AN EMPIRICAL STUDY OF TRANSFORMER-BASED NEURAL LANGUAGE MODEL ADAPTATION

Ke Li[1,2]*, Zhe Liu*[1]*, Tianxing He*[3]*, Hongzhao Huang*[1]*, Fuchun Peng*[1]*, Daniel Povey, Sanjeev Khudanpur*[2]

[1] Facebook AI, Menlo Park, CA, USA
[2] Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD, USA
[3] Massachusetts Institute of Technology, Cambridge, MA, USA

ABSTRACT

We explore two adaptation approaches of deep Transformer based neural language models (LMs) for automatic speech recognition. The first approach is a pretrain-finetune framework, where we first pretrain a Transformer LM on a large-scale text corpus from scratch and then adapt it to relatively small target domains via finetuning. The second approach is a mixer of dynamically weighted models that are separately trained on source and target domains, aiming to improve simple linear interpolation with dynamic weighting. We compare the two approaches with three baselines – without adaptation, merging data, and simple interpolation – on Switchboard (SWBD) and Wall Street Journal (WSJ). Experiments show that the mixer model generally performs better than baselines and finetuning. Compared with no adaptation, finetuning and the mixer approach obtain up to relative 11.5% and 14.1% WER reductions on SWBD, respectively. The mixer model also outperforms linear interpolation and merging data. On WSJ, the mixer approach achieves a new state-of-the-art WER result.

Index Terms— neural language model, language model adaptation, Transformer, linear interpolation, automatic speech recognition

1. INTRODUCTION

Neural language models (LMs) are an important module in automatic speech recognition (ASR) [1, 2]. They perform better in modeling long range dependency than n-gram language models [3], and are mainly used in the second-pass decoding stage via N-best or lattice rescoring [4]. Recurrent neural networks (RNNs) [1], especially LSTMs [5], are the most commonly used architecture. Recent study shows that well configured Transformer-based LMs outperform LSTM-based ones for rescoring on Librispeech [6]. Thus, in this study, we use Transformer architecture on large-scale corpora.

Most neural LMs in ASR systems are built from transcriptions or text corpora from similar domains. Lack of adequate in-domain data negatively affects the performance of neural LMs. Thus, how to improve the performance of a neural LM for a relatively small target corpus by effectively leveraging out-of-domain large-scale corpora is an important research topic. Mismatch between source and target domains makes such adaptation challenging.

In this work, we empirically explore two adaptation approaches for Transformer LMs. The first is a two-stage training approach including pretraining and finetuning. We pretrain a Transformer LM on the source domain with large-scale text and then adapt it to the target domain via weight transfer (i.e., finetuning parameters). Strategies of finetuning mainly depend on two factors – the size of the target domain and its similarity to the source domain. Our discussion is under the scenario that the target domain is similar to the source one, since when domains are too different, any adaptation technique can hardly help. We experiment with two weight transfer methods: (i) finetuning all parameters of a pretrained model, and (ii) only adapting the top fully connected (FC) feed forward layers, keeping other parameters freezed in the pretrained model.

Though finetuning is a common adaptation approach, to finetune large pretrained models with limited adaptation data tends to suffer from overfitting. Model interpolation is a simple and robust alternative, which has been widely used as an adaptation method for statistical LMs in ASR [7, 8]. In this work, we apply model interpolation to neural LMs, and propose a mixer model that dynamically fuses separately trained models based on local context, aiming at improving simple linear interpolation which uses fixed combination weights globally. The mixer model consists of both source and target models, and a mixer layer to dynamically predict fusion weights of hidden states from each model at each time step.

Our mixer approach is mainly inspired by similar work from Irie et al. [9] for building a robust RNNLM on multi domains and Das et al. [10] for building a universal acoustic modeling. However, the former one uses word embeddings as inputs to the mixer layer, which is not context-aware. And without comparison with linear interpolation makes it is unclear about the gain from dynamic weighting. In this study, we compare the proposed approaches with baseline methods including data merging and linear interpolation. Through this work we hope to provide some practical guidance on which neural LM adaptation method to choose for ASR.

2. RELATED WORK

In this section, we briefly review previous work on adaptation approaches for neural LMs in ASR systems. Most previous studies have focused on LSTM based LMs. There are mainly two categories depending on what a model adapts to. In the first category, a model adapts to recent history. This is inspired by cache models in n-gram LMs [11, 12]. There have been studies for RNNLMs to adapt to recent history [13, 14] under the fast margin adaptation framework [15].

In the second category, a model adapts to a specific topic, style, or domain. The strategy can be feature-based or model-based. For feature-based method, Mikolov et al. [16] used context-aware vectors as extra input features to RNNLMs to adapt long span topic information. Similarly, Chen et al. [17] explored topic modeling approaches to extract topic features as additional inputs to RNNLMs for genre and topic adaptation in a multi-genre broadcast transcrip-

Ke Li performed this work while she was a research intern at the Speech Team of Facebook AI.

tion task. For model based approach, Gangireddy et al. [18] investigated two domain adaptation approaches: i) scaling forward-propagated hidden activations and ii) directly finetuning the whole RNNLM in a broadcast transcription task. Ma et al. [19] explored three finetuning strategies for LSTM based LMs.

3. METHODS

3.1. Transformer Decoder as Language Model

The original Transformer model [20] contains an encoder and a decoder for neural machine translation. For the task of language modeling, we do not need the encoder component. Specifically, we adopt the generative pretraining Transformer (GPT) [21] as model architecture in this work.

The GPT model is a stack of 12 Transformer decoder blocks and each block consists of a masked multi-head self-attention module and a feed forward module, as shown in "Source Model" in Figure 1. The mask is used to prevent the model from using any future context, which is essential for auto-regressive LMs. Positional embeddings [20] are added on inputs.

Let us denote the input of the l-th Transformer block at time t as $h_t^{(l-1)}$, which is the output from (l-1)-th Transformer block. The l-th self-attention module transforms it as follows:

$$q_t^{(l)}, k_t^{(l)}, v_t^{(l)} = \mathbf{Q}h_t^{(l-1)}, \mathbf{K}h_t^{(l-1)}, \mathbf{V}h_t^{(l-1)} \quad (1)$$

$$s_t^{(l)} = (s_{t-1}^{(l)}, (k_t^{(l)}, v_t^{(l)})) \quad (2)$$

$$c_t^{(l)} = \mathbf{W_s}\text{SELFATTENTION}(s_t^{(l)}, q_t^{(l)}) + b_s \quad (3)$$

$$c_t^{(l)} = \text{LAYERNORM}(h_t^{(l-1)} + c_t^{(l)}) \quad (4)$$

where $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are projection matrices of l-th self-attention module on the input for obtaining queries, keys, and values, and $s_t^{(l)}$ is the sequence of cached key-value pairs up to time t. SELFATTENTION denotes the masked multi-head self-attention module. $\mathbf{W_s}$ and b_s are parameters of the output projection layer applied to the output of the multi-head self-attention module. LAYERNORM denotes the layer normalization operation [22]. We follow the original paper [20] to add layer normalization after each module (as shown in Figure 1), instead of before (as this paper [6] does).

The normalized output $c_t^{(l)}$ is fed into the feed forward module as below:

$$h_t^{(l)} = \mathbf{W_1}\text{Activation}(\mathbf{W_0}c_t^{(1)} + b_0) + b_1 \quad (5)$$

$$h_t^{(l)} = \text{LAYERNORM}(c_t^{(l)} + h_t^{(l)}) \quad (6)$$

The feed forward module consists of two FC layers with the Gaussian error linear unit [23] as activation function, followed by a residual connection with layer normalization. We refer to the output $h_t^{(L)}$ of the last L-th Transformer block as "hidden states" in the following paper.

3.2. Approach 1: Finetune GPT

Effectively leveraging large amount of raw text is critical to reduce the dependency on labeled data in natural language processing (NLP). Evidence suggests that unsupervised generative pretraining on a large amount of generic data can improve performance on a range of NLP tasks [21, 24, 25, 26]. A commonly used two-stage training approach [21] combines pretraining and finetuning. First, a neural model is trained on the unlabeled raw text via a language modeling objective. The parameters of the model are then adapted to a target task with labels using the corresponding task-specific objective. In this work, since our end task is still language modeling, we do not need to change the objective function in the finetuning stage.

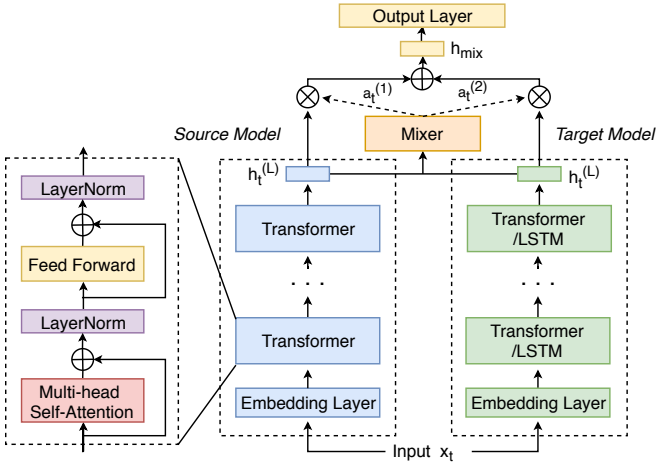

Fig. 1: Mixer model of a source GPT, a target Transformer or LSTM model, and a mixer layer with hidden states as inputs.

We experiment with two finetuning strategies in this work. The first one is to tune all parameters of the pretrained model on the target (i.e. adaptation) data. However, when the target-domain data is small-scale comparing to the pretraining data, this approach could make the model suffer from overfitting. Thus, in the second strategy, we freeze all the parameters except for the last FC feed forward layer in the last Transformer block of the pretrained model, and only finetune this layer. Since the pretrained model has a feed forward module in each Transformer block, we do not add an extra FC layer as this paper [19] does. For both finetuning methods, we use a learning rate 10 times smaller than in pretraining stage.

3.3. Approach 2: The Mixer Model with Dynamic Weighting

Intuitively, each source and target model may cover partial data distributions of a test corpus. The source model helps when it contains useful information that the target model does not have. Thus, we can view the adaptation problem as a model interpolation task to properly integrate the useful information from the source model with the target model. Simple linear interpolation is widely used for model combination or adaptation for n-gram LMs in ASR [7, 8]. Although it is simple and robust, it is not optimal in theory since the globally fixed interpolation weights are not aware of local context, except for extreme scenarios (for example, one model always outperforms the other). Dynamically integrating models with fine-grained weights could outperform simple linear interpolation.

To verify the potential of dynamic weighting, we compute the *oracle* perplexity by choosing the model that gives the best probability at each time step. Comparing with simple linear interpolation, we observed a significant perplexity drop (The "Oracle" result in Table 3). This indicates there is sufficient room to improve the baseline.

We explore a mixer model approach to achieve dynamic weighting motivated by the *oracle* experiments. The mixer model consists of both source and target models, and a mixer layer that predicts the fusion weights of hidden states from each model at each time step, as shown in Figure 1.

The mixer layer is a 1-layer multi-head self-attention module, followed by a feed forward module and a softmax output layer. Let us assume we have K separately trained models, and denote the hidden state from the last layer L of each model as h_i^L, where i is in range [1, K]. For simplicity, we remove the subscript for time in notations here. The concatenated hidden states are first projected to a common space by a linear projection layer and then fed into the self-attention module. The mixer layer's output a, a K-dimensional vector, is the probability distribution of weights on these states. The weight vector a, mixed hidden state h_{mix}, and final output y are written as follows:

$$a = \text{MIXER}(\mathbf{W}\text{concat}(h_1^L, ..., h_K^L) + b) \quad (7)$$

$$h_{\text{mix}} = \sum_{i=1}^{K} a_i h_i^L \quad (8)$$

$$y = \text{Softmax}(\mathbf{W_o} h_{\text{mix}} + b_o) \quad (9)$$

4. EXPERIMENTS

4.1. Datasets

We use two large text datasets to train the source GPT models. The first is mainly a corpus of public Facebook posts and comments. It contains around 110M English sentences. We also include the text of Fisher corpus into it since it is similar to one of the target domain. We denote the merged corpus as "Source1" in all tables. The second corpus is an English subset (10%) of the Common Crawl News (CCNews) [27], a news corpus contains articles published worldwide between September 2016 and February 2019. The subset we use contains 100M sentences and 2.7 billion words.

We evaluate the adaptation methods on three target datasets including two speech corpora – Switchboard (SWBD) and Wall Street Journal (WSJ), and one text corpus – Wikitext-103 (Wiki103). For SWBD, we report results on the full HUB5'00 evaluation set ("Eval'00 (all)"), its "SWBD" subset ("Eval'00 (swb)"), and the RT03 test set (LDC2007S10). For WSJ, we evaluate approaches on the eval92 and dev93 test sets. We use two subsets of Wiki103 training text for analysis. One contains the first 2.5M sentences ("Wiki103-Large") and the other contains the first 0.25M sentences ("Wiki103-Small"). We use the standard dev and test sets from Wiki103. Table 1 shows a summary of the datasets.

Table 1: Data Information.

Domain	Corpus	Audio (hours)	Text (#sents)
Source	Posts + Comments	-	110M
	Fisher*	-	2.2M
	CCNews subset	-	100M
Target	SWBD	260	0.26M
	WSJ	~ 80	1.6M
	Wiki103 subsets	-	2.5M; 0.25M

* We only use text of Fisher dataset in this study.

4.2. Setups

We use Kaldi [28] for acoustic model training, decoding, and N-best rescoring. Lattice-free maximum mutual information (LF-MMI) [29] objective with the factorized time-delay neural networks (TDNN-F) [30] are used for acoustic modeling on SWBD and WSJ.

We use Fairseq[1] to implement and train neural LMs. All neural LMs in our experiments are on subword level. Table 2 presents model details. Since SWBD and Wiki103-Small are relatively small, we use a 2-layer LSTM instead of Transformer. For the rest corpora, the LMs are GPTs. We use 1-layer self-attention in the mixer layer in all experiments as we do not observe further performance improvement with more layers. The multi-head self-attention module in the mixer layer contains 4 heads and 768 hidden nodes.

Since the neural LMs are on subword level, for N-best rescoring, we need to convert word level hypotheses from first pass decoding into subword sequences. We then score them with our neural models and use these scores in the N-best rescoring script in Kaldi.

We use 16 Nvidia V100 GPUs to train GPTs. We use Adam optimizer and early stopping. The dropout rates are 0.1 and 0.3 for GPTs and LSTMs, respectively. We use byte pair encoding (BPE) [31] to get subword units. There are 25416 BPE tokens on Facebook posts and comments, and 62760 BPE tokens on the CCNews subset. Note that we use "Source1" as the source data for SWBD and WSJ, and the CCNews subset as the source data for Wiki103.

Table 2: Model Details of Neural LMs.

Corpus	Model	Layers	FC Dim*	Heads	Output
Source1,WSJ	GPT	12	3072	12	25K
SWBD	LSTM	2	768	-	25K
CCNews,Wiki103-Large	GPT	12	3072	12	62K
Wiki103-Small	LSTM	2	768	-	62K

* The hidden dimension of fully connected feed forward layer.

As for training procedure, we first pretrain source and target neural LMs separately. We then initialize the parameters of the final output layer in the mixer model in Figure 1 with the source model's corresponding ones, and train the mixer model on training data of the target corpora, with parameters of source and target models freezed. For SWBD, we also include Fisher data to train the mixer since their domains are similar.

4.3. Perplexities and WERs

The BPE level perplexities on SWBD dev and test sets are shown in Table 3. Here "Oracle" means the best results that dynamic interpolation can achieve in theory. "w/o adaptation" means the LSTM model trained only on SWBD. "Merging data" means training a GPT model on merged source1 and SWBD datasets.

Note that, "Pretrain + Finetune" in all tables means finetuning all parameters of the pretrained model since we find that it outperforms only tuning the top FC layer. This is out of our expectation since SWBD is a relatively small corpus. We think the reason can be the pretrained GPT model is already well regularized (e.g. LayerNorm and dropout). We also observe that the mixer approach obtains best perplexity on the test set "Eval'00 (all)".

We then conduct N-best rescoring experiments on SWBD, and report WERs in Table 4. We set N = 20 for all experiments. We list Kaldi RNNLM's results for reference. Since Kaldi RNNLM is trained on both SWBD and Fisher corpora, the WERs are not directly comparable with "LSTM (SWBD): w/o adaptation" in Table 4. Compared with merging data and linear interpolation, both proposed approaches perform better, and the mixer model slightly outperforms finetuning.

We then experiment with WSJ corpus, and report N-best rescored WERs by differently adapted neural LMs, as well as Kaldi RNNLM's reference results in Table 5. The "GPT (WSJ): w/o

[1] https://github.com/pytorch/fairseq

Table 3: Perplexities of neural LMs on SWBD.

Models	Dev	Eval'00 (all)
Oracle (for Mixer)	31.0	46.3
GPT (source1)	55.9	79.4
LSTM (SWBD): w/o adaptation	61.1	114.8
Baseline 1: Merging data	50.9	78.0
Baseline 2: Interpolation	43.2	66.8
Pretrain + Finetune	**34.6**	67.1
Mixer	38.5	**63.5**

Table 4: N-best rescored WERs by neural LMs on SWBD.

Models	Eval'00(all)	Eval'00(swb)	RT03
Kaldi RNNLM	11.5	7.8	13.8
GPT (source1)	10.7	7.1	13.2
LSTM (SWBD): w/o adaptation	11.8	7.8	14.5
Baseline 1: Merging data	10.6	7.1	13.1
Baseline 2: Interpolation	10.6	7.1	13.3
Pretrain + Finetune	10.5	6.9	13.0
Mixer	**10.4**	**6.7**	**12.7**

adaptation" means the target GPT model trained on WSJ. It outperforms Kaldi RNNLM trained on the same corpus, which indicates the superiority of Transformer-based LM than LSTM on relatively large corpus.

The "3-Mixer" in Table 5 means we include the GPT model trained on merged source and target data as an extra source model. In the "3-Mixer" setting, the extra source model is used to initialize the final output layer in the mixer model in Figure 1. We find that this initialization performs better than using the one trained only on source data. As expected, "3-Mixer" outperforms other methods on Eval92 and achieves a new state-of-the-art WER on WSJ.

5. ANALYSIS AND DISCUSSIONS

5.1. Mixer Layer's input: Word Embeddings v.s. Hidden States

Intuitively, hidden states contain more information than word embeddings, and thus could potentially give better performance as inputs of the mixer layer. We conduct an experiment with the "3-Mixer" setting described in Section 4.3 on WSJ. And the N-best rescored WERs in Table 6 verify that hidden states indeed slightly outperform word embeddings.

In this study, the model integration is performed on hidden state level, while a more direct way is to simply combine the two models on probability level. We try to interpolate the output probabilities of the source and target models, keeping the mixer layer the same. But results on SWBD show that mixing on hidden state level performs better.

5.2. Effect of the Size of Target Data

In practice, finetuning a pretrained model is relatively easier than the mixer approach, and so insights of choosing which one can be useful. We hypothesize that when the size of the target corpus is relatively small, the gain of the mixer approach could be larger, since the overfitting issue can be more severe with less data for finetuning.

Table 5: N-best rescored WERs by neural LMs on WSJ.

Models	Dev93	Eval92
Kaldi RNNLM	2.66	1.52
GPT (source1)	3.52	1.58
GPT (WSJ): w/o adaptation	2.62	1.40
Baseline 1: Merging data	2.78	1.33
Baseline 2: Interpolation	**2.50**	1.33
Pretrain + Finetune	2.64	1.17
2-Mixer	2.54	1.24
3-Mixer	2.53	**1.10**

Table 6: WERs by a mixer model with different inputs on WSJ.

Model	Inputs	Dev93	Eval92
3-Mixer	Word Embeddings	2.54	1.12
	Hiddens States	**2.53**	**1.10**

We conduct a perplexity evaluation on the Wiki103-Large and Wiki103-Small datasets. The source model is trained on the CC-News subset corpus. Perplexity results from the finetuned model and the mixer model are shown in Table 7. We observe larger gains by the mixer approach on Wiki103-Small. This observation supports our hypothesis that the mixer model potentially perform better than finetuning on relatively small corpus.

Table 7: Perplexities of neural LMs on Wikitext-103.

Target Datasets	Approach	Dev	Test
Wiki103-Large	Pretrain + Finetune	27.4	27.7
	Mixer	26.9 (-1.8%)	27.3 (-1.4%)
Wiki103-Small	Pretrain + Finetune	39.7	39.7
	Mixer	37.7 (**-5.0%**)	37.8 (**-4.8%**)

6. CONCLUSIONS

In this work, we mainly explore two adaptation approaches for deep Transformer based neural LMs used in ASR systems, and compare them with several baselines. The first approach is to finetune a pretrained GPT model. The second one is a mixer approach to dynamically integrate source and target models with context-aware weights. In general, we observe that the two approaches perform better than baselines under most scenarios in our experiments. In particular, we achieve a state-of-the-art WER result via the mixer adaptation approach on WSJ. Finally, we verify that hidden states outperforms word embeddings as inputs of the mixer model, and the mixer approach is more effective than finetuning when the target-domain data is small-scale.

7. ACKNOWLEDGEMENTS

The author would like to thank Jun Liu, Julian Chan, and Qiujia Li for helpful discussions, and thank Desh Raj for proofreading.

8. REFERENCES

[1] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech*, 2010.

[2] Xie Chen, Xunying Liu, Mark JF Gales, and Philip C Woodland, "Improving the training and evaluation efficiency of recurrent neural network language models," in *Proc. ICASSP*, 2015.

[3] Joshua T Goodman, "A bit of progress in language modeling," *Computer Speech & Language*, vol. 15, no. 4, pp. 403–434, 2001.

[4] Hainan Xu, Tongfei Chen, Dongji Gao, Yiming Wang, Ke Li, Nagendra Goel, Yishay Carmiel, Daniel Povey, and Sanjeev Khudanpur, "A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition," in *Proc. ICASSP*, 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney, "Language modeling with deep transformers," in *Proc. Interspeech*, 2019.

[7] Jerome R Bellegarda, "Statistical language model adaptation: review and perspectives," *Speech communication*, vol. 42, no. 1, pp. 93–108, 2004.

[8] Gokhan Tur and Andreas Stolcke, "Unsupervised language model adaptation for meeting recognition," in *Proc. ICASSP*, 2007.

[9] Kazuki Irie, Shankar Kumar, Michael Nirschl, and Hank Liao, "Radmm: recurrent adaptive mixture model with applications to domain robust language modeling," in *Proc. ICASSP*, 2018.

[10] Amit Das, Jinyu Li, Changliang Liu, and Yifan Gong, "Universal acoustic modeling using neural mixture models," in *Proc. ICASSP*, 2019.

[11] Roland Kuhn and Renato De Mori, "A cache-based natural language model for speech recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 6, pp. 570–583, 1990.

[12] Frederick Jelinek, Bernard Merialdo, Salim Roukos, and Martin Strauss, "A dynamic language model for speech recognition," in *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*, 1991.

[13] Mittul Singh, Youssef Oualil, and Dietrich Klakow, "Approximated and domain-adapted lstm language models for first-pass decoding in speech recognition.," in *Proc. Interspeech*, 2017.

[14] Ke Li, Hainan Xu, Yiming Wang, Daniel Povey, and Sanjeev Khudanpur, "Recurrent neural network language model adaptation for conversational speech recognition.," in *Proc. Interspeech*, 2018.

[15] Reinhard Kneser, Jochen Peters, and Dietrich Klakow, "Language model adaptation using dynamic marginals," in *Fifth European Conference on Speech Communication and Technology*, 1997.

[16] Tomas Mikolov and Geoffrey Zweig, "Context dependent recurrent neural network language model," in *Proc. SLT*, 2012.

[17] Xie Chen, Tian Tan, Xunying Liu, Pierre Lanchantin, Moquan Wan, Mark JF Gales, and Philip C Woodland, "Recurrent neural network language model adaptation for multi-genre broadcast speech recognition," in *Proc. Interspeech*, 2015.

[18] Siva Reddy Gangireddy, Pawel Swietojanski, Peter Bell, and Steve Renals, "Unsupervised adaptation of recurrent neural network language models.," in *Proc. Interspeech*, 2016.

[19] Min Ma, Michael Nirschl, Fadi Biadsy, and Shankar Kumar, "Approaches for neural-network language model adaptation.," in *Proc. Interspeech*, 2017.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[21] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, "Improving language understanding by generative pre-training," 2018.

[22] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton, "Layer normalization," in *NeurIPS*, 2016.

[23] Dan Hendrycks and Kevin Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL*, 2019.

[25] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[26] Prajit Ramachandran, Peter J Liu, and Quoc V Le, "Unsupervised pretraining for sequence to sequence learning," *Proc. EMNLP*, 2017.

[27] Anton Bakhtin, Sam Gross, Myle Ott, Yuntian Deng, Marc'Aurelio Ranzato, and Arthur Szlam, "Real or fake? learning to discriminate machine from human generated text," *arXiv preprint arXiv:1906.03351*, 2019.

[28] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *Proc. ASRU*, 2011.

[29] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi.," in *Proc. Interspeech*, 2016.

[30] Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur, "Semi-orthogonal low-rank matrix factorization for deep neural networks.," in *Proc. Interspeech*, 2018.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch, "Neural machine translation of rare words with subword units," *Proc. ACL*, 2016.