LATTICEBART: LATTICE-TO-LATTICE PRE-TRAINING FOR SPEECH RECOGNITION

Lingfeng Dai, Lu Chen†, Zhikai Zhou, Kai Yu†

X-LANCE Lab, Department of Computer Science and Engineering
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
Shanghai Jiao Tong University, Shanghai, China
State Key Lab of Media Convergence Production Technology and Systems, Beijing, China
{randool,chenlusz,zhikai.zhou,kai.yu}@sjtu.edu.cn

ABSTRACT

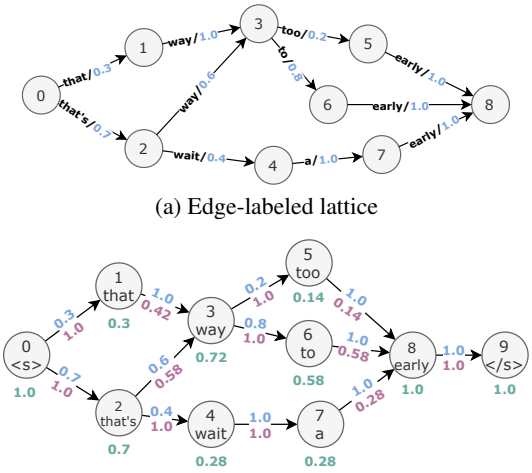
To improve automatic speech recognition, increasing work has attempted to further fix the output of ASR systems with advanced sequence models. However, the output of ASR systems differs significantly from the input form of standard sequence models. To encompass richer information, the output of ASR systems is often a compact lattice structure containing multiple sentences. This mismatch in input form significantly limits sequence models' ability. On the one hand, the widely used pre-trained models cannot directly input lattice structures and are therefore difficult to use for this task. On the other hand, the sparsity of the supervised training data forces the model to have the ability to learn from limited data. To address these problems, we propose LatticeBART, a model that decodes the sequence from the lattice in an end-to-end fashion and can use the pre-trained language models' prior. In addition, this paper proposes the lattice-to-lattice pre-training method, which can be used when annotated data is missing, using easily generated lattice with the ASR system for training. The experimental results show that our model can effectively improve the output quality of the ASR system.

Index Terms— speech recognition, lattice-to-lattice, lattice-to-sequence, pre-trained language model

1. INTRODUCTION

Automatic speech recognition (ASR) systems have evolved considerably over the years, from the traditional WFST-based ASR systems to the more innovative end-to-end (E2E) ASR systems [1, 2]. Benefiting from advanced architectures and cleverer training methods, ASR systems have surpassed human recognition accuracy in specific environments, but their results in generic environments are still less than satisfactory. On the one hand, environmental noise may cause ASR systems to give incorrect results [3]. On the other hand, the reliance on large amounts of data makes it quite challenging to train a robust ASR system. In addition, the direct output of most ASR systems often does not match human reading habits, such as lack of punctuation, lack of capitalization, etc. This is because text regularisation is often used to reduce the learning difficulty of ASR models. To further improve the quality of ASR recognition results, speech recognition post-processing systems were introduced.

Rule-based post-processing systems have many grammar rules built in to do inverse text normalization of the output of the ASR system. Such systems are demonstrably efficient, but constructing complex rules can be a challenging task. Models trained with autoregressive goals can easily exploit large unsupervised corpora, allowing them to achieve good results in many areas of sequence modeling, such as language modeling and machine translation [4]. As a result, a growing number of post-processing systems use sequence modeling models pre-trained on large-scale corpora to introduce a priori linguistic knowledge into the task.



(a) Edge-labeled lattice

(b) Node-labeled lattice.

Fig. 1. Example of edge-labeled lattice and node-labeled lattice. Forward (blue), backward (purple) and marginal (green) scores are listed.

Typically, however, the output of an ASR system is not limited to a single sentence, but a collection of possible sentences. Both WFST-based ASR systems and E2E ASR systems can output more compact lattices at the decoding end, in contrast to the classical sequence-to-sequence (Seq2Seq) model [5, 6], where the input is often a single sentence. This formal difference limits the performance of the Seq2Seq model in post-processing systems. Some work uses the lattice directly as the input to the model. [7] feeds the nodes in the lattice sequentially into the encoder in a topologically order, and the decoder recovers the correct sentence. [8, 9] are based on the Transformer [10] and improves on its structure to enable consuming input in the form of a lattice. However, these works require large amounts of supervised data to train the model to handle lattice.

For the problem mentioned above, we propose a model called LatticeBART having two features: 1) The model can directly use BART [11] (or other Transformer-based [12], e.g., BERT [13]) parameters to leverage the prior knowledge of the pre-trained language model (PTLM); 2) A tailored lattice-to-lattice (L2L) training approach can sufficiently train the model to gain the ability to model

† The corresponding authors are Lu Chen and Kai Yu.

lattices in the presence of only lattices.

2. LATTICE REPRESENTATION

The form of data input has a close relation to the structure of the model, so before presenting the model, we first answer the question of how to input the graph-structured lattice into the model.

Nodes The lattice output from the ASR model is a directed acyclic graph $G = \langle V, E \rangle$ which contains multiple paths from the source node to the sink. Different paths sharing some of the nodes, the lattice is a very compact structure that can hold more information than the n-best list. The lattice that records all words and weights on the transfer edges is not conducive to modeling the model, so the word information of the lattice needs to be driven to the nodes (node-labeled) using the line-graph algorithm [14] first. After applying the line-graph algorithm, the node-labeled lattice may have multiple start and end nodes. We also need to add the start and end symbols, marked as BOS and EOS, to the node-labeled lattice to ensure the uniqueness of the start and end points and preventing the lattice from separating into multiple independent subgraphs. Fig 1 shows the lattices before and after the transformation, respectively. We can use topological sorting to obtain a sequence of nodes (i.e., a series of words) of the node-labeled lattice as input to the model. The results of the topological sort are not unique, and either result is valid. The only caveat is that once this node sequence is fixed, other graph structure information needs to refer to this order as well.

Forward-backward score After doing the processing mentioned above, we move our attention to the weights of the transfer edges. Typically, a transfer edge contains a language model score and an acoustic model score, both of which describe the current node's semantic plausibility and pronunciation similarity, respectively. Two scores can be combined to form a lattice forward score by a pair of interpolation coefficients. It is common practice to normalize each node out-degree score to ensure a reasonable range of values. In addition, to represent the lattice information more comprehensively, one needs to consider the backward information flow. Lets the forward score of nodes i to j be $f_{i \to j}$. Then using equation $b_{i \leftarrow j} = f_{i \to j} m_i / m_j$ [15], one can calculate the backward score from the forward score, where m_i and m_j are the marginal scores that will be introduced immediately. The forward-backward score describes the local structural information of the lattice.

Marginal score If the forward information is considered as an information flow, then the marginal score (formula) assigned from the source node to the successor node can be used to represent the importance of the node. Suppose the marginal score of the starting node is $m_0 = 1$ and the predecessors of node i is P_i. Then the marginal score of node i $(i > 0)$ is $\sum_{j \in P_i} m_j f_{j \to i}$ [15]. It can be proved that the marginal score of the terminating node is also 1. The marginal score of a node is also one of the inputs to the model.

Lattice mask When the Transformer-like encoder extracts sentence information, each token in the sentence will do attention with other tokens under the control of an attention mask filled with 1. As a compact form of multiple sentences, each path in the lattice can be considered a sentence. Then attention will occur between the individual tokens of these sentences. If the attention matrices for the different paths are put together, we get a lattice mask in the shape of Fig 2 (5).

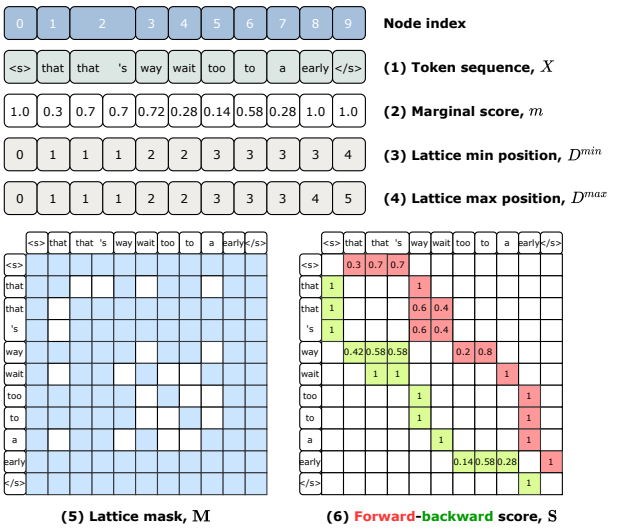


Fig. 2. Input of lattice information to the model.

Lattice positional embedding The same idea of using positional embedding to mark the position of a token in Vanilla Transformer can be used to handle the position of nodes in a lattice. The difference is that the position of the same node in different paths may vary. For example, the word '*early*' in Fig 1 (b) can be at position 3 or 4 in different paths. It is not practical to enumerate all the positions of all nodes. In LatticeBART, we use the shortest and longest distance of a node from the start point as the node position coordinates and the average of the positional embedding of the two distances as the positional embedding of the node.

Ideally, the number of node-labeled lattice nodes coincides with the number of tokens in the input model, but this requires our vocabulary to contain all possible words. In contrast, the vocabulary of the generic pre-trained language model uses the subword as the minimum word construction unit, and a node is inevitably mapped to multiple tokens. For this reason, we need to "expand" the above categories of information based on the splitting of tokens. In Fig 1 (2), the word '*that's*' of node 2 is expanded into '*that*' and '*'s*', as shown in Fig 2.

3. LATTICEBART

Learning from an extensive training corpus using unsupervised training is the source of cutting-edge pre-trained language models' superior sequence modeling capabilities. ASR post-processing models need to cope with the complexity of ASR recognition results, and only a robust model can output high-quality results. For our models to perform robustly, inheriting the rich prior of PTLM is a very effective means. As stated in the introduction, LatticeBART infrastructure has been adapted from the Vanilla Transformer to handle the lattice structure and use BART parameters directly. LatticeBART decoder is consistent with the BART structure, so this section will only cover the details of the encoder.

Let the length of the input sequence be n, and the self-attention of each header of the encoder operates as shown in Equation 1.

$$\mathbf{H_i} = Softmax\left(\frac{\mathbf{Q_i K_i^T}}{\sqrt{d_k}} + \mathbf{S} + \mathbf{M}\right)(\mathbf{V_i} + \mathbf{mZ}). \quad (1)$$

Here $\mathbf{Q_i}$, $\mathbf{K_i}$, and $\mathbf{V_i}$ have the same meaning as Transformer and result from three linear mappings of the input hidden states. $\mathbf{S}, \mathbf{M} \in$

$\mathbb{R}^{n\times n}$ are the forward-backward fraction matrix and the lattice mask, respectively. $\mathbf{m} \in \mathbb{R}^{n\times 1}$ is the sequence of marginal fractions corresponding to the token, $\mathbf{Z} \in \mathbb{R}^{1\times d_k}$ is the only parameter that does not appear in the BART and serves to map the marginal scores to the same vector space as $\mathbf{V_i}$ [16].

$$\mathbf{H} = Concat\left(\mathbf{H_1}, \mathbf{H_2}, \cdots, \mathbf{H_h}\right)\mathbf{W}. \quad (2)$$

Finally, as described in Equation 2, the output of the current encoder layer is obtained by stitching together the results of the different attention heads and mapping them with the learnable parameter matrix $\mathbf{W}$.

The lattice positional embedding mentioned in section 2 is added to the token embedding as input to the first layer of the encoder.

4. TRAINING METHODS

Although the adaptation improvements to LatticeBART only introduce a tiny number of parameters on top of BART, how lattice attention is calculated affects the output distribution of each layer of the model. To better exploit the talents of LatticeBART, additional training data for adapting the lattice input is necessary. Most previous work has input lattice structures at the Encoder side and predicted sentences at the decoder side. This format has the advantage of being highly consistent with the downstream task, making the model directly usable after training. However, the disadvantage is that pairs of training data need to be constructed, and annotated data is often expensive, so we propose L2L pre-training.

4.1. Lattice-to-Lattice pretraining

Collecting the audio for training an ASR system is relatively easy, whereas managing the text corresponding to the audio is tedious. With an ASR system already in place, it is possible to obtain lattices by feeding audio into the system, making it easy to get many unlabeled lattices. In contrast, previous methods can not train high-quality post-processing models in the absence of labeled text. Some unsupervised techniques commonly used for Seq2Seq, such as reinforcement learning, can make use of unlabeled data. Still, the sparse learning signal of reinforcement learning dramatically reduces the availability of the data. To make more efficient use of unsupervised data, we propose the lattice-to-lattice (L2L) training method.

The core idea of L2L pre-training is to feed the noise-added lattice to the encoder and then let the decoder repair the lattice information. In terms of implementation, in addition to the causal mask that acts inside the decoder, we also give the lattice mask so that each token can only see the token before it, preventing interaction between different paths when decoding. It is easy to see that the objectives of the L2L task are highly relevant to the downstream tasks (i.e., ASR post-processing).

The fact that lattice contains a wealth of information is also how we can do diverse additive noise processing. In the L2L pre-training task, we propose the following noise addition.

Homophone substitution The most significant association between different paths in a lattice is that they have very similar pronunciations. Therefore, we perform synonym substitution on a randomly selected node as a form of noise addition. Using CMU dictionary [1], the pronunciation of a word can be obtained and a word selected from a list of similarly pronounced candidates as the replacement result.

[1]CMU dictionary (the Carnegie Mellon Pronouncing Dictionary)

Token masking We borrowed the masking strategy used in the previous BERT training, where randomly selected tokens are replaced with a special MASK symbol. However, here replacement occurs at the node level. That is, if a node is selected, all tokens contained in it will be masked.

Path masking Randomly selecting some paths on a lattice to mask is equivalent to doing path pruning on the lattice. Adding a path mask can train the model to cope with inputs from lattices of different beam widths.

Depth offset Lattice position will be randomly added to a randomly selected integer from 0 to D_r. In our experiments, we let $D_r = 10$.

Random perturbation of weights LatticeBART is expected to have the ability to resist weight perturbations in different scenarios where the acoustic and language scores of the lattice need to be adjusted appropriately. The perturbation here refers to random scaling, or random discarding, on forward-backward and marginal scores.

4.2. Lattice-to-Sentence training

Benefiting from the model structure, it is also possible to use our model directly for lattice-to-sequence (L2S) training, where the input to the decoder is a sentence, and the lattice mask is no longer input.

5. EXPERIMENTS

5.1. Experiments settings

For the purpose of inheriting the prior knowledge from BART, LatticeBART is therefore aligned with the BART base in terms of the model configuration, i.e., LatticeBART contains six adaptively modified lattice encoder layers and six decoder layers. The hidden state dimension is 768. In addition, to test the performance gain of the LatticeBART structural bias, we also set up a control group with random initialization of the parameters, i.e., the performance obtained by the model using only existing data learning without including prior knowledge. When LatticeBART used the BART parameters [2], the initial learning rate was set to 1e-5. In contrast, LatticeBART with randomly initialized parameters used 1e-4. The cosine learning rate decay method was used during training. We used the AdamW [17] optimizer by default.

In L2L pre-training, the batch size is 16, and the length of the input sequence is limited to 256, beyond which the data is truncated. On the encoder side, tokens have a 5% probability of being replaced with homonyms and a 10% probability of being masked. Lattice weights have a 10% probability of being discarded. The path masking operation is equivalent to the lattice prune. For reasons of computational efficiency, each lattice is pre-processed with a different weighting factor and pruned with Kaldi [18] tools to obtain multiple backups. During training, the encoder and decoder inputs are randomly selected from these backups. After path masking, the beam width of the lattice used for training is from 2 to 6, and by default, the beam width used for testing is 4. The node depth offset is a random integer from 0 to 10. The rest of the L2S training process configuration does not differ from L2L except that no noise is added at the encoder end.

[2]We use Hugging Face facebook/bart-base.

5.2. Datasets

Following [16], our experiments used Switchboard (SWBD) 300 hr and SWB-Fisher 2000 hr data. We follow the EESEN [1] SWBD recipe to build the baseline phone-based CTC ASR system. A 5-layer BiLSTM acoustic model with the hidden size of 320 is trained on SWBD 300 hours speech, and a 3-gram language model is trained on SWB-Fisher transcripts. The word lattices used for training LatticeBART are generated by the WFST-based approach described in [1]. The training set of LatticeBART and validation sets contain 2,131,620 and 10,000 pieces of data, respectively. We tested the performance of LatticeBART on the eval2000 and rt03 datasets, which contain 4,458 and 8,422 lattices, respectively. The sources for the eval2000 data are callhome and SWBD, while the sources for the rt03 data are Fisher and SWBD.

Table 1. WER (%) results on eval2000 and rt03.

	Model	eval2000		rt03		Avg.
		Callhome	SWBD	Fisher	SWBD	
1	lattice best path	22.5	12.4	17.1	26.0	19.5
2	10-best rescore	23.0	12.6	17.3	26.1	19.8
3	20-best rescore	22.9	12.6	17.3	26.1	19.7
4	$L2S^{*}_{20\%}$	22.9	13.3	17.6	26.4	20.1
5	$L2S^{*}_{100\%}$	21.1	12.2	15.9	24.4	18.4
6	$L2S_{20\%}$	21.4	11.5	15.9	24.6	18.4
7	$L2S_{100\%}$	19.1	10.0	13.7	22.4	16.3
8	$L2L_{20\%} \rightarrow L2S_{20\%}$	20.1	10.5	14.8	23.8	17.3
9	$L2L_{80\%} \rightarrow L2S_{20\%}$	20.1	10.6	14.7	23.6	17.3

5.3. Results and analysis

Table 1 shows the results for the different models on eval2000 and rt03. Row 1 is the result of decoding the optimal path from the lattice. Rows 2 and 3 show the results of using the 3-layer LSTM rescoring language model. We train the LSTM model on the training set text. To our surprise, the rescoring of the language model does not lead to a reduction in WER but rather hurts the final results. We conjecture that the reason for this is that the current lattice already has a superior path score.

The experimental subjects in row 4 and after that are all LatticeBARTs. L2S and L2L represent the lattice-to-sequence and lattice-to-lattice from the previous section. Superscript * indicates that the model does not use the parameters of PTLM (BART). The subscript indicates the weight of the amount of data for training LatticeBART to the complete training set. L2L $\rightarrow$ L2S indicates a two-stage training mode with lattice-to-lattice pre-training followed by lattice-to-sequence training. In the two-stage training, the L2L training set does not contain text.

We can draw the following conclusions from the results: 1) The role of more training data on post-processing tasks is evident. In row 4, 20% of the training data does not allow the randomly initialized LatticeBART to have a good enough ability to recover information, while the total amount of training data in row 5 can significantly improve this aspect of the model. 2) The prior knowledge of PTLM is important for post-processing tasks. As shown in row 6 and 7, while using only 20% of the training data, LatticeBART that inherits the PTLM prior knowledge can still significantly reduce WER and when using the full amount of labeled data, LatticeBART's performance in recovering correct sentences from noisy lattices is further improved. 3) To further improve the performance of the experienced LatticeBART when the labeled data is small, the training mode of L2L $\rightarrow$ L2S can be of great help. With L2L's noise-resistant training, even if L2S does subsequent training on more minor training data, the WERs in rows 8 and 9 are still significantly lower than in row 6.

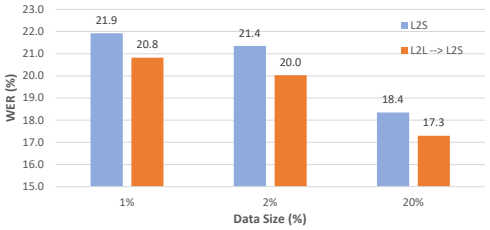


Fig. 3. Performance comparison of models with small-scale data. WER is the average over eval2000 and rt03.

To further highlight the role of L2L pre-training, we trained the model under very little data. Specifically, we stripped 1% and 2% of the total data used to train LatticeBART for the experiments. The experiments were divided into two groups. The control group "L2S" was trained with supervised data only. The experimental group L2L $\rightarrow$ L2S was first pre-trained with L2L using unsupervised data and then trained with supervised data, where the amount of unsupervised data was 80%. The experimental results are shown in the Fig 3. The results show that the experimental group significantly outperforms the control group when the data volume is minimal, which confirms that the L2L training mode can help PTLM make input data adaptation. Furthermore, this adaptation does not harm the model performance thanks to the high consistency of L2L and L2S training objectives. In addition, when the training data is increased from 1% to 20%, the WER of the control group tapers off after a period of rapid decline, while the performance of the experimental group continues to improve with the increase of data volume.

Table 2. Effect of different beam widths on WER (%).

Beam width	eval2000		rt03		Avg.
	Callhome	SWBD	Fisher	SWBD	
1	21.0	10.9	15.5	24.1	17.9
2	20.3	10.5	14.7	23.2	17.2
3	19.3	10.2	14.2	22.7	16.6
4	19.1	10.0	13.7	22.4	16.3
5	18.9	10.0	13.7	22.2	16.2
6	18.7	9.9	13.6	22.1	16.1

We also explored the performance of the model under different beam width inputs. We selected the model $L2S_{100\%}$ in Table 1 and tested it under lattices with beam widths vary from 1 to 6. The experimental results are shown in Table 2. As the beam width increases, the WER on the test set continues to decrease. When the beam width is 1, the input of the model is a sentence, and the comparison with the results in the first row of Table 1 shows that although the inputs of the model are all lattice during the training process, it still can repair the sequences.

6. CONCLUSION

This work introduces a novel model called LatticeBART that can directly convert the lattice to the sequence in an end-to-end fashion. By introducing PTLM prior knowledge and a clever lattice-to-lattice pre-training method, LatticeBART can decode better sentences from lattice than other methods.

7. ACKNOWLEDGEMENT

This work has been supported by the China NSFC Projects (No. 62120106006 and No. 62106142), CCF-Tencent Open Fund and Startup Fund for Youngman Research at SJTU (SFYR at SJTU).

8. REFERENCES

[1] Yajie Miao, Mohammad Gowayyed, and Florian Metze, "Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 167–174.

[2] Zhehuai Chen, Mahaveer Jain, Yongqiang Wang, Michael L Seltzer, and Christian Fuegen, "Joint grapheme and phoneme embeddings for contextual end-to-end asr.," in *INTERSPEECH*, 2019, pp. 3490–3494.

[3] Tian Tan, Yanmin Qian, Hu Hu, Ying Zhou, Wen Ding, and Kai Yu, "Adaptive very deep convolutional residual network for noise robust speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 8, pp. 1393–1405, 2018.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[5] Nal Kalchbrenner and Phil Blunsom, "Recurrent continuous translation models," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1700–1709.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun, Eds., 2015.

[7] Matthias Sperber, Graham Neubig, Jan Niehues, and Alex Waibel, "Neural lattice-to-sequence models for uncertain inputs," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1380–1389.

[8] Matthias Sperber, Graham Neubig, Ngoc-Quan Pham, and Alex Waibel, "Self-Attentional Models for Lattice Inputs," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 1185–1197.

[9] Pei Zhang, Niyu Ge, Boxing Chen, and Kai Fan, "Lattice transformer for speech translation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 6475–6484.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 5999–6009, 2017.

[11] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7871–7880, Association for Computational Linguistics.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[14] Robert L Hemminger, "Line graphs and line digraphs," *Selected topics in graph theory*, 1983.

[15] Lawrence R Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[16] Rao Ma, Hao Li, Qi Liu, Lu Chen, and Kai Yu, "Neural lattice search for speech recognition," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7794–7798.

[17] Ilya Loshchilov and Frank Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[18] 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 *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number CONF.