Factorised Hidden Layer Based Domain Adaptation for Recurrent Neural Network Language Models

Michael Hentschel*†, Marc Delcroix†, Atsunori Ogawa†, Tomoharu Iwata†, Tomohiro Nakatani†
* Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara, Japan
michael.hentschel.mc5@is.naist.jp
† NTT Communication Science Laboratories, NTT Corporation, 2-4, Hikaridai, Seika-cho, Kyoto, Japan
{marc.delcroix, ogawa.atsunori, iwata.tomoharu, nakatani.tomohiro}@lab.ntt.co.jp

***Abstract*—Language models, which are used in various tasks including speech recognition and sentence completion, are usually used with texts covering various domains. Therefore, domain adaptation has been a long-ongoing challenge in language model research. Conventional methods mainly work by the addition of a domain dependent bias. In this paper, we propose a novel way to adapt neural network-based language models. Our proposed approach relies on a linear combination of factorised hidden layers, which are learnt by the network. For domain adaptation, we use topic features from latent Dirichlet allocation. These features are input into an auxiliary network, and the output of this network is used to calculate the hidden layer weights. Both the auxiliary network and the main network can be trained jointly by error backpropagation. This makes our proposed approach completely unsupervised. To evaluate our method, we show results for the well-known Penn Treebank and the TED-LIUM dataset.**

I. Introduction

Language models (LMs) are usually trained on training data covering a large variety of text domains. However, domain specific language models usually show a lower perplexity (PPL) and better performance in automatic speech recognition (ASR) tasks than general LMs at test time. Adapting general LMs to specific topics or genres has therefore been a subject of ongoing research interest. Various approaches have been proposed for N-gram and neural network-based LMs, which benefit from topic or genre information. In many approaches, this topic information is provided by topic models, such as, latent Dirichlet allocation (LDA) [1] .

Conventional feature-based domain adaptation methods for neural network-based LMs usually add a bias to the network input or output. This bias depends on the topic feature or a (given) domain label. Our proposed method is fundamentally different from these conventional methods. We factorise the output layer into multiple layers, where each layer is weighted by a factor weight. This means we use a linear combination of different subspaces. With our method, some adaptation layers can learn information that is common to different topics, while other layers can learn topic specific information. This is similar to the linear combination of multiple domain dependent LMs, which share a common hidden state. We cannot form such a linear combination with only a bias. The factor weights are calculated from an auxiliary network, which is trained jointly with the main network by standard error backpropagation. As input into the auxiliary network, we use LDA features extracted from a sliding window. This method has been successfully employed for acoustic model adaptation [2], [3] and speaker aware beamforming [4] and we investigate its application to LMs.

We provide experimental results for the well-known Penn Treebank (PTB) [5] and the TED-LIUM dataset [6], [7] to evaluate our method. We use state-of-the-art long short-term memory (LSTM) [8] as a recurrent unit because LSTM-LMs perform better than vanilla recurrent neural network LMs (RNN-LMs) [9], [10]. The results show that our method performs consistently better on both corpora than a baseline LSTM-LM.

II. Previous Research

In previous research, topic tracking LMs [11] were proposed for the topic adaptation of classical N-gram LMs. The topic tracking LMs used an LDA-like framework to adapt N-gram probabilities to topic information. This method was effective when used on an English and a Japanese lecture corpus.

For neural network-based LMs, there are two main paradigms, namely model-based and feature-based adaptation. With model-based adaptation, which means re-training or fine-tuning network weights, an adaptation layer is inserted into the network. The weights in this adaptation layer are updated using in-domain data. model-based adaptation has been used for feed-forward [12], [13] and RNN-LMs [14]. Recently, model-based adaptation with a linear hidden network (LHN) [15], [16] was also proposed. However, fine-tuning can be a problem if the adaptation data are few, because the adapted models are prone to overfitting [17].

In feature-based adaptation, that is topic-based bias adaptation, domain specific features are used as an additional input into the network. These topic features mainly act as an additional bias, which depends on the topic. The first approach for RNN-LMs [18] made use of LDA features to allow the network to exploit the context information of the current word. This technique has also been used for RNN-LMs [19] on multi-domain broadcast data in the MGB Challenge [16]. In the context of the MGB Challenge, the authors showed that feature-based model adaptation outperforms model-based adaptation. Domain adaptation has mainly been proposed using vanilla RNN-LMs. To the best of our knowledge, the only other previous research on LSTM-LM adaptation was

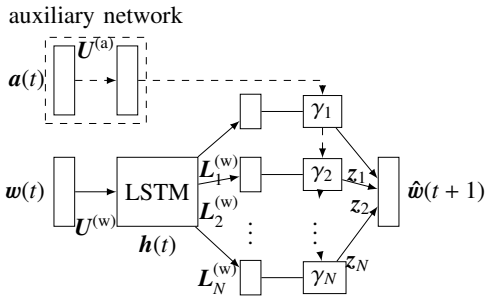

Fig. 1. LSTM-LM with factorised hidden layer for model adaptation (factLSTM).

presented in [20], which uses the same adaptation mechanism as [18].

III. LSTM Language Model

Before explaining our proposed adaptation technique in more detail, we briefly review a baseline LSTM-LM. Let us denote the vector encoding the current word ID by a one-hot vector $\boldsymbol{w}(t)$. Then the input into the LSTM $\boldsymbol{x}(t)$ is calculated using the embedding matrix $\boldsymbol{U}^{(\mathrm{w})}$

$$\boldsymbol{x}(t) = \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}(t). \tag{1}$$

The following set of equations is then used to calculate the output $\boldsymbol{h}(t)$ and the state $\boldsymbol{c}(t)$ of a single LSTM cell [10]

$$\boldsymbol{i}(t) = \sigma(\boldsymbol{W}^{(\mathrm{i,w})}\boldsymbol{x}(t) + \boldsymbol{W}^{(\mathrm{i,h})}\boldsymbol{h}(t-1) + \boldsymbol{b}^{(\mathrm{i})}), \tag{2}$$
$$\boldsymbol{f}(t) = \sigma(\boldsymbol{W}^{(\mathrm{f,w})}\boldsymbol{x}(t) + \boldsymbol{W}^{(\mathrm{f,h})}\boldsymbol{h}(t-1) + \boldsymbol{b}^{(\mathrm{f})}), \tag{3}$$
$$\boldsymbol{o}(t) = \sigma(\boldsymbol{W}^{(\mathrm{o,w})}\boldsymbol{x}(t) + \boldsymbol{W}^{(\mathrm{o,h})}\boldsymbol{h}(t-1) + \boldsymbol{b}^{(\mathrm{o})}), \tag{4}$$
$$\boldsymbol{g}(t) = \tanh(\boldsymbol{W}^{(\mathrm{g,w})}\boldsymbol{x}(t) + \boldsymbol{W}^{(\mathrm{g,h})}\boldsymbol{h}(t-1) + \boldsymbol{b}^{(\mathrm{g})}), \tag{5}$$
$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \odot \boldsymbol{c}(t-1) + \boldsymbol{i}(t) \odot \boldsymbol{g}(t), \tag{6}$$
$$\boldsymbol{h}(t) = \boldsymbol{o}(t) \odot \tanh(\boldsymbol{c}(t)), \tag{7}$$

where $\boldsymbol{i}(t)$, $\boldsymbol{f}(t)$ and $\boldsymbol{o}(t)$ are usually named the input, forget and output gates, respectively. $\boldsymbol{W}^{(j,\mathrm{w})}$ and $\boldsymbol{W}^{(j,\mathrm{h})}$ denote the weight matrices for gate j for the word input and the previous hidden layer, respectively. $\boldsymbol{b}^{(j)}$ indicates the bias vectors for the respective gates. Since we use vector notation in the above equations, $\sigma(\cdot)$ is the element-wise sigmoid, $\tanh(\cdot)$ is the element-wise hyperbolic tangent and $\odot$ denotes an element-wise multiplication.

In the case of a simple LSTM-LM, $\boldsymbol{h}(t)$ would be followed by a linear layer and the softmax function to calculate the probability for the next word $\hat{\boldsymbol{w}}(t+1)$

$$\hat{\boldsymbol{w}}(t+1) = \mathrm{softmax}(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}(t) + \boldsymbol{b}^{(\mathrm{V,w})}), \tag{8}$$

where $\boldsymbol{V}^{(\mathrm{w})}$ and $\boldsymbol{b}^{(\mathrm{V,w})}$ are the weight matrix and the bias vector, respectively.

IV. Proposed Factorised Hidden Layer Based LSTM-LM Adaptation

In the conventional method for acoustic model adaptation in ASR, an adaptation feature is used as the input of an adaptation layer. However, a novel approach to acoustic model adaptation was introduced in [2], [3]. The authors showed that the adaptation performance could be further improved by applying a rather different scheme, where they used a linear combination of multiple hidden layers. When applied to our task of LM adaptation, it can be described as follows.

Figure 1 shows our proposed approach (hereafter denoted as factLSTM) using factorised hidden layers for LM adaptation. In this case, the output of the LSTM is used on the input of N linear layers with the corresponding weight matrix $\boldsymbol{L}_n^{(\mathrm{w})}$ and bias $\boldsymbol{b}_n^{(\mathrm{L,w})}$. The size of each weight matrix $\boldsymbol{L}_n^{(\mathrm{w})}$ is the number of hidden units times the vocabulary size. Each of the linear layers is weighted by a factor weight γ_n and summed up on the input to the softmax function

$$\boldsymbol{z} = \sum_{n=1}^{N} \underbrace{\gamma_n(\boldsymbol{L}_n^{(\mathrm{w})}\boldsymbol{h}(t) + \boldsymbol{b}_n^{(\mathrm{L,w})})}_{=\boldsymbol{z}_n} \tag{9}$$

$$\hat{\boldsymbol{w}}(t+1) = \mathrm{softmax}(\boldsymbol{z}). \tag{10}$$

There is no non-linearity after the factorised hidden layers. There is only a multiplication with a weighting factor before calculating the probability for the next word $\hat{\boldsymbol{w}}(t+1)$.

Each factor weight γ_n is calculated from the output of an auxiliary network, which uses topic features calculated from an LDA as input. This auxiliary network can be of arbitrary depth. We use a single linear layer followed by a sigmoid non-linearity

$$\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \ldots \gamma_n, \ldots, \gamma_N] = \sigma(\boldsymbol{U}^{(\mathrm{a})}\boldsymbol{a}(t) + \boldsymbol{b}^{(\mathrm{U,a})}), \tag{11}$$

where $\boldsymbol{U}^{(\mathrm{a})}$ and $\boldsymbol{b}^{(\mathrm{U,a})}$ are respectively the weight matrix and bias for the linear layer. The auxiliary network can be, as shown in [3], trained jointly with the main network by standard error backpropagation.

V. Latent Dirichlet Allocation

LDA is a generative model for estimating topics in a collection of documents. It assumes that each document in this collection is modelled by a mixture of an underlying set of topics. These topics are modelled by a mixture over a set of topic probabilities. As a bag-of-words model, LDA ignores the word order in a document and provides a low-rank representation of the document, namely the number of topics.

In LDA, the following generative process generates each document in a collection:

1) Sample the document length M from a Poisson distribution: $M \sim \mathrm{Poisson}(\xi)$
2) Choose a multinomial distribution over topics for the document by sampling from a Dirichlet distribution parametrised by α: $\Theta \sim \mathrm{Dir}(\alpha)$
3) For each word w_m of the M words:
   - Choose a topic: $q_m \sim \mathrm{Multinomial}(\Theta)$
   - Choose a word w_m from the unigram distribution associated with the topic $p(w_m|q_m,\beta)$

The key parameters in LDA are α and β, which have to be learned during model training. α determines the shape

of the Dirichlet distribution over the multinomial distribution from which the topics are drawn. β is a parameter directly influencing the word probabilities.

When the training corpus consists of a single document, we regard a chunk of sentences as a single document for LDA training in the experiments. To generate LDA features for each word, we estimate the topic distribution for a fixed size window of past words.

At this point, we would like to mention that we are aware that the LSTM itself captures some kind of context information. However, the cell state in the LSTM (which we relate to the context stored in an LSTM) is processed by an exponentially decaying function. That means that a context from further in the past will be weighted less than one from the more recent past. On the other hand, LDA neglects word order and therefore all the words in a document will contribute equally when a topic distribution is calculated.

VI. Experiments

A. Dataset

We use two different datasets in the experiments. The first is the well-known PTB. It consists of articles from the Wall Street Journal and the training set has roughly 0.9M words and a 10K vocabulary. We used the standard preprocessing, that is, sections 0-20 as training, 21-22 as validation and 23-24 as the test set.

The second corpus we used was the TED-LIUM corpus [6]. Our ASR system was based on the standard Kaldi [21] recipe. However, to train the LMs, we used an enhanced dataset consisting of subtitles crawled from other TED talks. In total we crawled subtitles for 2494 talks. The resulting training set had a size of 5.1M tokens with a vocabulary size of 73K words. We thresholded the vocabulary to include only words that appeared more than once, which resulted in an effective vocabulary size of 43K words.

We used our own validation and test sets when training our LMs. The order of the talks was the same as in the IWSLT 2011 evaluation campaign [22]. For consistency, we generated our datasets from the original subtitles in the same way as our 5.1M word training set. In the experimental results section, we will report results for our own test set and the Kaldi recipe's test set.

B. LDA Training and Topic Estimation

To train our LDA model, we applied two different schemes depending on the dataset. For PTB, we used the same processing scheme as in [23]. That means, we divided the training set into chunks of 10 utterances, and we regard each of these chunks as a single document. For the TED talks, it was not necessary to make this segmentation because the dataset consists of different talks and we can use each talk as a separate document.

Before training the LDA, we removed a list of common stop words, as well as, high and low frequency words. This preprocessing was only employed to train the LDA and to compute the LDA features. The LDA implementation for our experiments was the one provided in scikit-learn [24].

To calculate the LDA features for each word in the datasets, we used LDA features extracted from a sliding window covering the previous 50 words in case of PTB and 200 words in case of TED. The LDA features represent the topic distribution over this sliding window.

TABLE I
PPL on the validation set of PTB for different numbers of factorised hidden layers versus different LDA dimensions (LSTM-LM 105.66). The number in brackets gives the number of factors used.

Model \ LDA topics	30	40	50	60
factLSTM (5)	105.06	105.55	105.93	106.08
factLSTM (10)	102.15	102.11	102.81	101.02
factLSTM (20)	102.92	102.80	101.54	101.06
factLSTM (30)	101.36	102.64	102.24	100.91
factLSTM (40)	103.75	102.69	101.75	**100.69**

C. Neural Network Training Parameters

The networks in our experiments used 300 LSTM units. We used a single layer of LSTM units, as this exhibited the best performance with the following training parameters. All networks were trained for 20 epochs on PTB and 40 epochs on TED talks. The results given below were chosen from the best model on the validation set. The learning rate was 0.1 and we used the AdaGrad optimiser [25]. Gradients were clipped to an L2-norm of 5. The mini-batchsize was 128 and the backpropagation through time length was 20 words. In all our models, we applied dropout [26] with a dropout ratio of 50% to the input of the LSTM $\boldsymbol{x}(t)$, as well as, to the output of the LSTM $\boldsymbol{h}(t)$. We implemented our LMs with the open source toolkit Chainer [27].

D. Penn Treebank Results

First, we will show PPL results for the validation and test sets of PTB. As a baseline N-gram LM, we estimated a trigram LM with Kneser-Ney [28] smoothing using the SRILM toolkit [29]. However, the PPLs we give for the neural network LMs are without trigram interpolation. That means these PPLs are the values obtained using only the neural network LM.

The performance of factLSTM depends on the number of LDA topics. To investigate which combination of LDA topics and factorised layers works well, we conducted experiments for different LDA and factorised layer combinations. Table I shows the results for LDA sizes of 30 to 60 and 5 to 40 factors. First, when the number of factors is kept constant, the PPL generally decreases as the LDA size increases. However, with a small number of factors, the PPL did not decrease significantly compared with the LSTM-LM baseline.

The second parameter to look at is the number of factorised layers. In our experiments, the PPL decreased with an increase in the number of factors. In addition, networks with more factors can make more effective use of a larger LDA. However, choosing a large number of factors does not seem to yield

TABLE II
PPLs FOR BASELINE AND FACTLSTM MODEL ON THE VALIDATION AND TEST SETS OF PTB.

Model	val	test
trigram	182.16	171.68
LSTM	105.66	98.94
factLSTM	**100.69**	**94.99**

TABLE III
PPL AND WER FOR OUR OWN SUBTITLE-BASED TEST SET AND TED-LIUM WITH 50 LDA TOPICS, A 200-WORD WINDOW SIZE AND FACTLSTM WITH 15 FACTORS. THE TRIGRAM RESULT REPRESENTS THE 1-BEST RESULT AND THE RESULTS FOR THE NEURAL NETWORK LMS ARE FOR 100-BEST RESCORING.

Model	test		WER[%]	
	subtitle	TED-LIUM	val	test
trigram	156.41	222.05	16.3	15.1
LSTM	51.98	156.29	14.2	12.1
factLSTM	**38.37**	**109.59**	13.7	11.7

much further PPL reduction on the PTB dataset. This might be due to the small size of PTB.

Comparing the results for the factLSTM with the LSTM-LM baseline, our proposed method consistently outperforms an LSTM-LM. When a large enough number of factors is used, our proposed method achieves a significant PPL reduction.

In Table II we show the final PPL results for the baseline models and our proposed approach. We used 60 topics for factLSTM with 40 factors. Our proposed factLSTM improved 5% on the validation set and 4% on the test set compared with the LSTM-LM baseline.

E. TED-LIUM Results

For the TED talks we show PPL results for the trigram distributed with the Kaldi recipe [7]. As with PTB, the PPLs shown for TED lack any N-gram interpolation. However, we used trigram interpolation for 100-best rescoring. The corresponding interpolation weights for each LM and the trigram were optimised on the validation set (provided with the Kaldi recipe).

Table III shows our results for the TED-LIUM dataset. The trigram, which is provided with the Kaldi recipe, uses a different vocabulary and more training data than our models. However, the training data are mainly out of domain data, which explains the difference in PPL compared with the neural network-based models[1].

With an LSTM-LM the PPL decreases greatly (as expected) compared with the trigram LM. The reduction was 60% for our subtitle-based test set and 17% for the TED-LIUM test set. In 100-best rescoring, we reduced the WER by 20% with an LSTM-LM.

For TED-LIUM we used LDA features from 50 topics and a window size of 200 words. For this factLSTM we used 15 factors (limitation due to GPU memory size). On the TED-LIUM test set, our method showed significantly lower PPL than LSTM-LM, that is a 29% relative reduction. In 100-best rescoring, the WER was 3% relative lower than an LSTM-LM on the test set.

[1]We also estimated a trigram on our own training data, which considerably reduced the PPL of the trigram LM to 106.58, but in order to stay consistent with the Kaldi recipe we did not use this trigram in the experiments.

VII. CONCLUSION AND FUTURE WORK

We presented a novel approach to unsupervised domain adaptation for neural network LMs based on factorised hidden layers. The method achieves a lower PPL than a conventional LSTM-LM on the PTB and TED-LIUM corpus. We also saw a significant reduction in WER compared with an LSTM-LM baseline after 100-best rescoring on the TED-LIUM dataset. Our proposed method was able to outperform the other methods for two very different datasets.

To further improve our method, we are currently looking into combining factLSTM with bias adaptation. In addition, our proposed approach increases the number of parameters, and we are considering different strategies for parameter reduction by, for example, applying the factorisation to a different layer in the network.

REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[2] M. Delcroix, K. Kinoshita, T. Hori, and T. Nakatani, "Context adaptive deep neural networks for fast acoustic model adaptation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 4535–4539.

[3] M. Delcroix, K. Kinoshita, C. Yu, A. Ogawa, T. Yoshioka, and T. Nakatani, "Context adaptive deep neural networks for fast acoustic model adaptation in noisy conditions," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5270–5274.

[4] K. Zmolikova, M. Delcroix, K. Kinoshita, H. Takuya, A. Ogawa, and T. Nakatani, "Speaker-aware neural network based beamformer for speaker extraction in speech mixtures," in *INTERSPEECH*, 2017, pp. 2655–2659.

[5] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[6] A. Rousseau, P. Deléglise, and Y. Esteve, "TED-LIUM: an automatic speech recognition dedicated corpus." in *LREC*, 2012, pp. 125–129.

[7] W. Williams, N. Prasad, D. Mrva, T. Ash, and T. Robinson, "Scaling recurrent neural network language models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015*. IEEE, 2015, pp. 5391–5395.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling." in *INTERSPEECH*, 2012, pp. 194–197.

[10] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 23, no. 3, pp. 517–529, 2015.

[11] S. Watanabe, T. Iwata, T. Hori, A. Sako, and Y. Ariki, "Topic tracking language model for speech recognition," *Computer Speech & Language*, vol. 25, no. 2, pp. 440–461, 2011.

[12] J. Park, X. Liu, M. J. Gales, and P. C. Woodland, "Improved neural network based language modelling and adaptation," in *INTERSPEECH*, 2010.

[13] T. Alumäe, "Multi-domain neural network language model." in *INTERSPEECH*, vol. 13, 2013, pp. 2182–2186.

[14] O. Tilk and T. Alumäe, "Multi-domain recurrent neural network language model for medical speech recognition." in *Baltic HLT*, 2014, pp. 149–152.

[15] R. Gemello, F. Mana, S. Scanzio, P. Laface, and R. De Mori, "Linear hidden transformations for adaptation of hybrid ANN/HMM models," *Speech Communication*, vol. 49, no. 10, pp. 827–835, 2007.

[16] P. Bell, M. J. Gales, T. Hain, J. Kilgour, P. Lanchantin, X. Liu, A. McParland, S. Renals, O. Saz, M. Wester *et al.*, "The MGB challenge: Evaluating multi-genre broadcast media recognition," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 687–693.

[17] S. R. Gangireddy, P. Swietojanski, P. Bell, and S. Renals, "Unsupervised adaptation of recurrent neural network language models." in *INTERSPEECH*, 2016, pp. 2333–2337.

[18] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model." *SLT*, vol. 12, pp. 234–239, 2012.

[19] X. Chen, T. Tan, X. Liu, P. Lanchantin, M. Wan, M. J. Gales, and P. C. Woodland, "Recurrent neural network language model adaptation for multi-genre broadcast speech recognition," in *INTERSPEECH*, 2015.

[20] D. Soutner and L. Müller, "Application of LSTM neural networks in language modelling," in *International Conference on Text, Speech and Dialogue*. Springer, 2013, pp. 105–112.

[21] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi Speech Recognition Toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011.

[22] M. Federico, L. Bentivogli, P. Michael, and S. Sebastian, "Overview of the IWSLT 2011 evaluation campaign," in *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, 2011.

[23] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 5528–5531.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[25] J. Duchi, E. Hazan, and Y. Singer, "Adative subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[27] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: a next-generation open source framework for deep learning," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in the Twenty-ninth Annual Conference on Neural Information Processing (NIPS)*, 2015.

[28] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *1995 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1. IEEE, 1995, pp. 181–184.

[29] A. Stolcke, "SRILM – an extensible language modeling toolkit," in *International Conference on Speech and Language Processing*, 2002, pp. 901–904.