Improving End-to-End Speech-to-Intent Classification with Reptile

Yusheng Tian[*]*, Philip John Gorinski*

Huawei Noah's Ark Lab, London, UK

yusheng.tian@hotmail.com, philip.john.gorinski@huawei.com

Abstract

End-to-end spoken language understanding (SLU) systems have many advantages over conventional pipeline systems, but collecting in-domain speech data to train an end-to-end system is costly and time consuming. One question arises from this: how to train an end-to-end SLU with limited amounts of data? Many researchers have explored approaches that make use of other related data resources, typically by pre-training parts of the model on high-resource speech recognition. In this paper, we suggest improving the generalization performance of SLU models with a non-standard learning algorithm, Reptile. Though Reptile was originally proposed for model-agnostic meta learning, we argue that it can also be used to directly learn a target task and result in better generalization than conventional gradient descent. In this work, we employ Reptile to the task of end-to-end spoken intent classification. Experiments on four datasets of different languages and domains show improvement of intent prediction accuracy, both when Reptile is used alone and used in addition to pre-training.

Index Terms: spoken language understanding, intent classification, low-resource, Reptile

1. Introduction

Spoken language understanding (SLU) is a key component in assistive conversational agents. The goal of SLU is to infer users' intentions from speech utterances, such that actions can be taken accordingly to meet users' requests [1, 2]. Conventional SLU is a pipeline of automatic speech recognition (ASR) and natural language understanding (NLU). End-to-end SLU [3, 4, 5], on the other hand, directly maps audio to semantics without the intermediate ASR. Compared to the conventional pipeline SLU, it avoids error propagation, requires less computation, and has the potential to utilize information that is only present in speech but not in text.

Despite these advantages, training end-to-end SLU systems usually requires in-domain annotated audio data, which is very expensive and time-consuming to collect. Due to time and cost constraints, even for high-resource languages like English, current SLU datasets usually only contain less than 20 hours of speech data. Models trained on such limited data are at risk of over-fitting and may generalize poorly on unseen cases, e.g. a new speaker or a paraphrased command. This has motivated researchers to explore approaches that leverage other related data resources, typically by pre-training parts of the model using a high-resource ASR [6, 7, 8], whose language might even be different from that of the SLU, when the SLU target language itself is of low-resource [9]. Other approaches include multi-task learning with ASR if audio transcription data is available [5], and training SLU on synthetic speech [10]. While these methods all prove to be effective, we propose to improve SLU models' generalization capabilities by training with a non-standard learning algorithm: Reptile [11]. This allows us to improve the model performance when no additional data is available, or achieve a further improvement on top of pre-training.

Reptile was originally proposed as a first-order algorithm for model-agnostic meta-learning (MAML) [12]. Like the classic MAML, Reptile is well-established in few-shot learning and can optimize generalization, which is very desirable in low-resource settings. However, it can't be directly applied to end-to-end SLU: the original intention of Reptile is to learn a good parameter initialization from multiple source tasks (e.g. image classification of different categories) for fast adaption on a new but related task (e.g. image classification of a new group of categories), but for SLU we lack such source tasks. However, the formulation of Reptile gives rise to an interesting research question: Can we make use of Reptile to optimize generalization during the model training phase, instead of using it as a model initializer? In this paper, we argue that Reptile can be adapted to directly learn an SLU task by dropping the "task sampling" procedure in its original algorithm, resulting in better generalization than conventional gradient descent. We describe analysis justifying this argument by comparing the meta-gradients of the classic MAML and Reptile. We apply Reptile to end-to-end speech-to-intent classification and test its efficacy on 4 datasets of different languages and domains. Experiment results on all datasets show improvement of intent classification accuracy, both when Reptile is used alone and with pre-training.

The contributions of this paper are as follows: (i) we motivate the use of Reptile for single-task learning in low-resource settings; (ii) we adapt and employ Reptile to the task of end-to-end speech-to-intent classification; (iii) we show how our method helps boost performance on diverse datasets of 4 different languages and domains, both with and without pre-training.

2. Modeling End-to-End SLU

In this work we concentrate on end-to-end speech-to-intent classification. Given a speech command, we would like to find the most probable intent label from pre-defined categories. For example, in smart-home setting, the utterance *"Turn the heat down"* would be mapped to the intent `decrease_heat`. Almost invariably, current end-to-end models achieve this goal in two steps. First, an encoder maps the input sequence of audio signals (pre-processed acoustic features or raw waveforms) to a fixed-length utterance embedding. Then a decoder predicts a probability distribution over all possible intent labels conditioned on this utterance embedding. The intent label with the highest probability is selected as the output.

There has been some work on improving intent classification by utilizing a novel architecture: [13] replaced the soft-max classifier with a capsule network, and showed that it can make efficient use of limited training data. However, their model is a speaker-dependent system and makes use of pre-defined speech commands; [14] used a multi-label classifier instead of a single-

[*]Work conducted while this author was an intern at Huawei Noah's Ark Lab, London.

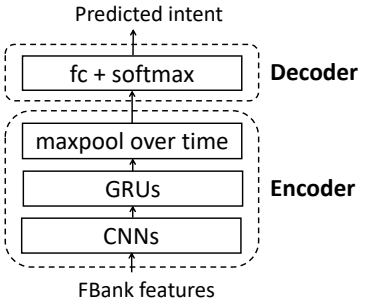


Figure 1: *Encoder-decoder architecture for SLU*

label classifier for intent prediction, but this architecture is tailored to a certain type of datasets whose intents are combinations of several slots.

Since our main focus in this paper is on the learning algorithm rather than the model architecture, we adopt a simple encoder-decoder architecture similar to that in [4] and [9], illustrated in Figure 1. The choice of a simple architecture also ensures that when comparing our models with SotA results – see section 5 – the relative gain of intent prediction accuracy comes from the training strategy rather than a more advanced architecture. Our model architecture is not restricted to a certain type of datasets, and can be augmented for experiments with pre-training, for example by replacing the bottom CNN layers with pre-trained ASR layers, as in [9]. Details of the model architecture and implementation are described in Section 4.

3. Reptile Learning

Reptile [11] is proposed as a first-order algorithm for MAML [12, 15], which aims to train a model on multiple source tasks, such that the resulting model can be fine-tuned on a new but related task with only a small amount of training examples. In other words, MAML learns an initialization, rather than a good model [16]. We argue that while this applies to the classic MAML algorithm, Reptile can be adapted to learn a low-resource target task (in our case, spoken intent classification) and improve the model's generalization performance. To explain why this works, we will first give a brief description of Reptile and its origin, the classic MAML, and then introduce the adapted Reptile for SLU.

3.1. Reptile as a MAML algorithm

The classic MAML accomplishes the meta-learning goal by directly incorporating the fast adaptation process into its objective function:

$$\min_{\theta} \mathbb{E}_{\tau \sim p(\mathrm{T})} \left[L_\tau \left(U_\tau^k \left(\theta, D_\tau^{tr} \right), D_\tau^{test} \right) \right] \tag{1}$$

We can interpret the objective as: for any task τ that is sampled from a distribution of tasks $p(\mathrm{T})$, search for an initialization of model parameters θ, such that after k gradient descent updates (e.g. SGD, Adam) on the training set D_τ^{tr} (usually of a small size), the model with updated parameter vector $U_\tau^k \left(\theta, D_\tau^{tr} \right)$ has a minimum loss L_τ on the test set D_τ^{test}. In other words, we are using U_τ^k to find a good *initialization* of θ for future training, but we are not training θ itself. Also note that D_τ^{tr} and D_τ^{test} in the objective function (1) are different from the conventional training and test set. They refer to the meta training and test sets, which form the whole training data.

MAML solves the above optimization problem through stochastic gradient descent, i.e. by repeatedly sampling a task τ and updating θ with

$$\theta \leftarrow \theta - \alpha g_{MAML} \tag{2}$$

where α is the meta step size, and

$$g_{MAML} = \nabla_\theta \left[L_\tau \left(U_\tau^k \left(\theta, D_\tau^{tr} \right), D_\tau^{test} \right) \right] \tag{3}$$

The above meta update involves a second-order gradient and might be computationally expensive. Reptile simplifies the meta gradient as (4), which is very convenient to compute:

$$g_{Reptile} = \theta - U_\tau^k \left(\theta, D_\tau \right) \tag{4}$$

and similar to (2), we now update model parameters θ with

$$\theta \leftarrow \theta - \alpha g_{Reptile} = \theta + \alpha (U_\tau^k \left(\theta, D_\tau \right) - \theta) \tag{5}$$

3.2. Reptile for SLU

We can see from (4) that Reptile entirely eliminates the train/test split for meta optimization, therefore there is not a meaningful objective function as in (1) that corresponds to the meta gradient of Reptile. This means that, as opposed to classic MAML, Reptile is not strictly defined for learning a model initialization. In addition, from (5) we can see that Reptile still pushes model parameters towards the trained weights of standard (e.g. SGD, Adam) training algorithms, just at a slower pace. This suggests that unlike the classic MAML, the resulting model of Reptile learning is still a good model itself for the tasks that it has been trained on. On the other hand, like the classic MAML, the meta gradient of Reptile ($g_{Reptile}$) contains some terms that maximize the inner product between gradients computed at different steps (e.g. different mini-batches). This in turn encourages gradients at different steps to point to similar directions, as illustrated in Figure 2. We refer readers to [11] for a detailed theoretical proof of this. By encouraging gradients at different steps to point to similar directions, Reptile promotes within-task generalization. This is very beneficial for training SLU with only limited amounts of data.

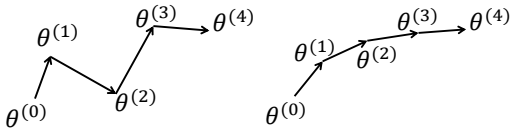
When we apply Reptile to training end-to-end SLU, we only learn a single task. Therefore there is no need to repeatedly sample a task like the original Reptile, and the algorithm can be simplified as Algorithm 1. We have eliminated the subscript τ because it always refers to the same task of spoken intent prediction. The basic unit of iteration in Reptile training are Episodes (operations within the While loop, Algorithm1), comprising k Epochs training plus one interpolation. Details of the training procedure are described in Section 4.

Our adapted Reptile algorithm is very similar to that of the Lookahead Optimizer in [17]. The Lookahead algorithm is not based on Reptile, but also contains k steps forward and 1 interpolation step. The major difference between these two methods is that in the adapted Reptile, the interpolation happens over the full data (k epochs), while the Lookahead Optimizer operates on a much smaller scale (around 10 mini-batches). The Lookahead optimizer also does not target low-resource tasks, which is a direct motivation for our adaptation of Reptile.

4. Experimental Setup

4.1. Datasets

We use the following four SLU datasets. (1) fluent speech commands (FSC) [8]: English speech commands related to personal



(a) *without Reptile* (b) *with Reptile*

Figure 2: *An illustration of Reptile encouraging gradients computed at different steps to point to similar directions.*

Algorithm 1 Reptile for end-to-end SLU

Require: k: number of inner epochs
Require: α: step size
1: Randomly initialize θ
2: **while** not done **do**
3: Compute $\theta^{'} = U^{k}(\theta, D_{train})$, denoting k gradient descent (SGD/Adam) updates over the entire training set D_{train}, i.e. k epochs.
4: Update $\theta \leftarrow \theta + \alpha(\theta^{'} - \theta)$
5: **end while**

assistant services. (2) Grabo [18]: speech commands in Dutch for controlling robot movement. (3) Tamil [19]: speech commands in the low-resource language Tamil for requesting banking services. (4) CN: internally collected Mandarin Chinese speech commands for operating a mobile phone. Audio from different speakers was recorded using the same mobile phone. Statistics of the four datasets are summarized in Table 1.

We use the default train/validation/test split of the FSC dataset. The Grabo dataset is designed for developing speaker-dependent systems, with each speaker having recorded 36 different commands with 15 repetitions. For each speaker, we randomly select 2 recordings of each command for training, 4 for validation, and use the remaining 9 recordings for testing. On the Tamil dataset we perform 5-fold cross-validation as in [19]. We split the dataset into 5 parts, such that each subset has an approximately equal number of samples, and no speaker is spread across different subsets. Each fold is repeated 3 times to allow different train/validation combinations. For the CN dataset we have 6 main speakers. Each of them has contributed over 1200 recordings. We use recordings from these speakers as the training data. We randomly select 24 out of the remaining 36 speakers and use their recordings (around 4500 audio clips) as the held-out set for testing. The rest of the data (around 2400 audio clips) forms the validation set.

4.2. Model architecture and training

Our model takes 39-dimensional log Mel-Filterbank feature plus energy as input, computed with a window size of 25 ms and a shift of 10 ms. The encoder consists of a stack of 2-layer 2-D CNN and 2-layer uni-directional GRU. For each CNN layer, we stride with a factor of 2 along time to reduce the sequence length, and apply batch normalization as well as dropout. For all datasets we use GRU cells of 128 hidden units, except for the CN dataset, where we set the number to 256 as it has significantly more target intents. The output of the GRU layers is fed into a maxpooling layer over all time steps in order to extract fixed-length utterance embeddings. The decoder is a simple softmax classifier with only 1 hidden layer.

Table 1: *Statistics of the SLU datasets.* * *indicates the data is for each speaker.*

Dataset	FSC	Grabo	Tamil	CN
#intents	31	36*	6	130
#phrases	248	36*	31	13756
#utts	30043	540*	400	14771
#speakers	97	10	40	42

We investigate the effectiveness of Reptile both when the model is trained from scratch, and when utilizing pre-training. For pre-training, we follow the strategy proposed in [9] which incorporates layers extracted from pre-trained acoustic models to the input of SLU models, and supports cross-lingual transfer learning. For all SLU datasets, regardless of the spoken language, we use the hidden representation of a pre-trained end-to-end English phoneme recognizer provided in the source code of [8] as our model's input, and directly feed it to the GRU layers. There is no need to keep the CNN layers in our model when pre-training is applied, because all its functionality can be achieved by the pre-trained ASR layers.

We use the Adam optimizer [20] with a learning rate of 0.001 both when training with Reptile and without. For Reptile learning, we set the inner epoch iteration to $k = 5$ for all datasets. We empirically set a step size $\alpha = 0.3$ for Grabo, and $\alpha = 0.1$ for all other datasets. Within each epoch, we train our model on mini-batches of data, and experiment results show that using a small batch size helps stabilize training when we have extremely low-resource data (e.g. we used a batch-size of 8 for the Grabo and Tamil dataset). We train with Reptile on the training set. After each episode, we record the intent prediction accuracy on the validation set to examine model's generalization performance, and decide whether to stop training. During each training, we employ early stopping with patience of 10.

5. Results

5.1. Effectiveness of Reptile

Table 2 shows the intent classification accuracies of our models under all training conditions for all four SLU datasets, as well as state-of-the-art results for the three freely available datasets. Our results are the average of 10 speakers for the Grabo dataset, the average of 5 folds for the Tamil dataset, and the average of 3 runs for the FSC and CN datasets. It can be seen that Reptile gives improvement on all datasets, both with and without pre-training. The lowest accuracy, but also the most significant improvement is observed on the Chinese dataset, perhaps because this dataset contains over 10,000 different expressions in total while only having 6 speakers in the training set. Note that it is not strictly fair to compare our results with those in the literature because they used different and often more elaborate architectures (e.g. [21] used a simple 2-layer CNN encoder, while [13] used a capsule network for classification), but we can see that with Reptile learning our model with its simple architecture achieves at least comparable intent classification accuracy to state-of-the-art results on the open datasets. The SotA result for the Tamil dataset without pre-training is confusingly low. This is because the number reported in the original paper is from an SVM model, and the authors did not employ a deep-learning model without pre-training.

To assess the significance of improvements gained through Reptile training on the small tested datasets, we carried out two-tailed paired t-tests on the multi-run experiment results of our

Table 2: *Intent prediction accuracy.* ***–pretrain*** *indicates no pre-training is applied,* ***+pretrain*** *indicates using pre-trained ASR layers. Rows labeled* ***Base*** *are for models trained with standard Adam optimization, while* ***Reptile*** *employs our adapted Reptile training. Where applicable,* ***SotA*** *shows previously published State of the Art results as reported in [1][14], [2][13], [3][19], [4][10], [5][21], and [*] indicates that results were read off a graph.*

		FSC	Grabo	Tamil	CN
–pretrain	Base	98.3	88.9	85.8	25.8
	Reptile	98.8	94.4	90.2	36.9
	SotA	98.6[1]	94.5[*2]	29.2[3]	–
+pretrain	Base	98.7	98.6	94.2	49.9
	Reptile	99.2	98.9	94.5	55.2
	SotA	99.1[4]	–	81.7[5]	–

Table 3: *Paired t-test results between baseline and Reptile models.* ***–pretrain*** *indicates no pre-training is applied,* ***+pretrain*** *indicates using pre-trained ASR layers.*

	FSC	Grabo	Tamil	CN
-pretrain	0.012	0.023	0.0073	0.0051
+pretrain	0.0042	0.026	0.014	0.0013

simple model when trained with and without Reptile. The p-value results on all datasets are within 0.05, as summarized in Table 3, showing that the performance difference between the models trained with Reptile and the baseline is statistically significant. Since we do not have predictions of the SotA systems in all settings, and our focus is on the efficacy of Reptile when compared to standard training, we do not make claims about statistical differences between our best models and SotA results.

5.2. Impact of Training Data Size

Figure 3 shows the impact of decreasing the amount of training data on intent classification performance for a model trained with and without Reptile. The experiment is carried out on the FSC dataset because it is the only one of the open datasets that is of large enough size. We vary the amount of training data by gradually removing speakers along with their recordings from the training set. We can see that the model trained with Reptile consistently outperforms the baseline model, and the gap between the two models increases as the training data size shrinks. When only 10% of the original data are used for training, the absolute accuracy difference between Reptile and the baseline reaches 1% with pre-training, and over 3% without pre-training. This demonstrates that Reptile learning helps improve the model generalization when there is only limited speaker variation present in the training data. It also shows that under extreme low-resource settings, Reptile can bring appreciable intent prediction improvement even when no pre-training is applied.

5.3. Learning Curves

The difference between Reptile learning and the conventional gradient descent (in our case Adam) is also reflected in their learning curves. Figure 4 compares learning curves of the two optimization approaches on the Tamil dataset, without pre-training. Since the basic unit of iteration in Reptile training is an episode, which consists of k epochs and 1 interpolation, we cannot compare the i^{th} Reptile episode to the i^{th} epoch of SGD training. But when looking at their learning curves as a whole, we can see that the learning curve of Reptile is much smoother than the learning curve of Adam, with the accuracy on the validation set growing steadily. The model trained with Reptile also reaches a higher steady point of accuracy than the baseline. Similar patterns can also be observed for the other two low-resource datasets: Grabo and CN, sometimes even when pre-training is applied. This conforms with the analysis in Section 3 that Reptile encourages gradients computed at different steps to point to similar directions, promoting generalization.

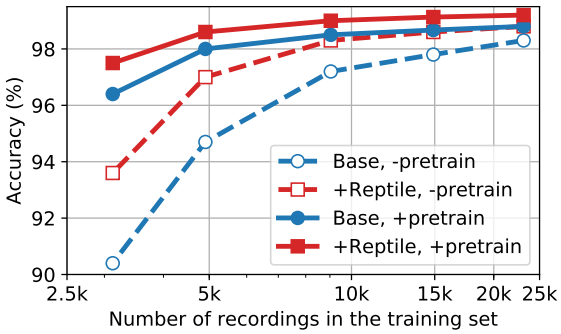


Figure 3: *Test set intent prediction accuracy w.r.t the size of training data (FSC dataset, with pre-training).*

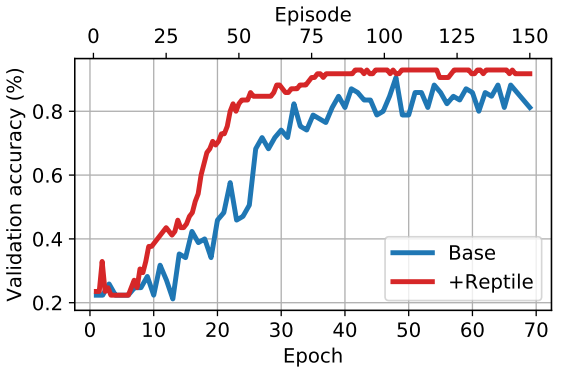


Figure 4: *Learning curves of intent prediction accuracy on the validation set (Tamil dataset, w/o pre-training).*

6. Conclusions and Future Work

In this paper, we applied Reptile to training low-resources, end-to-end SLU, in order to cope with the data scarcity problem. We tested Reptile on various SLU datasets of different languages and domains. Experimental results demonstrate that Reptile improves model generalization, and helps the model to better deal with speaker variations than conventional gradient descent. We should point out that under very low-resource settings pre-training is still the most effective way of bootstrapping model performance, and Reptile can only act as an assistant rather than a replacement of pre-training. In this paper we focus on intent classification. In future work we would like to extend Reptile learning to other SLU tasks such as slot filling.

7. References

[1] Y.-Y. Wang, L. Deng, and A. Acero, "Spoken language understanding," *IEEE Signal Processing Magazine*, vol. 22, no. 5, pp. 16–31, 2005.

[2] G. Tur, D. Hakkani-Tür, and L. Heck, "What is left to be understood in atis?" in *2010 IEEE Spoken Language Technology Workshop*. IEEE, 2010, pp. 19–24.

[3] Y. Qian, R. Ubale, V. Ramanarayanan, P. L. Lange, D. Suendermann-Oeft, K. Evanini, and E. Tsuprun, "Exploring asr-free end-to-end modeling to improve spoken language understanding in a cloud-based dialog system," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2017, Okinawa, Japan, December 16-20, 2017*, 2017, pp. 569–576.

[4] D. Serdyuk, Y. Wang, C. Fuegen, A. Kumar, B. Liu, and Y. Bengio, "Towards end-to-end spoken language understanding," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5754–5758.

[5] P. Haghani, A. Narayanan, M. Bacchiani, G. Chuang, N. Gaur, P. Moreno, R. Prabhavalkar, Z. Qu, and A. Waters, "From audio to semantics: Approaches to end-to-end spoken language understanding," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 720–726.

[6] Y.-P. Chen, R. Price, and S. Bangalore, "Spoken language understanding without speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6189–6193.

[7] S. Ghannay, A. Caubrière, Y. Estève, N. Camelin, E. Simonnet, A. Laurent, and E. Morin, "End-to-end named entity and semantic concept extraction from speech," in *2018 IEEE Spoken Language Technology Workshop, SLT 2018, Athens, Greece, December 18-21, 2018*, 2018, pp. 692–699.

[8] L. Lugosch, M. Ravanelli, P. Ignoto, V. S. Tomar, and Y. Bengio, "Speech Model Pre-Training for End-to-End Spoken Language Understanding," in *Proc. Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, September 15-19 2019*, 2019, pp. 814–818.

[9] S. Bhosale, I. Sheikh, S. H. Dumpala, and S. K. Kopparapu, "End-to-End Spoken Language Understanding: Bootstrapping in Low Resource Scenarios," in *Proc. Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, September 15-19 2019*, 2019, pp. 1188–1192.

[10] L. Lugosch, B. Meyer, D. Nowrouzezahrai, and M. Ravanelli, "Using speech synthesis to train end-to-end spoken language understanding models," *arXiv preprint arXiv:1910.09463*, 2019.

[11] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *CoRR*, vol. abs/1803.02999, 2018.

[12] C. Finn, "Learning to learn with gradients," Ph.D. dissertation, UC Berkeley, 2018.

[13] V. Renkens and H. V. hamme, "Capsule networks for low resource spoken language understanding," in *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, 2018, pp. 601–605.

[14] E. Palogiannidi, I. Gkinis, G. Mastrapas, P. Mizera, and T. Stafylakis, "End-to-end architectures for asr-free spoken language understanding," *arXiv preprint arXiv:1910.10599*, 2019.

[15] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1126–1135.

[16] J. Gu, Y. Wang, Y. Chen, V. O. K. Li, and K. Cho, "Meta-learning for low-resource neural machine translation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3622–3631. [Online]. Available: https://www.aclweb.org/anthology/D18-1398

[17] M. R. Zhang, J. Lucas, G. Hinton, and J. Ba, "Lookahead optimizer: k steps forward, 1 step back," in *33rd Conference on Neural Information Processing Systems, December 8-14, Vancouver, Canada*, 2019.

[18] V. Renkens, S. Janssens, B. Ons, J. F. Gemmeke, and H. V. hamme, "Acquisition of ordinal words using weakly supervised NMF," in *2014 IEEE Spoken Language Technology Workshop, SLT 2014, South Lake Tahoe, NV, USA, December 7-10, 2014*, 2014, pp. 30–35.

[19] Y. Karunanayake, U. Thayasivam, and S. Ranathunga, "Transfer learning based free-form speech command classification for low-resource languages," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 288–294. [Online]. Available: https://www.aclweb.org/anthology/P19-2040

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[21] Y. Karunanayake, U. Thayasivam, and S. Ranathunga, "Sinhala and tamil speech intent identification from english phoneme based asr," in *2019 IEEE International Conference on Asian Language Processing, IALP 2019, Shanghai, China, November 15-17, 2019*, 2019.