TEST*.

We use the 20 users with the largest amount of text after the top 10 to form our random samples, where we randomly extract samples of text of the same sizes as *USER-TRAIN* from this group to form *RANDOM*.

The text from the remaining 19,290 users is used to build our background corpus (*BACKGROUND*). The text from any one user is limited to 30k tokens to avoid text from one user saturating the corpus and biasing the corpus to strongly represent their text. We replace each type in *BACKGROUND* that has a frequency less than 10 with a special *UNK* token. We do this to reduce the cost of training language models. The resulting corpus contains approximately 116M tokens and 93k types.

We randomly select 5 users from *RANDOM* for tuning our models. We split up the text into training (*TUNE-TRAIN*) and testing (*TUNE-TEST*), where *TUNE-TRAIN* contains 1000 tokens from each user and *TUNE-TEST* contains the remaining tokens.

We randomly select sentences from non-test users (i.e., users not in the *USER* set) that have a common age and gender with the target user, to generate 5 demographic-based corpora per user in *USER* with different amounts of text to form the set *DEMOGRAPHIC*. The sizes of the corpora for each user are similar to those for (*USER-TRAIN*), i.e., roughly 100, 500, 1000, 10k, and 100k tokens. The number of users that have the same age and gender as the target user ranges from 45 (females aged 47 years) to 2380 (males aged 17 years).

We perform the same preprocessing on all corpora, which involves casefolding, tokenization by the Stanford Core NLP toolkit (Manning et al., 2014), and converting all numerals to a special <*num*> token. The details for each dataset are shown in Table 1.

3.2. Evaluation

We evaluate our models on each of the 10 users from *USER-TEST*. We use the text from *USER-TRAIN* that corresponds to the same user to personalize a language model trained on *BACKGROUND*. For example, we use text from user x in *USER-TRAIN* to personalize our model, which is evaluated on text from user x in *USER-TEST*. We allow our models to use different amounts from *USER-TRAIN*, which includes 100, 500, 1000, 10k, and 100k tokens, and the full amount of the text available. The amount of text available for each user varies, and ranges from approximately 226k to 446k tokens. We test our models at the sentence level,

Name	Definition
BACKGROUND	Blogs from 19,290 users used to train background models
USER-TRAIN	Samples of text of varying sizes from each of 10 different users; models train on tokens from a single user
USER-TEST	10k tokens from each of the 10 users from USER-TRAIN; used for testing models
TUNE-TRAIN	Tokens from each of 5 different users; similar to USER-TRAIN but for tuning models
TUNE-TEST	Varying amounts of text from each of the 5 users in TUNE-TRAIN; used for evaluation during model tuning
RANDOM	Text from 20 users with the next largest amount of text after those in USER-TRAIN and USER-TEST; used as random text for benchmarking
DEMOGRAPHIC	Text from users that share the same age and gender as the target user; contains samples of the same amounts of text as USER-TRAIN

Table 1: Description of datasets.

which means that the language models cannot use information from previous sentences, and that the cell state of the LSTM is reset before each test sentence so that the results are not affected by the ordering of the test sentences.

3.3. Evaluation Metrics

In this section, we describe each of the three evaluation metrics that we use.

3.3.1. Adjusted perplexity

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

Perplexity is defined in Equation 1, where N is the number of tokens in the corpus. It is one of the most common ways to evaluate language models when each model contains the same vocabulary. Perplexity can't be used to compare language models with different vocabularies, because language models with smaller vocabularies replace more tokens with *UNK*, which can artificially increase the performance of the model. For example, a language model that has a very small vocabulary could perform well in terms of perplexity by only predicting *UNK*. Ueberla (1994) proposed an evaluation metric called adjusted perplexity to address this problem. Adjusted perplexity penalizes language models when the target type is not in the vocabulary. The equation for adjusted perplexity is the same as perplexity, with the exception that the probability of *UNK* is recalculated using Equation 2 below:

$$\text{p}(\textit{UNK}) = \frac{\text{p}(\textit{UNK})}{|\textit{UNK-TYPES}|} \qquad (2)$$

with *UNK-TYPES* being the set of types that are converted to *UNK* in the test data.

3.3.2. Accuracy@k

To evaluate our models on a more extrinsic setup, we evaluate using accuracy@k for each token in the test set. This metric says a language model predicted correctly if the next word in the corpus is in the top k words predicted by the model. It reflects the desired behaviour of a next-word suggestion feature, as on many smartphone soft keyboards. We focus on *k=3*, which is common for such systems. We exclude the end-of-sentence marker from this evaluation because it would not be part of a text being typed.

3.4. Accuracy@k Given c Keystrokes

This metric extends accuracy@k by allowing the model to see the first c characters of the target token. If c is greater than or equal to the length of the target token, then the probability for the target token is set to 1. Again, this evaluation metric lends itself to the desired behavior of a language model that is assisting a person who is writing a message, and has typed the first c keystrokes of the next word. Similarly to accuracy@k, we focus on *k=3* and exclude the end-of-sentence marker from this evaluation.

4. Language Models

We explore two different types of language models in our experiments. The first model is an n-gram language model that uses Kneser-Ney smooting (Heafield et al., 2013) with $n = 3$. Preliminary experiments showed that $n = 3$ yielded the lowest adjusted perplexity on *TUNE-TEST*.
The second model is a neural language model that uses an LSTM. We performed preliminary experiments by testing on *TUNE-TEST* to tune our models. We performed a grid search over the following parameters with their associated values: number of hidden units $(128, 256, 512, 1024, 2048)$; number of layers $(1, 2)$; size of embeddings $(64, 128, 256, 512, 1024)$; number of training epochs $(1, 2, 3)$; and batch size $(1, 2, 5, 15, 30, 45)$. We found that the best performing neural language models that were trained on *TUNE-TRAIN* and tested on *TUNE-TEST* contained 256 hidden units, an embedding size of 256, contained 1 layer, and trained for 1 epoch with a batch size of 2. We used the default settings for the neural language model that was trained on the background corpus which was an embeddings size of 128 with 1024 hidden units, and 1 layer, which was trained using a batch size of 45 for 1 training epoch. The batch size for the user-level model is smaller than the model trained on the background corpus, because the training data for each user is relatively small. For all LSTMs, we set the sequence length to 30, the learning rate to 0.002, and use cross entropy as the loss function.

5. Personalization Techniques

We discuss the three different personalization techniques in the following subsections.

5.1. Interpolation

In this approach, we interpolate a language model that was trained on *BACKGROUND* with a language model that was trained on *USER-TRAIN* by calculating the element-wise mean of their probability distributions. If only one of the two language models' vocabularies contains a given type, then the language model that is missing the type gives a probability of 0. Therefore, the interpolated probability for that type would be (p+0)/2, where p is the probability from the language model that contains that type. This is the only personalization technique that is applied to the n-gram model that was trained on *BACKGROUND* because of its relatively poor performance compared to the neural model, discussed in Section 6.1.

5.2. Continue Training

This personalization technique allows our neural language model that was trained on *BACKGROUND* to continue training on *USER-TRAIN*. We do not update the vocabulary of the language model, which means that all out-of-vocabulary words will be replaced with *UNK*. We set the batch size to 2 for the training on *USER-TRAIN* (similar to the models trained only on user-level text) because of the small amount of text that is used for training. All other parameters remain the same.

5.3. Priming

In this setup, we prime a language model that was trained on *BACKGROUND* by inputting tokens from *USER-TRAIN*, which updates the cell state of the language model without changing the weights of the LSTM. We then freeze the state of the language model and perform testing as usual. We reinitialize the state to the frozen primed state before each sentence in the test data.

<Neural/n-gram>_background	Neural or n-gram language model trained on *BACKGROUND*
<Neural/n-gram>_user	Neural or n-gram language model trained on the user's text
<Neural/n-gram>_background+ <neural/n-gram>_user	<Neural/n-gram>_background interpolated with a neural or n-gram language model trained on the user's text
Neural_background+n-gram_background	Neural_background interpolated with n-gram_background
Neural_background_primed_user	Neural_background, primed on the user's text
Neural_background_continue_training_user	Neural_background, with training continued on the user's text

Table 2: Description of language models.

6. Experimental Results

In this section, we show results for our experiments in terms of adjusted perplexity (Section 6.1.), accuracy@k (Section 6.2.), and accuracy@k given c (Section 6.3.). We then compare personalized models against models that use the same techniques, but with text from users that share demographic characteristics with the target user, and with randomly-selected users (Section 6.4.).

We separate the results based on the background model and the evaluation metric. All graphs show results over different amounts of text from the secondary corpus, except when using accuracy@k given c, where we show results for different values of c. The background models are all trained on the entire amount of text from *BACKGROUND*. The models are named using their model type and personalization technique. For example, the neural model that was trained on *BACKGROUND* and then primed on text from the user would be called neural_background_primed_user. A description of the models is shown in Table 2.

6.1. Adjusted Perplexity

In this subsection, we evaluate our personalized models, along with non-personalized background models, using adjusted perplexity, for differing amounts of text from the user.

In Figure 1, we show the adjusted perplexity for models that involve the neural_background model. We see that none of the models are the best performing model for every amount of text from the user. For less than 10k tokens of user-specific training data, neural_background_primed_user achieves the best performance, but above 10k tokens, the models that are interpolated with the user-only models start to perform better. For 100k tokens and more, neural_continue_training starts to outperform neural_background_primed_user but still does not outperform the interpolated models. We also included neural_background interpolated with n-gram_background for comparison, and we see that it outperforms neural_background by itself. Even though n-gram_background, n-gram_user, and neural_user all perform poorly by themselves, they are all able to improve the performance of neural_background when interpolated with it.[1] Neural_background_primed_user achieves a similar adjusted perplexity, regardless of the amount of text from the user. This might be due to how many steps in the past that the state can remember, i.e., the state may largely be unaffected by the more distant tokens used for priming. This phenomenon was explored by Khandelwal et al. (2018).

In Figure 2, we show the adjusted perplexity for models that involve the n-gram_background model. We see that n-gram_background interpolated with either of the user-only models (n-gram_user and neural_user) outperforms n-gram_background by itself when at least 1000 tokens are available. n-gram_background interpolated with n-gram_user outperforms n-gram_background for all amounts of text from the user, and outperforms n-gram_background interpolated with neural_user when less than 100k tokens from the user are used. That neural_background interpolated with neural_user performs better with larger amounts of text is most likely a result of neural models requiring more text to train. This is interesting since n-gram_user outperforms neural_user for all amounts of text.[2]

[1]These three models are not shown due to their large adjusted perplexities. n-gram_user does not outperform neural_background regardless of the amount of text from the user.

[2]User-only models are not shown in the graph due to their poor performance when using less than 10k tokens for training.

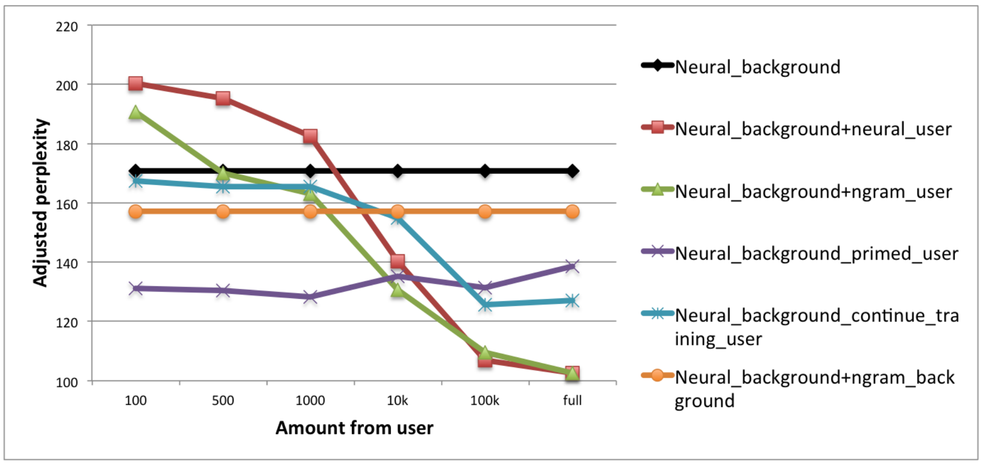

Figure 1: Adjusted perplexity for personalized and non-personalized models that include the neural_background model. (Lower is better.)

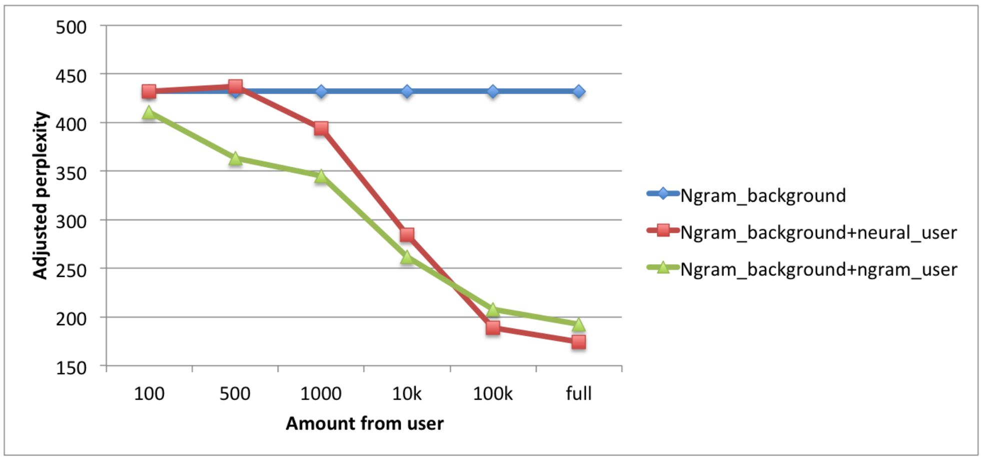

Figure 2: Adjusted perplexity for personalized and non-personalized models that include the n-gram_background model.

Evaluating with adjusted perplexity showed that a background model interpolated with a user-only model performs well when larger amounts of user-specific text are available. In particular, neural_background interpolated with either user-only model outperforms all other models when using 10k tokens or more of the user's text. The priming method achieves the best performance when given fewer than 10k tokens from the user.

6.2. Accuracy@3

In this subsection, we evaluate our personalized models, and non-personalized background models, using accuracy@3 for differing amounts of text from the user.

In Figure 3, we show the accuracy@3 for models that involve neural_background. One of the main differences from adjusted perplexity for the neural models is that now neural_background_primed_user outperforms all models, for each amount of text from the user, except for the case of the full amount of text available. Moreover, neural_background_primed_user with 1000 tokens from the user outperforms all other models, including those that use the full amount of user-specific text. Neural_background interpolated with neural_user now outperforms neural_background interpolated with n-gram_user. The improvement with the use of neural_user over the use of n-gram_user for only accuracy@3, and not adjusted perplexity, might be because of accuracy@3 being a more relative metric than adjusted perplexity. For example, a model needs to assign high probability for the correct type to perform well under adjusted perplexity (which a standard neural model does not do with a small amount of training text), but with accuracy@3, a model only needs to generate a probability for the correct type that is greater than other types in the model's vocabulary. This figure also shows that interpolating the two background models no longer outperforms neural_background by itself. Neural_background_continue_training_user outperforms neural_background when at least 100k tokens from the user are available, as opposed to the adjusted perplexity metric, which showed that only 10k tokens are needed for neural_background_continue_training_user to outperform neural_background.

In Figure 4, we show that n-gram_background interpolated with either user-only model outperforms n-gram_background by itself when the user-only model is trained on at least 500 tokens. There is not a large differ-

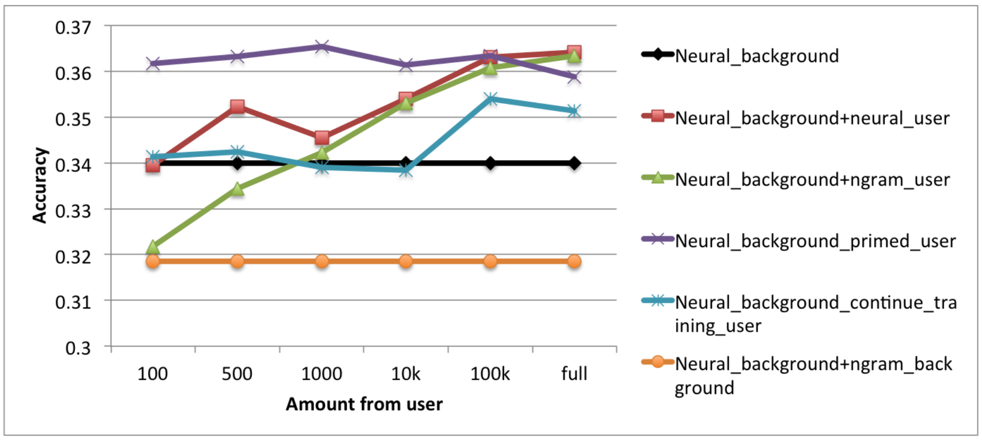

Figure 3: Accuracy@3 for personalized and non-personalized models that include the neural_background model. (Higher is better.)

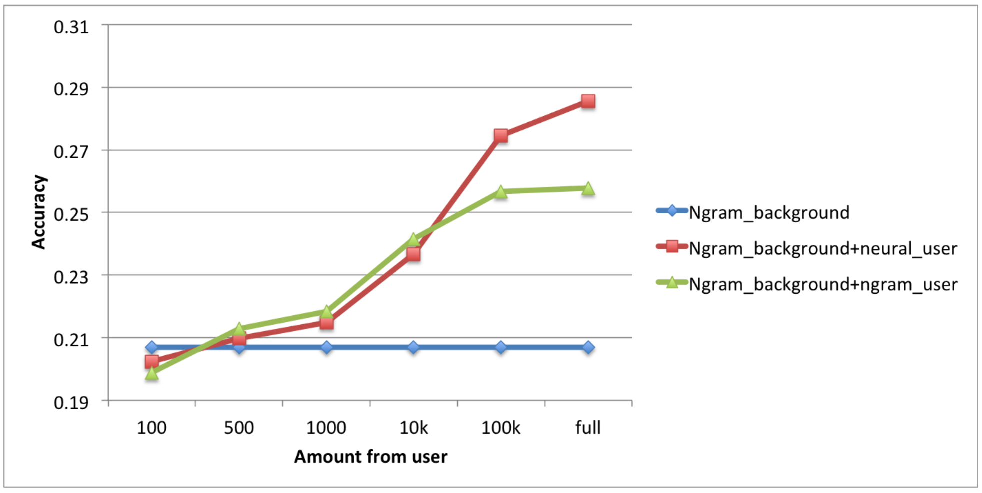

Figure 4: Accuracy@3 for personalized and non-personalized models that include the n-gram_background model.

ence between the two interpolated models until 100k tokens are used, when n-gram_background interpolated with neural_user starts to largely outperform n-gram_background interpolated with n-gram user. This supports the earlier observation of neural models requiring more text for training to outperform n-gram models. Due to models that use neural_background greatly outperforming models that use n-gram_background, we do not further consider n-gram_background.

This evaluation with accuracy@3 showed that priming performs the best, or similar to the best model, for all amounts of user-specific text considered. Moreover, priming with 1000 tokens performed the best of all models considered, even those using orders of magnitude more user-specific training data.

6.3. Accuracy@3 Given c Keystrokes

Figure 5 shows accuracy@3 given c keystrokes for some of our best performing models, along with neural_background. For each approach to personalization, we consider the amount of text from the user that gave its best performance in terms of accuracy@k. Specifically we use 100k tokens for neural_background_continue_training, the full amount of user-level text for neural_background interpolated with neural_user, and 1000 tokens for the priming method.

Given only one character, all of our models achieve approximately 70% accuracy@3. Neural_background_primed_user achieves the best performance when no characters are available to the models, but neural_background interpolated with neural_user achieves the best performance when there is at least one character available. This may be due to the introduction of the user-only model's vocabulary, resulting in non-zero probabilities for more words that were used by the user.

Evaluating with accuracy@3 given c keystrokes, we see that each of the approaches to personalization outperforms the background model, for all values of c, and that the interpolated model performs best when at least one keystroke is given.

6.4. Comparing to RANDOM and DEMOGRAPHIC

In this section, we compare the approaches to personalizing language models with variations of these approaches in

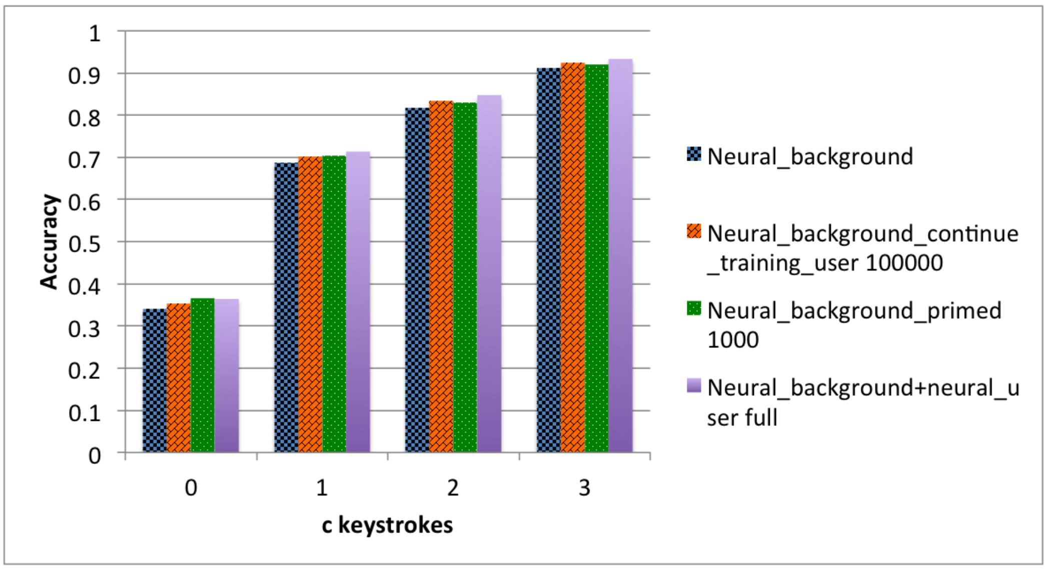

Figure 5: Accuracy@3 given c keystrokes for selected models.

which text from the user (i.e., *USER-TRAIN*) is replaced with text from *DEMOGRAPHIC* or *RANDOM*. We carry out these experiments with *DEMOGRAPHIC* and *RANDOM* to determine whether language model adaptation based on demographic characteristics is as effective as personalization, and whether the improvements in model performance we see with personalization are simply a result of more text being available to the models.

In Figure 6, we show the adjusted perplexity and accuracy@3 for models based on interpolation.[3] Neural_background interpolated with neural_user always outperforms neural_background interpolated with random and neural_background interpolated with neural_demographic.[4]

In Figure 7, we show the performance of models that continue training on a second corpus. Continuing to train on user-specific text always outperforms neural_background by itself, and continuing to train on *RANDOM* or *DEMOGRAPHIC*. We see an unexpected behaviour from the models that continue training on *RANDOM* and *DEMOGRAPHIC*, which might be due to how these corpora were generated. To build *RANDOM* and *DEMOGRAPHIC*, we randomly select sentences, meaning that consecutive sentences in the corpora are most likely not actually consecutive sentences in blog posts. Our models train across sentences using sequences of 30 tokens, and therefore the words seen in a sequence can be "out of place" when trying to predict the target word. This effect was not present in the interpolated models. This might be because the continue training model starts with the neural_background model — which was trained on sentences that were in the correct order — and then continues training on sentences that are not in the correct order. On the other hand, the language models trained on the second corpus for the interpolated models only see sentences that are out of order.

In Figure 8, we show that all primed models outperform the unprimed neural_background model, which suggests that priming on any in-domain text can improve performance. Neural_background primed on user-specific text outperforms all other models, for all amounts of text considered. Neural_background primed on *DEMOGRAPHIC* outperforms neural_background primed on *RANDOM* — showing that model adaptation based on demographics is indeed useful, but less effective than personalization. The issue that was present in models that continue training in Figure 7 for *RANDOM* and *DEMOGRAPHIC* does not appear to affect primed models, which might be due to the state of the LSTM being less affected by words further away from the target word — making words outside the current sentence less likely to affect the state.

These experiments show that language models that are adapted using text from a specific user — i.e., personalized models — outperform models that are adapted based on demographic factors, and models that simply use additional in-domain text for adaptation, for all amounts of text used for model adaptation, and for both evaluation metrics considered.

7. Conclusions

In this work, we showed that a background language model can be improved through personalization for a specific user, while requiring relatively little text from that user. We considered three different personalization techniques, and three evaluation metrics. We showed that for adjusted perplexity and accuracy@3, each personalization technique can improve over the background model, with interpolated models performing better when a relatively large amount of text is available from the user, while priming performs better when a relatively small amount of text from the user is available. We further showed that priming outperforms all

[3]For experiments in this subsection, the largest corpus size we consider is 100k tokens. Using all text available for *RANDOM* would give a much larger corpus than for any individual user in *USER-TRAIN*, and for *DEMOGRAPHIC* would give corpora of widely varying sizes.

[4]Neural_demographic was trained using a batch size of 45 in anticipation of training on a relatively large amount of text

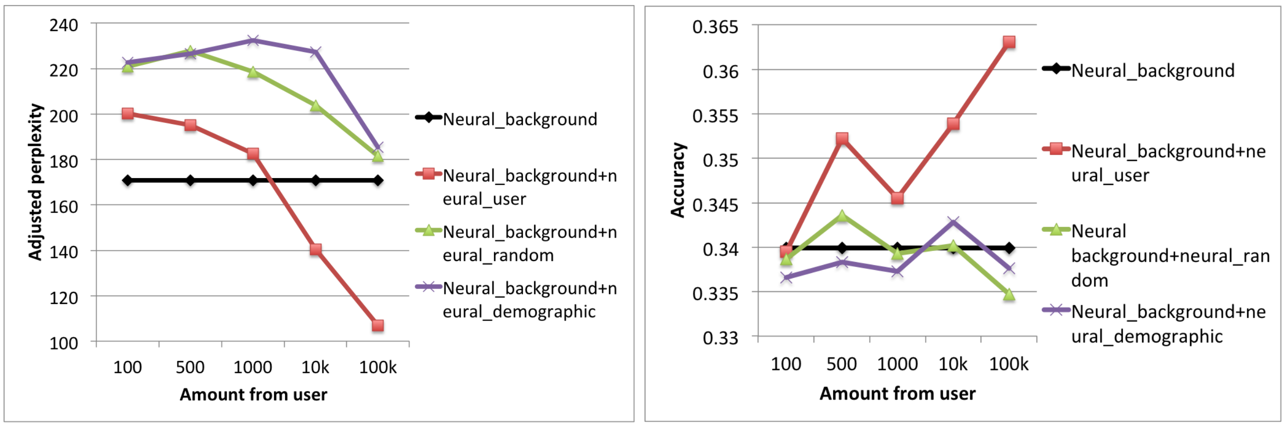

Figure 6: Adjusted perplexity (left) and accuracy@3 (right) for neural_background and interpolated models.

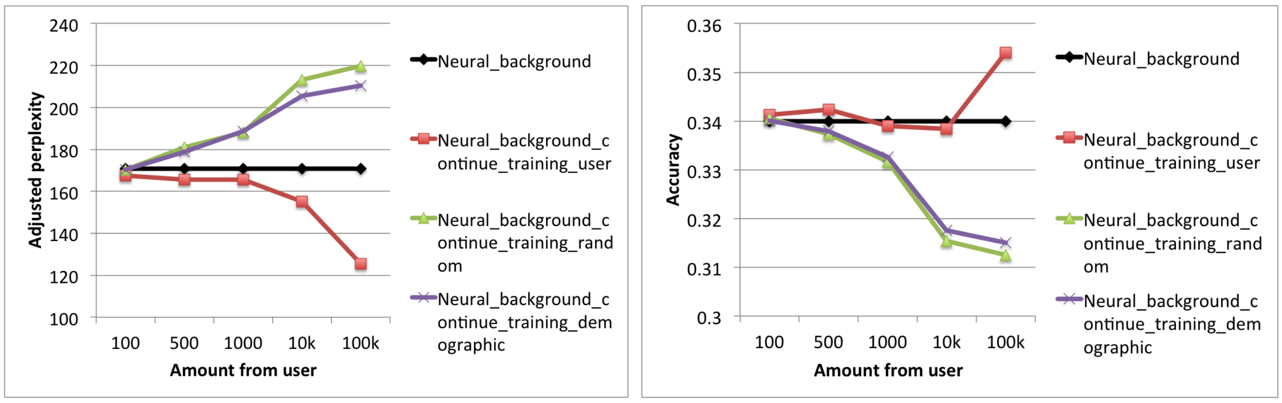

Figure 7: Adjusted perplexity (left) and accuracy@3 (right) for neural_background and models that continue training on a second corpus.

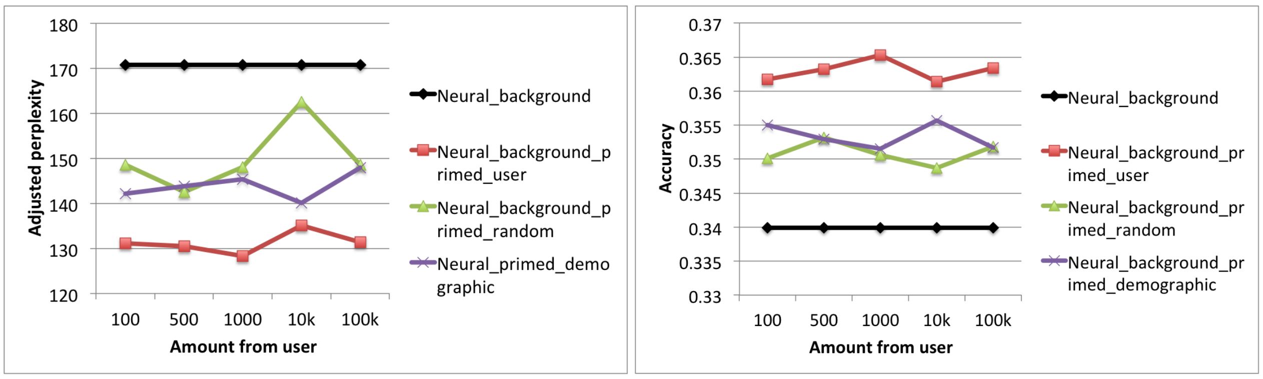

Figure 8: Adjusted perplexity (left) and accuracy@3 (right) for neural_background and neural_background primed on a second corpus.

other models, even those which use orders of magnitude more user-level text for personalization, for accuracy@3. For accuracy@3 given c keystrokes, we showed that the best performing models get a large increase in performance — from approximately 0.35 to 0.7 — when at least one keystroke is given. Finally, we showed the importance of personalization over model adaptation based on demographic factors. For each personalization technique, using text from a specific user for model adaptation outperforms using text from users with similar demographic characteristics. In future work we plan to further explore interpolation, specifically by tuning the interpolation weight, which currently gives equal weight to the background and user-specific language models. We further intend to consider transformers for personalization, which have shown state-of-the-art language modelling results (Radford et al., 2019).

8. Bibliographical References

Alam, F., Joty, S., and Imran, M. (2018). 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)*, pages 1077–1087, Melbourne, Australia.

Choe, D. K. and Charniak, E. (2016). Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria.

Hill, F., Bordes, A., Chopra, S., and Weston, J. (2015). The goldilocks principle: Reading children's books with explicit memory representations. *CoRR*, abs/1511.02301.

Howard, J. and Ruder, S. (2018). 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)*, pages 328–339, Melbourne, Australia.

Irie, K., Kumar, S., Nirschl, M., and Liao, H. (2018). Radmm: Recurrent adaptive mixture model with applications to domain robust language modeling. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6079–6083.

Jaech, A. and Ostendorf, M. (2018). Personalized language model for query auto-completion. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 700–705, Melbourne, Australia.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, April.

Khandelwal, U., He, H., Qi, P., and Jurafsky, D. (2018). Sharp nearby, fuzzy far away: How neural language models use context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 284–294, Melbourne, Australia.

Lin, Z., Sung, T., Lee, H., and Lee, L. (2017). Personalized word representations carrying personalized semantics learned from social network posts. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 533–540.

Lynn, V., Son, Y., Kulkarni, V., Balasubramanian, N., and Schwartz, H. A. (2017). Human centered NLP with user-factor adaptation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1157–1166, Copenhagen, Denmark.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, USA.

Marti, U.-V. and Bunke, H. (2001). On the influence of vocabulary size and language models in unconstrained handwritten text recognition. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 260–265. IEEE.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *ArXiv*, abs/1609.07843.

Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239. IEEE.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH 2010*, pages 1045–1048.

Och, F. J. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Schler, J., Koppel, M., Argamon, S., and Pennebaker, J. W. (2006). Effects of age and gender on blogging. In *AAAI spring symposium: Computational approaches to analyzing weblogs*, volume 6, pages 199–205.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM neural networks for language modeling. In *INTERSPEECH 2012*, pages 194–197.

Ueberla, J. P. (1994). Analyzing and improving statistical language models for speech recognition. *CoRR*, abs/cmp-lg/9406027.