RESIDUAL ADAPTERS FOR TARGETED UPDATES IN RNN-TRANSDUCER BASED SPEECH RECOGNITION SYSTEM

Sungjun Han[*]

Universität Stuttgart, Germany

Deepak Baby, Valentin Mendelev

Amazon Alexa

ABSTRACT

This paper investigates an approach for adapting RNN-Transducer (RNN-T) based automatic speech recognition (ASR) model to improve the recognition of unseen words during training. Prior works have shown that it is possible to incrementally finetune the ASR model to recognize multiple sets of new words. However, this creates a dependency between the updates which is not ideal for the hot-fixing use-case where we want each update to be applied independently of other updates. We propose to train residual adapters on the RNN-T model and combine them on-the-fly through adapter-fusion. We investigate several approaches to combine the adapters so that they maintain the ability to recognize new words with only a minimal degradation on the usual user requests. Specifically, the sum-fusion which sums the outputs of the adapters inserted in parallel shows over 90% recall on the new words with less than 1% relative WER degradation on the usual data compared to the original RNN-T model.

Index Terms— speech recognition, end-to-end models, RNN-T, incremental learning, residual adapters

1. INTRODUCTION

End-to-end automatic speech recognition (ASR) models based on recurrent neural network transducer (RNN-T) or transformer transducer (TT) have achieved remarkable success [1–4]. However, it requires a lot of time and computational resources to train these models to reach high recognition accuracy. This is an important challenge in maintaining a real-world voice assistant ASR model up-to-date with the changing distribution of users' requests. Regular model updates are needed to enable correct recognition of new words because RNN-T / TT models tend to incorrectly recognize words not present in the training data [5, 6].

One solution to this challenge is to adapt the external language model for shallow-fusion [7, 8] or re-scoring [9] or use a spelling-correction system [10, 11]. However, this is not a complete solution to the problem as it is still contingent on the ability of the RNN-T model to assign significant weight to hypotheses compatible with a new word. If the weight is too low it's unlikely that the correct word will be recovered.

Another solution is directly adapting the RNN-T model through incremental learning [12, 13] where the model is updated as the training data for the new words becomes available rather than training from scratch. A recent work [14] has demonstrated in a real-world scenario that incrementally updating the RNN-T model through simple fine-tuning is an effective approach in learning to recognize new words.

The approach from [14] requires changing all the parameters of the model for each incremental update and leads to models that are successively dependent on predecessors, though in general it is desirable to make incremental updates independent from one another and affecting as low number of parameters of a model as possible. Encapsulating knowledge into a small set of parameters makes it possible to flexibly apply the adapter when needed (e.g. on a specific device type, or for a certain user cohort).

In this paper, we investigate a real-world scenario with multiple independent *hot-fixes* on the RNN-T-based ASR model using real-production traffic data where each fix consists of a small set of new words. By a hot-fix we mean a change in the model which is expected to alter recognition result for a very limited set of queries (e.g. where a certain new word is mentioned) and which can be deployed while the system is running. In order to achieve the non-dependency between the hot-fixes, we propose to use residual adapters [15]. For each set of target words, new residual adapter layers are introduced and trained while the original parameters of the model remains fixed. Also, the base model training data are assumed to be available to minimize degradation on the average data during training of each adapter (*data replay*). Then, the independently trained adapters from different updates are inserted into the base model in parallel without further tuning. Given the input, the adapter output activations are fused through a sum operation (see Figure 1).

The proposed approach is suitable for real-world voice assistant applications in dealing with multiple hot-fixes simultaneously. Since only the residual adapter layers are trained on the data associated with unseen words, the ability to bias the model towards those words is solely represented by the corresponding adapters and can be pruned when needed without affecting other updates. Also, they are easy to deploy and save as the adapters are relatively small compared to the base model.

[*]Performed the work while at Amazon as an intern.

978-1-6654-7189-3/22/$31.00 ©2023 IEEE

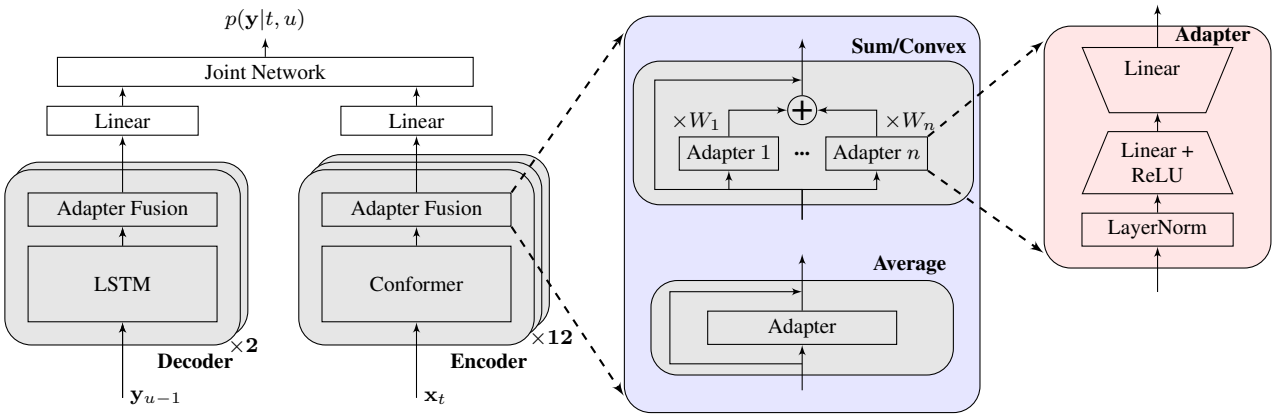


Fig. 1. RNN-T with n independently trained adapters combined through different adapter-fusion methods is illustrated. We consider three fusion methods: sum, convex, and average. For the sum and convex fusion, the adapters are inserted in parallel and the output activations are summed after being multipled by their respective weights w_i. For the sum-fusion, all the weights are set to 1 while for the convex fusion they are set to $1/n$. The average-fusion averages the weights of all the adapters into a single adapter with equal weighting. The adapter architecture used for the experiments is also shown.

The main contributions of the paper are as follows. 1) We show that with a careful selection of the data replay strategy and the adapter-fusion method, it is possible to combine multiple independently trained adapters on-the-fly without degrading the performance on the new target words and the average data (Sections 5.1 and 5.2). 2) Through experiments on the different fusion methods and adapter configurations, we elucidate the key ingredients to successfully combine adapters (Sections 5.2 and 5.3). We also conduct a qualitative analysis to further understand the results (Section 5.4).

2. RELATED WORK

To our knowledge, there is no prior work done on combining independently trained adapters without the help from an additional classifier tasked to select which adapter to use. While being a popular topic in natural language processing [16–18], only a few works have explored the use of adapters in ASR for the RNN-T. One work [19] studied the use of adapter to better deal with rarely occurring heavily accented speech and found the adapters to work well. However, they have not explored how to use multiple adapters simultaneously. Another work [20] studied using multiple adapters for multilingual ASR to balance the performance of high and low resource languages. It was assumed that the appropriate adapter would be selected through an external language classifier at test time. However, such a classifier is not easily applicable in detecting specific new words in a streaming fashion as we cannot know when the new word will occur until it actually does.

3. METHODOLOGY

We simulate a real-world hot-fixing scenario for a voice assistant application with the RNN-T system where there are new words the model has not seen before. The use of an external language model or a spelling-correction system in addition to the RNN-T is out-of-scope for this paper and is not considered. Examples for the new words can be names of newly released songs or athlete names during the Olympics. The RNN-T model is trained from scratch using all the data $\mathcal{D}$ available to a certain point in time. We refer to this model as the base model M. We assume n batches of mutually exclusive sets of m new words $\{B_1, B_2, ..., B_n\}$ each consisting of training examples for the new words. Each batch represents a distinct short-term trend appearing in the data distribution .

We train new residual adapters layers in the base model for each batch. Hence, we train n adapters, $\mathcal{A} = \{A_1, A_2, ..., A_n\}$, one for each batch. The newly trained adapters are inserted into the base model by combining with the old ones. The key question is whether multiple adapters can be combined without the adapters interfering with each other and degrading much on the average data. We assume that we have access to $\mathcal{D}$ to utilize for the data replay when learning to recognize the new words. We explore different strategies by using different sampling ratio for the old data and the new data. Different data replay strategies are denoted as $(1-w, w)$ where w refers to the amount of sampling weight given to the new words.

3.1. Residual Adapters

Residual adapter layers are a small set of new parameters inserted in the base model for an efficient transfer learning. During training, only the parameters of the adapter are trained while the parameters of the base model remains frozen. We follow the same adapter architecture proposed in [19]. Figure 1 illustrates the adapter architecture. It consists of a layer normalization layer followed by a down-projecting feed-

forward layer with a ReLU non-linear activation. It projects the input to an intermediate representation of size d_a that is smaller than the input dimension d. This is followed by an up-projecting feed-forward layer which projects the hidden representation back up to d. The adapters are inserted in between the encoder layers or the decoder layers of the RNN-T with a residual connection.

3.2. Adapter Fusion

We investigate three approaches for combining more than one one adapter that we wish to utilize simultaneously: *sum*-fusion where the adapters are inserted in parallel and the adapter outputs are linearly combined with all the weights set to 1 (Eqn. 1), *convex*-fusion where the adapters are also inserted in parallel but instead the outputs are combined with equal weights of $\frac{1}{n}$ (Eqn. 2), and *average*-fusion where a single adapter with the weights of the the adapters averaged with equal weighting is inserted (Eqn. 3). The three adapter fusion approaches are illustrated in Figure 1. In all approaches, each adapter can be applied and taken out without needing to discard or retrain other adapters. For the sum-fusion, nothing needs to be done when a new adapter is inserted or one of the operating adapters is deleted. For the convex-fusion, only the weights need to be adjusted through a re-normalization to make them sum to 1. For the average-fusion, the weights of the remaining adapters need only be averaged again.

$$\text{Sum-Fusion}(x, A) = \sum_{A_i \in A} A_i(x) \quad (1)$$

$$\text{Convex-Fusion}(x, A) = \sum_{A_i \in A} \frac{A_i(x)}{|A|} \quad (2)$$

$$\text{Average-Fusion}(x, A) = \hat{A}(x),\ \hat{A} = \sum_{A_i \in A} \frac{A_i}{|A|} \quad (3)$$

4. EXPERIMENTAL SETTINGS

4.1. Model Configuration

The RNN-T model used for the experiments contains 73.6M parameters with a 12-layer encoder, 2-layer decoder, feed-forward network with tanh non-linear activation as the joint network, and softmax output non-linearity. No external language model for shallow fusion or rescoring is used. We use Conformer [21] as our choice of the encoder. Denoting d as the model dimensionality, f as the feed-forward sub-layer dimensionality, h as the number of self-attention heads, and c as the kernel size of the convolutional sub-layer, our choice for the Conformer architecture was $d = 512$, $f = 1024$, $h = 8$, and $c = 32$. The decoder comprised of 1024 Long-Short Term Memory [22] units. The encoder and decoder outputs are summed as the input to the joint network with the size of 512. The output layer size is 4001 with 4000 word-pieces and a blank-symbol. The vocabulary was constructed on a subset of $\mathcal{D}$ using a uni-gram language model [23]. The intermediate size d_a for the adapter layers are set to half the model dimension d. Hence, the adapter layers inserted in the decoder have a size of 512 and the layers in the encoder have a size of 256. If we have more than one adapter layers to insert into the encoder or the decoder, we always insert them from the top.

The base model was trained for 170 epochs with 5000 steps per epoch with a total batch size of 1024 samples/step using 32 GPUs. It was optimized using Adam [24] with a learning rate of 6.55×10^{-5} and a warm-up decay learning-rate scheduler described in [21]. All adapter runs are trained on 32 GPUs with a total batch size of 512 samples/step and a constant-learning rate of 1×10^{-4}.

4.2. Data

For the experiments, we used English in-house dataset of de-identified recordings from voice-controlled far-field devices. The possible use of TTS data is not considered here. The training data for the base model consists of human-transcribed 47.7k hours of audio. We took the test data corresponding to period of the last two months of $\mathcal{D}$ to use as the average data to measure the level of degradation on general requests for each adapter configuration.

For the targeted updates, we identified 20 new-words with enough training examples that were not included in $\mathcal{D}$. There are limited number of training examples for each word with the average of 36.5 utterances and the minimum of 11. The new words consists of song names, names of public personnel, newly trending food names, etc. We considered two scenarios: 1) a *Large Batch* setup where we have three updates with six to seven words per round and 2) a *Small Batch* setup with 10 batches with two words each. Note that the updates need not be necessarily done in a sequential manner and combining all permutations of the adapters is possible. The performance of adapters for each batch are evaluated on a corresponding testset containing the same words but with unseen utterances.

5. RESULTS

For all experiments, we use Recall-5 (higher is better) by considering the top-5 hypotheses generated by the model to measure how well the updated model is able to recognize the new words. This is a more realistic measure than considering only the most likely hypothesis since the RNN-T outputs are processed with an external language model in a production setting which could also be updated with the new words. We only report on the new words the model has been trained on. For example, when combining A_i and A_j, only the words included in B_i and B_j (denoted as $B_{i,j}$) are included in the evaluation. We also track the relative word error rate reduction (rWERR) (higher is better) on the average data with respect to the base model to measure the level of degradation that had resulted by the introduction of the adapters.

Fusion	Weights (%)	Recall-5 (%)							rWERR (%)						
	(1-w,w)	+B_1	+B_2	+B_3	+$B_{1,2}$	+$B_{1,3}$	+$B_{2,3}$	+$B_{1,2,3}$	+B_1	+B_2	+B_3	+$B_{1,2}$	+$B_{1,3}$	+$B_{2,3}$	+$B_{1,2,3}$
M	-	35.8	28.6	20.5	33.2	29.4	24.2	29.4	-	-	-	-	-	-	-
Sum	$(99, 1)$	88.7	91.4	97.4	89.7	92.3	94.7	92.1	0.0	-0.1	-0.1	-0.2	-0.2	-0.2	-0.3
	$(95, 5)$	94.3	97.1	100	95.4	95.6	98.7	96.0	-0.3	-0.3	-0.2	-0.7	- 0.5	-0.6	-1.0
	$(90, 10)$	94.3	97.1	100	95.4	97.7	98.7	96.8	-0.3	-0.5	-0.6	-1.2	-1.0	-1.1	-1.8
	$(50, 50)$	96.2	97.1	100	96.5	96.7	100	97.6	-2.5	-3.0	-2.8	-6.3	-5.9	-6.3	-10.1
	$(0, 100)$	96.2	97.1	100	96.5	94.5	93.2	75.3	-27.5	-38.0	-24.5	-112.4	-103.0	-118.2	-257.5
Convex	$(99, 1)$	-	-	-	56.8	64.1	63.7	47.3	-	-	-	-0.2	0.0	-0.1	0.2
	$(95, 5)$	-	-	-	72.6	72.8	78.3	57.5	-	-	-	-0.2	-0.1	-0.1	-0.2
	$(90, 10)$	-	-	-	73.8	77.1	79.8	61.5	-	-	-	-0.2	-0.2	-0.3	-0.2
	$(50, 50)$	-	-	-	76.0	81.4	81.2	66.2	-	-	-	-1.5	-1.2	-1.4	-1.0
	$(0, 100)$	-	-	-	76.9	71.7	75.6	59.9	-	-	-	-12.7	-11.8	-13.2	-8.5
Average	$(99, 1)$	-	-	-	40.0	39.2	32.5	33.2	-	-	-	0.0	0.0	-0.1	0.0
	$(95, 5)$	-	-	-	45.0	45.8	33.8	34.0	-	-	-	-0.1	-0.1	-0.1	-0.1
	$(90, 10)$	-	-	-	40.0	45.8	32.5	35.6	-	-	-	-0.0	-0.1	-0.1	-0.1
	$(50, 50)$	-	-	-	50.3	54.4	41.9	38.0	-	-	-	-0.7	-0.5	-0.7	-0.4
	$(0, 100)$	-	-	-	52.6	51.2	36.5	37.2	-	-	-	-6.0	-5.0	-6.2	-3.2

Table 1. Results for the targeted updates with the Large Batch setup are summarized. Each update consists of training on the utterances with 6 to 7 new words. Recall-5 in % (higher is better) is shown to evaluate the recognition of the new words. rWERR in % (higher is better) with respect to the base model M on the average data is tracked to measure the level of degradation on usual user requests. All three fusion methods are equivalent with only one adapter, thus we only list the numbers for one adapter (i.e. +B_1, B_2, and B_3) in the sum-fusion.

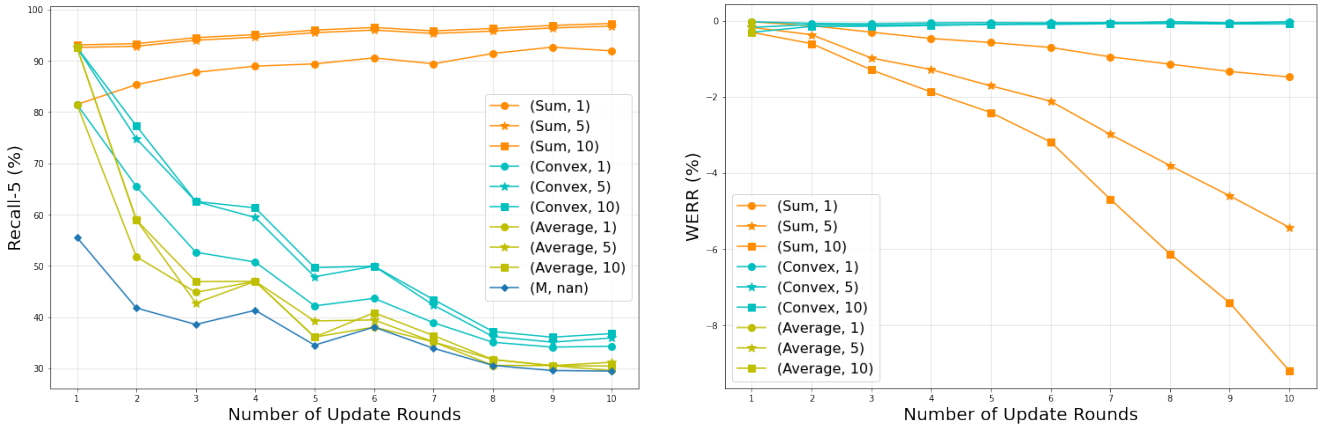
5.1. Targeted Updates with Large Batches

We first evaluate in the Large Batch setup on all possible permutations of the adapters. Denoting l_d as the number of adapter layers in the decoder and l_e as the number of adapter layers in the encoder, we use $l_d = 1$ and $l_e = 1$. While other configurations are possible, we use this setup as it performs the best compared to the other configurations. The impact of having different configurations are presented later in Section 5.3. We consider five different data replay strategies, $(0, 100)$, $(50, 50)$, $(90, 10)$, $(95, 5)$ and $(99, 1)$ for every adapter fusion method. All adapters with data replay are trained for 750 steps which was the best choice from doing a grid search on $\{250, 500, 750, 1000\}$. Chosen in the same way, the adapters without data replay are trained for 250 steps.

Table 1 summarizes the results. We see that data replay is crucial in preventing the adapters from degrading on the average data. Without data replay, the adapter trained on B_1 show significant degradation with 27.5% rWERR. Also, there is a performance trade-off between the new and the average data in using different data replay strategies. More sampling weight given to the new words lead to a better recognition of the new words but it also results in more degradation on the average data. Furthermore, we see that data replay is essential for minimizing interference between the adapters as the models without data replay show sharp degradation at the introduction of the third adapter. For example, the sum-fusion without data replay scores better than 93.2% Recall-5 with two adapters similar to the data replay models, but scores significantly worse at 75.3% with three adapters.

The sum-fusion with data replay shows no signs of interference when multiple adapters are combined and is able to emit the new word almost every time. This is contrary to the convex and the average-fusion where the performance on the new words degrade as more adapters are combined. The average-fusion in particular leads to a significant drop in performance. With three adapters, it scores almost as low as the performance of the base model. The trend is reversed on the average data. Combining more adapters lead to more averge data degradation for the sum-fusion. For example, the data replay $(90, 10)$ model loses around 0.6% rWERR going from two to three adapters. For the other two fusion methods, the performance improves as more adapters are introduced.

This result demonstrates that by carefully considering the data-replay performance trade-off, we are able to combine multiple independently trained adapters through the sum-fusion with minimal degradation on the average data. For example, in this setup we find the data replay (95,5) model to be the optimal operating point. This model scores around 95% Recall-5 on the new words with less than 1% rWERR degradation on the average data.



(a) Performance on the target batches.

(b) Performance on the average data.

Fig. 2. Results for the targeted updates with the Small Batch setup are summarized. Same as Large Batch, we track Recall-5 % on the new words to measure the recognition performance and rWERR % on the average data to measure the level of general degradation. For both figures, the legend indicates (Fusion-method, w) where w is the sampling ratio used for the new utterances in data replay.

5.2. Targeted Updates with Small Batches

Next, we evaluate in the Small Batch setup to understand the impact of having more specialized adapters. This setup has two new words per batch instead of seven. Also, since there are three times as many batches compared to the previous setup, we can further elucidate the role of interference between the adapters. Considering all permutations is not possible in this setup, thus we simply assume that each batch arrives sequentially over time from B_1 until B_{10}. Also, in this setup, the functional relationship between degradation on average data and the number of adapters inserted can be understood. Here, we only consider the most-promising data replay strategies, namely $(99, 1)$, $(95, 5)$ and $(90, 10)$ with the same adapter configuration as before. All adapters are trained for 250 steps instead of 750 steps as training data is smaller.

Figure 2 shows the results. For the sum-fusion, it is possible to combine 10 adapters without any degradation on recognition ability of the new words. This is contrary to the other two methods which quickly degrades below 50% recall after five adapters. This suggests that adapters do not interfere with each other when combined through the sum-fusion. We even see a continual improvement as more adapters are added for the sum-fusion, but this is most likely due to the simple fact that the first few batches were more difficult than the rest. Similar results to Large Batch on the average data can be seen. The degradation on the average data increases as more adapters are added while the opposite occurs for the other two methods.

Through having more updates, we can observe that for the sum-fusion, the functional relationship between the degradation on the average data and the number of targeted updates is quadratic. The rate of degradation is greater for the models with higher sampling ratio for the new data. This knowledge is important as we can properly understand the trade-off between the requests pertaining to the hot-fixes and the usual user-requests. Still, as in the previous setup, through a careful selection of the data replay strategy (i.e. $(99, 1)$ with the sum-fusion), it is possibly to recognize the new words with great accuracy and with minimal degradation ($<$2%).

l_e	l_d	Recall-5 (%)	rWERR (%)	Size (%)
1	0	91.3	-1.09	**0.36**
6	0	93.3	-1.26	2.14
12	0	95.7	-1.48	4.29
0	1	93.9	-1.55	1.43
0	2	93.5	-1.92	2.85
1	1	**96.5**	**-1.02**	1.78

Table 2. Results with different adapter configurations are shown. Data replay (95,5) in the Large Batch setup with all three adapters combined with the sum-fusion is used. l_e refers to the number of adapters in the encoder and l_d refers to the number of adapters in the decoder. Size refers to the number of parameters needed for each adapter configuration compared to the base model.

5.3. Adapter Configurations

In order to further understand the key components needed to make the on-the-fly adapter fusion successful, we evaluate in the Large Batch setup with different number of adapter layers in the encoder l_e and in the decoder l_d by combining all three adapters. We use the previously identified optimal set-

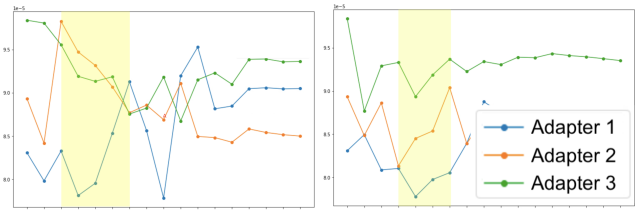
ting which is the data replay (95,5) with the sum-fusion. Out of 24 possible adapter configurations, we report only on 6 configurations as only a few adapters are needed to achieve a good result.

Table 2 summarizes the results. We see that the location of where the adapter layers are inserted is important. As previously mentioned, the best configuration is $l_e = 1$ and $l_d = 1$ which scores 96.5% on the new words and -1.02% on the average data. Only having the adapters in the encoder or only in the decoder does not work as well on both metrics. Also, we observe that adding more adapters in the decoder leads to significantly more degradation on the average data than the encoder. This indicates that the behaviour of the decoder is more important in deciding how much degradation we observe with the adapters. Finally, the best configuration requires less than two percent of the parameters of the base model. This is important as the small size makes the storage and deployment of adapter artifacts easier.

5.4. Illustration of the Adapter's Knowledge

In addition, we analyze the activation patterns of the trained adapters in response to the new words and the average data to gain more insight. We focus our attention to the adapters in the decoder as they are interpretive without alignment information. We use the adapters trained in the Large Batch setup with data replay of $(95, 5)$. We calculate the L2 norm of the output activations for every decoding-step for each adapter.

Figure 3 visualizes the results for two of the new words belonging to B_1 trained only by A_1. The highlighted segments are the word-pieces belonging to the new words. We see that the adapter has learned the boundaries of the new words where a local-minimum of activations can be seen on the non-boundary word-pieces. Other adapters also get activated but such patterns cannot be seen in their activations. This indicates that the adapter influences the RNN-T model through the relative differences between the generated activations and not by turning on and off for the new words and the usual requests respectively. For all the rest of the new words, a distinct activation pattern is also generated for each word. This elucidates the reason why the average-fusion and the convex-fusion leads to an inter-adapter interference while not for the sum-fusion. For the average-fusion, such distinct patterns of activations for the new words cannot be generated as the weights before the non-linear activation are averaged. For the convex-fusion, the distinct patterns are still generated, but they are scaled down by the number of active adapters. This leads to a less chance for the corresponding adapter to influence the output probability distribution as the relative differences between the activations become less prominent. Upon a qualitative analysis on the generate hypotheses, the convex-fusion indeed did generate hypotheses close that of the base model. Finally, this analysis result suggests a possible way to improve the adapter training to further minimize degradation on the average data by regularizing the adapters to also exhibit the on and off behaviour. We leave this study to future research.



(a) word 1 in B_1 (a) word 2 in B_1

Fig. 3. The L2 norms of the decoder adapter output activations for all three adapters for two different new words in Large Batch is shown. Each highlighted segment corresponds to the word-pieces of a new word. Both words are from B_1 Hence, only A_1 (i.e. Adapter 1) has seen the new words.

6. CONCLUSIONS

In this work, we investigated combining multiple adapters each trained on a small unique set of new words in real-world hot-fixing scenario. The proposed approach of fusing the adapters in parallel through a sum of its output activations is shown to recognize the new words with over 90% recall and a minimal degradation on the average data. Since the adapters are trained independently, each hot-fix can be implemented or taken out without affecting other hot-fixes.

7. REFERENCES

[1] Jinyu Li, Yu Wu, Yashesh Gaur, Chengyi Wang, Rui Zhao, and Shujie Liu, "On the comparison of popular end-to-end models for large scale speech recognition.," in *INTERSPEECH*. 2020, pp. 1–5, ISCA.

[2] Chung-Cheng Chiu, Anjuli Kannan, Rohit Prabhavalkar, Zhifeng Chen, Tara N. Sainath, Yonghui Wu, Wei Han, Yu Zhang, Ruoming Pang, Sergey Kishchenko, Patrick Nguyen, Arun Narayanan, Hank Liao, and Shuyuan Zhang, "A comparison of end-to-end models for long-form speech recognition.," in *IEEE-ASRU*, 2019, pp. 889–896.

[3] Yanzhang He, Tara N. Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, Qiao Liang, Deepti Bhatia, Yuan Shangguan, Bo Li, Golan Pundak, Khe Chai Sim, Tom Bagby, Shuo-Yiin Chang, Kanishka Rao, and Alexander Gruenstein, "Streaming end-to-end speech recognition for mobile devices.," in *IEEE-ICASSP*, 2019, pp. 6381–6385.

[4] Tara N. Sainath, Yanzhang He, Bo Li, Arun Narayanan, Ruoming Pang, Antoine Bruguier, Shuo-Yiin Chang, Wei Li, Raziel Alvarez, Zhifeng Chen, Chung-Cheng Chiu, David Garcia, Alexander Gruenstein, Ke Hu, Anjuli Kannan, Qiao Liang, Ian McGraw, Cal Peyser, Rohit Prabhavalkar, Golan Pundak, David Rybach, Yuan Shangguan, Yash Sheth, Trevor Strohman, Mirkó Visontai, Yonghui Wu, Yu Zhang, and Ding Zhao, "A streaming on-device end-to-end model surpassing server-side conventional model quality and latency.," in *IEEE-ICASSP*, 2020, pp. 6059–6063.

[5] Xianrui Zheng, Yulan Liu, Deniz Gunceler, and Daniel Willett, "Using synthetic audio to improve the recognition of out-of-vocabulary words in end-to-end ASR systems," in *IEEE-ICASSP*, 2021, pp. 5674–5678.

[6] Cal Peyser, Sepand Mavandadi, Tara N. Sainath, James Apfel, Ruoming Pang, and Shankar Kumar, "Improving tail performance of a deliberation E2E ASR model using a large text corpus," in *INTERSPEECH*. 2020, pp. 4921–4925, ISCA.

[7] Anjuli Kannan, Yonghui Wu, Patrick Nguyen, Tara N. Sainath, Zhifeng Chen, and Rohit Prabhavalkar, "An analysis of incorporating an external language model into a sequence-to-sequence model," 2017.

[8] Vijay Ravi, Yile Gu, Ankur Gandhe, Ariya Rastrow, Linda Liu, Denis Filimonov, Scott Novotney, and Ivan Bulyko, "Improving accuracy of rare words for rnn-transducer through unigram shallow fusion," 2020.

[9] Chao-Han Huck Yang, Linda Liu, Ankur Gandhe, Yile Gu, Anirudh Raju, Denis Filimonov, and Ivan Bulyko, "Multi-task language modeling for improving speech recognition of rare words," 2020.

[10] Junwei Liao, Sefik Emre Eskimez, Liyang Lu, Yu Shi, Ming Gong, Linjun Shou, Hong Qu, and Michael Zeng, "Improving readability for automatic speech recognition transcription," 2020.

[11] Yichong Leng, Xu Tan, Linchen Zhu, Jin Xu, Renqian Luo, Linquan Liu, Tao Qin, Xiang-Yang Li, Ed Lin, and Tie-Yan Liu, "Fastcorrect: Fast error correction with edit alignment for automatic speech recognition," 2021.

[12] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari, "End-to-end incremental learning.," in *ECCV*. 2018, vol. 11216, pp. 241–257, Springer.

[13] Zhizhong Li and Derek Hoiem, "Learning without forgetting.," *IEEE-TPAMI*, vol. 40, no. 12, pp. 2935–2947, 2018.

[14] Deepak Baby, Pasquale D'Alterio, and Valentin Mendelev, "Incremental learning for RNN-Transducer based speech recognition models," in *INTERSPEECH*. 2022, pp. 71–75, ISCA.

[15] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi, "Learning multiple visual domains with residual adapters," in *NIPS*, 2017, vol. 30.

[16] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly, "Parameter-efficient transfer learning for nlp," 2019.

[17] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych, "Adapterfusion: Non-destructive task composition for transfer learning," 2020.

[18] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig, "Towards a unified view of parameter-efficient transfer learning," 2021.

[19] Katrin Tomanek, Vicky Zayats, Dirk Padfield, Kara Vaillancourt, and Fadi Biadsy, "Residual adapters for parameter-efficient asr adaptation to atypical and accented speech," 2021.

[20] Anjuli Kannan, Arindrima Datta, Tara N. Sainath, Eugene Weinstein, Bhuvana Ramabhadran, Yonghui Wu, Ankur Bapna, Zhifeng Chen, and Seungji Lee, "Large-scale multilingual speech recognition with a streaming end-to-end model," 2019.

[21] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang, "Conformer: Convolution-augmented transformer for speech recognition," 2020.

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

[23] Taku Kudo and John Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv preprint arXiv:1808.06226*, 2018.

[24] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.