Local-to-global learning for iterative training of production SLU models on new features

Yulia Grishina and **Daniil Sorokin**
Amazon Alexa AI, Berlin, Germany
{yuliag,dsorokin}@amazon.com

Abstract

In production SLU systems, new training data becomes available with time so that ML models need to be updated on a regular basis. Specifically, releasing new features adds new classes of data while the old data remains constant. However, retraining the full model each time from scratch is computationally expensive. To address this problem, we propose to consider production releases from the curriculum learning perspective and to adapt the local-to-global learning (LGL) schedule (Cheng et al., 2019) for a neural model that starts with fewer output classes and adds more classes with each iteration.

We report experiments for the tasks of intent classification and slot filling in the context of a production voice-assistant. First, we apply the original LGL schedule on our data and then adapt LGL to the production setting where the full data is not available at initial training iterations. We demonstrate that our method improves model error rates by -7.3% and saves up to 25% training time for individual iterations.

1 Introduction

In many real-world NLP systems with ML models, new data becomes available with time and there is a need to refresh the model (Diethe et al., 2018). In some cases it is a passive flow, when new data arrives due to the properties of the application (e.g. daily search queries) or an active act of collecting new data to be incorporated into the system (e.g. a new feature). In this paper, we regard the use case of an active extension of data to incorporate a new customer-facing feature into a production NLP model. We consider a Spoken Language Understanding (SLU) model that is used to interpret user requests in a commercial task-oriented voice-assistant. The model is a joint intent classification (IC) and slot filling (SF) architecture that is used to process utterances in a single domain.[1] We select one data-rich domain for our experiments, Music, and construct a scenario when an existing IC+SF model is extended with a new user-facing feature that comprises a set of intents and slots to be now recognized by the model.

It is conventional to re-train the original ML model on a combination of the old training data and the additional data for the new feature, starting from the same randomly initialized or pre-trained architecture as the previous time. The practitioners tend to use pre-trained models (language modeling and transfer learning are widely used here) to improve the generalization performance of the model. It seems logical also to re-use the previous iteration of the model trained on the old data in the previous model release to warm-start the next iteration. This could result in a reduced training time and a smaller computational and environmental footprint of the model updates in a scenario where new features are added regularly. Yet, in practice it is usually considered 'safer' to start training from scratch or from the same general-purpose pre-trained model every time, the main concern being that repeated warm-starting would lead to overfitting and poorer generalization (Ash and Adams, 2020).

Re-training the same model architecture on nearly the same data with minimal changes, but extending the output space with a new class is a unique problem for industry applications. Many previous works on continual learning have focused on learning from a continuous stream of data (Biesialska et al., 2020) or on an incremental learning of new tasks (Kanwatchara et al., 2021) and languages (Castellucci et al., 2021). Payan et al. (2021) discuss a single-task continual learning setup and simulated a passive data extension

[1] Intent classification model determines the intent class the query belongs to (e.g., *PlayMusic*) and slot filling is responsible for identifying slot instances in the query (e.g., *SongName*).

scenario where new examples are coming in for all output classes on a public dataset. Similarly, Ash and Adams (2020) evaluate a batch-learning setup, where each model iteration is warm-started from the previous step and the whole training data is always available, while some new data is added across all output classes in each batch. In our scenario, we consider active data extension for new features and we do not restrict the access to the old training data. We rely on an offline training paradigm, where each model release is trained on the latest batch of data until convergence.

If each release adds data for new features with the old data being constant, we can view a sequence of model releases as a subclass of curriculum learning, a machine learning paradigm that aims to arrange training data into a meaningful order to improve model training. In our scenario, the data is arranged by feature. Cheng et al. (2019) describe a local-to-global learning (LGL) schedule for a statistical model that starts with fewer output classes and adds more classes with each iteration. We build upon their results in our work, but remove their assumption that the whole data is available in the first iteration.

In this paper, we repeatedly apply a warm-start for training a set of subsequent IC+SF model releases, each one being extended with a new set of features. We define a single feature as a set of new intents and slots to be recognized by the model that are added to the output space. We focus on a real-life production setting and report results on a dataset sampled from a commercial German voice assistant.[2] As our main contribution, we show that warm-start is an effective strategy to reduce training time for later model releases and improve overall model performance in a scenario when the added training data pertains to new features only.

2 Related work

2.1 Spoken language understanding

Recent research in the field of SLU has made significant advancements through the application of deep learning (Mesnil et al., 2013) and the joint modeling of IC and SF (Zhang and Wang, 2016; Chen et al., 2019; Louvan and Magnini, 2020). Semi-supervised learning and paraphrasing are frequently applied to bootstrap new features, overcome the class imbalance problem and improve the overall SLU performance (Cho et al., 2019; Sokolov and Filimonov, 2020). These methods often rely on the assumption that the number of classes is static, while in a real production SLU system, new classes are added on a regular basis, affecting the target data distribution. In contrast, in this work, we propose to focus on the learning schedule of a model that benefits directly from the increasing number of classes and thus can be adapted to the real-world scenario, where new features are added to the system iteratively.

2.2 Local-to-global learning schedule

The main idea of local-to-global learning (LGL) schedule used in this work is to gradually train a neural network starting with a few output classes and subsequently extending to more classes. It was first introduced in the work of Cheng et al. (2019), who applied it to a computer vision problem. LGL does not require any additional annotated training data, instead it utilises the entire training set in each iteration, but only the data for the classes that are being learned in this iteration is annotated. The data for the rest of the classes is added masked (unlabeled). In each LGL iteration, a set of new classes is added and the model weights are transferred from the previous iteration (see Figure 1).

Cheng et al. (2019) compare the LGL schedule to other curriculum learning and self-paced learning strategies. A typical curriculum learning approach relies on prior knowledge about the data to define a training schedule, such as, for example, the input length (Tay et al., 2019). Self-paced learning alleviates the requirement for prior knowledge by assigning a weight to each training sample based on model's loss (Kumar et al., 2010). Yet, it introduces additional model passes to compute the per-sample loss during training and makes self-paced learning approaches challenging to optimize (Cheng et al., 2019). LGL defines a learning schedule based on the target output classes, by focusing the early stages of training only on a subset of classes. We propose to view feature expansion in a production SLU system as a special case of curriculum learning akin to LGL.

LGL can be also considered a form of a task specific pre-training strategy or transfer learning. Numerous transfer learning strategies were suggested for NLP problems (Ruder et al., 2019) and a complete overview is beyond the scope of our work. In that view, the final stage of LGL train-

[2]The data was de-identified prior to the experiment so that any user identifiable information was removed.

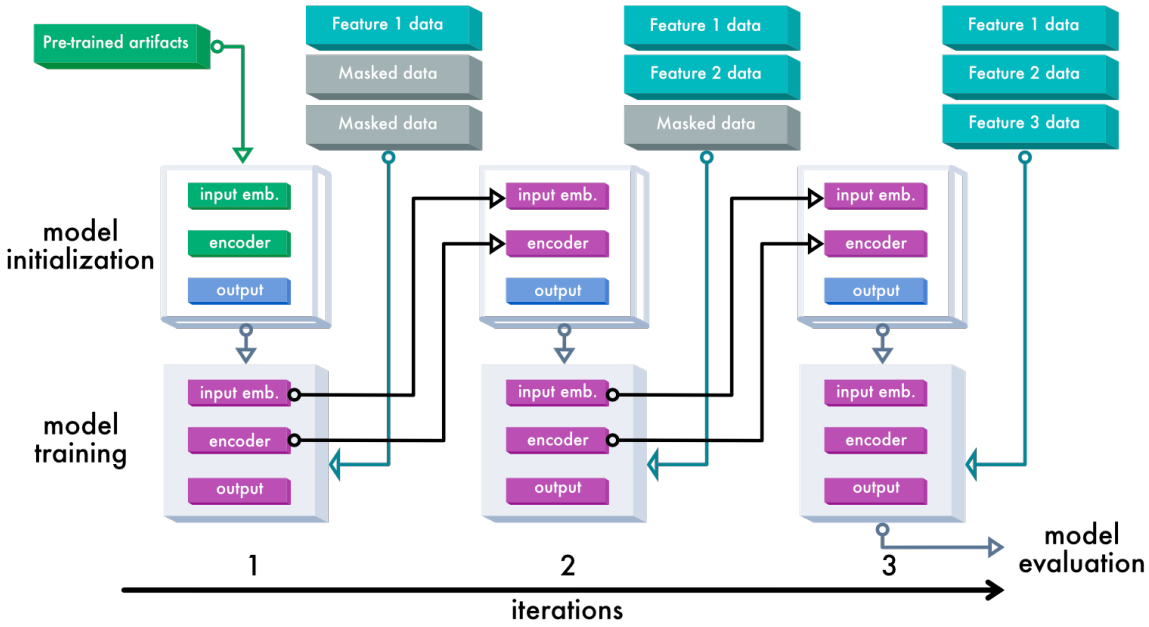

Figure 1: Local-to-Global model training set-up with 3 label batches (3 features) and hence 3 training iterations.

ing with complete annotations would correspond to the usual fine-tuning step. The preceding stages with a subset of classes are pre-finetuning steps, where pre-training is done repeatedly on the same task and with a reduced number of classes. The task-adaptive pre-training approach (Gururangan et al., 2020) uses a similar task to the target task to pre-train the model with a task-specific objective. Pruksachatkun et al. (2020) combine language model pre-training with task-specific pre-training and fine-tuning on the target task to test 110 pre-training task combinations. They conclude that it is still hard to predict, which task would be the most optimal for pre-training. From this perspective, LGL does not have this problem, as it pre-trains on the same task and the same dataset.

A sample LGL training set-up with 3 label batches and 3 training iterations is illustrated in Figure 1. In the first iteration the model is initialized from a pre-trained LM and the first batch of labels are unmasked in the training data, while the rest of data is left masked. In the next iteration, the embeddings and the encoder are initialized from the embeddings and the encoder of the previous iteration model. The second batch of labels is unmasked. In the final iteration, the embeddings and the encoder are initialized from the embeddings and the encoder of the second iteration model, while all three label batches are unmasked. After the last iteration the final model is exported and applied on the test set.[3]

In this work, we focus on applying LGL in a production SLU setup, where new models are released and new classes are added regularly. The experimental setup of Cheng et al. (2019) focuses on improving the final model performance on the full dataset and includes full (partially-masked) training data at each iteration. We first apply LGL to our internal data from a production SLU system. Second, to simulate a real-life situation, we modify this setup and conduct experiments where the data for new classes is not available at the early model training iterations. In Figure 1, this would correspond to not using the masked data batches.

3 Experimental setup

3.1 Dataset

We use a dataset sampled from a commercial German SLU system. The data was de-identified prior to the experiment (so that any user identifiable information was removed), and subsequently annotated across domains, intents and slots. For our experiment, we have selected Music domain, which contains mutually exclusive classes (intents), such as *PlayRadio* or *FindSoundtrack*. The evaluation set comes from the same distribution and was annotated in the same way. The distribution of relative frequencies of intents is typically a heavily skewed one; in the case of LGL, that can result in a large

[3] See Appendix A for an extended definition of LGL.

fraction of the annotated data being masked all at once. However, the main motivation behind LGL is that it is easier to learn fewer classes, while the amount of training data per class may vary (Cheng et al., 2019).

3.2 Model

We use an SLU architecture based on BERT for all of our experiments. Architectures based on pre-trained transformers have recently demonstrated the strongest performance on SLU tasks (Chen et al., 2019; Gaspers et al., 2020; Weld et al., 2021). The model consists of a pre-trained BERT encoder and an intent and slot decoders. The BERT encoder's outputs at sentence and token level are used as inputs for the intent and slot decoders, respectively. The intent decoder is a feed-forward network consisting of two dense layers and a softmax layer on top. The slot decoder uses a CRF layer on top of two dense layers to leverage the sequential information of slot labels. During training the IC and SL objectives are jointly optimized.

3.3 Metrics

We report results with two common metrics used in production SLU: intent classification error rate (ICER) and semantic error rate (SEMER). Both metrics are Recall-based, as they are computing the error rate with respect to the ground-truth domain (annotated manually by language experts). ICER is the ratio of incorrect intents to the total number of utterances (and we will mainly rely on this evaluation metric further for intent classification):

$$\text{ICER} = \frac{(\#\text{ incorrect intents})}{(\#\text{ total utterances})}. \quad (1)$$

SEMER considers both intent classification and slot classification together. SEMER allows us to measure the effect of improved intent classification on the overall joint model performance. It is computed based on the number of insertions, deletions and substitutions for slots and the intent in a recognised utterance compared to a reference utterance:

$$\text{SEMER} = \frac{(\#\text{ slot errors} + \text{intent errors})}{(\#\text{ reference slots} + \text{intents})} \quad (2)$$

4 Results

First, we study the impact of an unmodified LGL method on model training (4.1, 4.2), splitting our training data into several batches and masking parts of the data as described in Section 2. The batches are split per intent, with each batch containing several classes. We train the model in several iterations, gradually unmasking the data. Second, we adapt LGL to SLU production scenario and conduct experiments where the data for new classes is not available at the early model training iterations (4.3). In all experiments, we compare the result against a baseline, which is the same model trained on all classes in a single iteration.

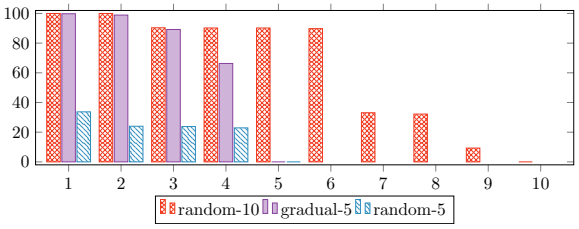

Figure 2: Masked data distribution per iteration (% of the full dataset) for random and gradual orderings in LGL for IC experiment. The last iteration (the 5th or the 10th) contains always only unmasked data.

	ordering, # of batches		
Metric	(r, 5)	(g, 5)	(r, 10)
SEMER	-2.61	-1.98	-2.50
ICER	-3.39	-2.34	-9.11

Table 1: Evaluation results for LGL applied to IC. The relative difference is with respect to baseline model that does not use any form of LGL or other curriculum learning.

4.1 LGL for intent classification

In the first experiment, we apply LGL to intent classification, i.e. only masking intent labels. Specifically, we replace all intent labels in masked batches by a placeholder (*OtherIntent*). We randomly group classes in the dataset into 5 and 10 batches for LGL training, so that each batch contains 5 to 6 intents (we include masked data statistics per batch in Figure 2). To account for the unbalanced class distribution in the dataset, we evaluate two strategies for selecting the order of batches for LGL:

- Random order (*r*). We select the order randomly, which results in 66% of the annotated data being included in the first iteration.
- Gradual order (*g*). We select batches based on the corresponding data size, starting with the smallest one. In that scenario, the first 4

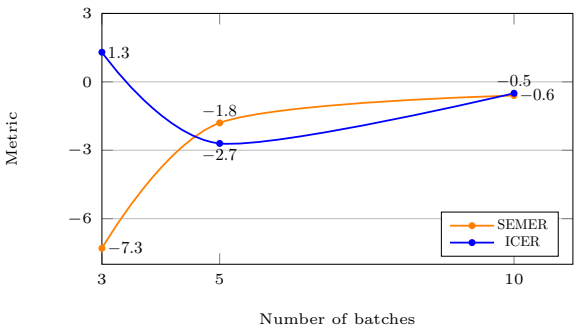

Figure 3: Evaluation results for LGL applied to NER and IC tasks (relative difference). The relative difference is with respect to the baseline model that does not use any form of LGL or other curriculum learning.

iterations include 33% of the annotated data, while the largest classes are added in the last iteration.

The results of the experiments on a real-life Music dataset are presented in Table 1. We can see that the experimental models trained using LGL outperform the baseline across all metrics, which confirms the results first obtianed in Cheng et al. (2019). For the ICER metric, our best model (LGL (r, 10)) improves the error rate by -9.11% compared to the baseline. We conclude that the increasing number of batches has a positive impact on the LGL performance here, as the model trained on 10 batches outperforms the same model trained on 5 batches by 5.72%. The improvement in SEMER is smaller, which is expected as we apply LGL to intent classification only (i.e., the dataset was split into batches based on intents only, and all slots were left unmasked).

The selection strategy (r vs. g) has a substantial impact on the model performance. The model trained on a random selection of the training classes performs better on both SEMER and ICER metrics than the same model trained on the sets of classes selected gradually (cf. (r, 5) vs. (g, 5) in Table 1). Therefore, we only use the random ordering in the other experiments.

We also experiment with LGL approach without resetting the learning rate. In the current model, we used learning rate scheduler to control the learning rate using the specified steps, where the first step reset the learning rate to 0 for each model iteration. However, since the encoder is initialised from the encoder of the previous model starting from the second iteration, we experimented with keeping the learning rate multiplier constant (0.1) for that and

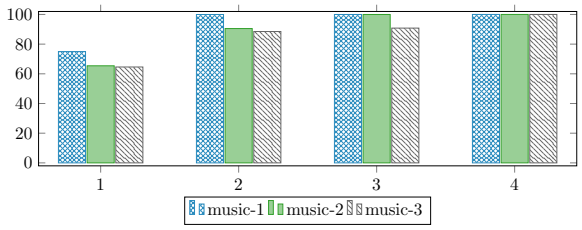

Figure 4: Data distribution per run and iteration (% of the full dataset; the last iteration always contains 100% of the data). Each run represents a different feature grouping and order.

all subsequent iterations. The results with respect to the (LGL (r, 5)) approach show only marginal improvement (avg. -0.27 rel. improvement over ICER and SEMER), therefore we conclude that resetting the learning rate does not have significant impact on the LGL training.

4.2 LGL for NER and IC

In this experiment, we apply masking to both intent and slot labels, splitting the training dataset into 3, 5 and 10 batches to further study the impact of the batch size[4]. In addition to intent masking (described in 4.1), slot masking is done as follows: *madonna|ArtistName -> madonna|OtherSlot*.

The results for the Music domain are visualized in Figure 3. As one can see, the best result is achieved when selecting a middle number of batches (between 3 and 5), while a very large number of batches (10) potentially overfits the model. This result differs from the LGL result on IC only (where using 10 batches is superior to using 5 on ICER metric) potentially due to a much larger number of slots that are left masked.

4.3 Gradually adding new features

Having applied LGL to the task of IC and NER, we showed that it is able to improve IC performance by -9.1% relative ICER and -2.5% relative SEMER. However, the downside of LGL for production SLU setup is the increased number of training iterations, which is associated with additional computational cost. In addition, in a real-life scenario, the data for new classes only becomes available with time. Therefore, in the following experiments, we modify the original LGL setup and conduct experiments where the data for new classes is added gradually within several iterations. Note that we select this setup to account for a production scenario when a

[4]We do not include data statistics here for space reasons.

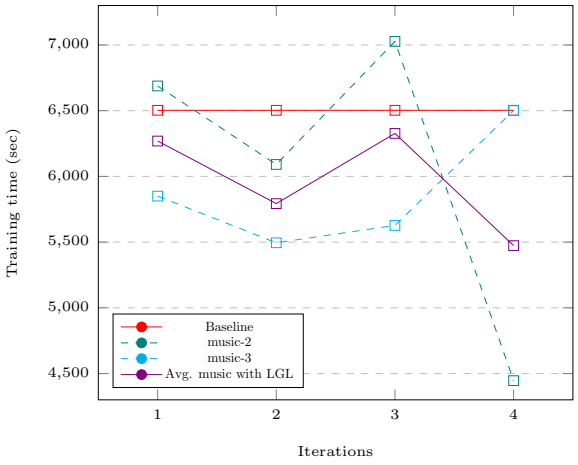

Figure 5: Average training time per LGL iteration compared to the baseline (note that the baseline here is the full model trained once on all available data in each iteration, which represents the upper bound).

model is trained in several iterations within a fixed release cycle (e.g., a period of several weeks); for simplicity, we assume that after each iteration the model is fully retrained on all data to avoid any model drift-related effects (which are out of scope of this work). This would also correspond to a 100% data replay strategy in continuous learning approaches (Payan et al., 2021).

We split the Music domain dataset into 4 batches corresponding to model releases, each one containing a new set of features (another reason for that is best result observed using 3 to 5 batches, cf. 4.2). With each iteration, a new batch of data (comprising several new classes and representing a new feature) is added to the model. We experiment with different feature order when grouping the data into batches (for instance, the first run may contain features represented by *PlaySong* and *PlayAlbum* intents grouped together for the first iteration, *PlayRadio* for the second iteration, while the second run could have *PlayAlbum* as the first iteration, and *PlaySong* and *PlayAlbum* for the second, etc.). We do not apply any masking in this scenario and at every step only the data for the currently supported features is used to train the model. The data distribution per iteration and run is presented in Figure 4.

The results after the final iteration are presented in Table 2 relative to a baseline that was once trained on full data. The experimental models trained using modified LGL setup outperform the baseline across SEMER and ICER in 2 out of 3 cases. In the last case, LGL outperforms the baseline on SEMER (-7.3%), while ICER slightly increases (+2.9%). This could be explained by the different number of classes added to the model – in the last iteration, we add 9.2% of training data, while for other orderings, a much smaller amount is added in the last step.

Run #	ICER	SEMER
music-1	-2.2	-2.3
music-2	-3.2	-2.7
music-3	2.9	-7.3

Table 2: Evaluation results for modified LGL method per run (each run represents a different feature grouping and order). The relative difference is with respect to a baseline model that does not use any form of LGL or other curriculum learning.

Another benefit of the modified LGL method is that it helps reduce training time when new features are added on top. In Fig. 5, we compare the training time for two runs and their average to the baseline model (we use the model trained once on all available data as upper bound; its training time is the same for each iteration). We see that the average training time for each of the iterations is less than the training time of the full model, because we use less training data in the first iterations, and initialise the model from the previous one in subsequent iterations. For individual iterations, we observe up to 25% training time reduction. Overall, we conclude that gradually adding features with warm-starting is beneficial for production SLU, as it helps improve model accuracy and reduces the overall training time spent per release cycle.

5 Conclusion

We applied LGL to the tasks of intent classification and slot filling in the context of SLU and studied the impact of LGL on intent classification error rate and semantic error rate. We conducted the experiments using different class selection strategies and showed that LGL improves intent classification performance for SLU by -9.1% relative ICER, without requiring any new training data or modified model architecture. In addition, we adapted original LGL setup to SLU production scenario when new features are gradually added within fixed release cycle, and showed that it is able to improve model accuracy by up to -7.3% relative SEMER while reducing average training time by up to 25% for individual iterations.

As future work, we would like to further explore

LGL application to feature expansion problem, apply it to other domains and investigate the impact of batch size on the model performance. In addition, we would track the impact of LGL training on model's generalization performance and computational cost over time.

References

Jordan Ash and Ryan P Adams. 2020. On warm-starting neural network training. *Advances in Neural Information Processing Systems*, 33.

Magdalena Biesialska, Katarzyna Biesialska, and Marta R. Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6523–6541, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Giuseppe Castellucci, Simone Filice, Danilo Croce, and Roberto Basili. 2021. Learning to solve NLP tasks in an incremental number of languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 837–847, Online. Association for Computational Linguistics.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. BERT for Joint Intent Classification and Slot Filling. *arXiv preprint arXiv:1902.10909*.

Hao Cheng, Dongze Lian, Bowen Deng, Shenghua Gao, Tao Tan, and Yanlin Geng. 2019. Local to global learning: Gradually adding classes for training deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4748–4756.

Eunah Cho, He Xie, and William M Campbell. 2019. Paraphrase generation for semi-supervised learning in nlu. In *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*, pages 45–54.

Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil D Lawrence. 2018. Continual learning in practice. In *Proceedings of the NeurIPS 2018 workshop on Continual Learning*.

Judith Gaspers, Quynh Do, and Fabian Triefenbach. 2020. Data balancing for boosting performance of low-frequency classes in spoken language understanding. In *Interspeech 2020*.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Kasidis Kanwatchara, Thanapapas Horsuwan, Piyawat Lertvittayakumjorn, Boonserm Kijsirikul, and Peerapon Vateekul. 2021. Rational LAMOL: A rationale-based lifelong learning framework. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2942–2953, Online. Association for Computational Linguistics.

M. Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Samuel Louvan and Bernardo Magnini. 2020. Recent neural methods on slot filling and intent classification for task-oriented dialogue systems: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 480–496, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of Recurrent-Neural-Network Architectures and Learning Methods for Spoken Language Understanding. In *Interspeech*, pages 3771–3775, Lyon, France.

Justin Payan, Yuval Merhav, He Xie, Satyapriya Krishna, Anil Ramakrishna, Mukund Sridhar, and Rahul Gupta. 2021. Towards realistic single-task continuous learning research for NER. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3773–3783, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R. Bowman. 2020. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5231–5247, Online. Association for Computational Linguistics.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota. Association for Computational Linguistics.

Alex Sokolov and Denis Filimonov. 2020. Neural machine translation for paraphrase generation. *arXiv preprint arXiv:2006.14223*.

Yi Tay, Shuohang Wang, Anh Tuan Luu, Jie Fu, Minh C. Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. 2019. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4922–4931, Florence, Italy. Association for Computational Linguistics.

H. Weld, X. Huang, S. Long, J. Poon, and S. C. Han. 2021. A survey of joint intent detection and slot-filling models in natural language understanding.

Xiaodong Zhang and Houfeng Wang. 2016. A Joint Model of Intent Determination and Slot Filling for Spoken Language Understanding. In *Proceedings of the Twenty-Fifth IJCAI*, page 2993–2999, New York, NY, USA.

A Local To Global Learning

The main idea of Local to Global Learning (LGL) algorithm used in this work is to gradually train the neural network starting with a few output classes and subsequently extending to more classes. In the following, we provide a more detiled oevrview of the method following Cheng et al. (2019).

The model training is performed in several iterations. The model for each iteration is initialized from the previous one. During each iteration, the entire training set is used, however, the classes that are not learned during that specific iteration are masked. Thus, the model is learned on a fraction of classes from the complete output space of the training set, while the whole dataset is still exposed. As compared to traditional model learning, the loss function is not minimized across all classes simultaneously, but is minimized iteratively, each time learning a new set of classes in addition to the already known classes. At each step, a set of new classes is added to the training setup by unmasking them in the dataset and the model is trained until convergence. Mathematically, it can be expressed as (we refer to (Cheng et al., 2019) for details):

$$
\begin{aligned}
w_k^* &= \arg\min_w L(w, X_{S_k}, Y_{S_k}; w^*{}_{k-1}) \\
s.t. i^* &= f(w, X_{S_{k-1}^C}, Y_{S_{k-1}^C}; w_{k-1}^*), \\
S_k &= S_{k-1} \cup \{i^*\},
\end{aligned}
$$

where L is the loss function and w^* are the model weight produced by minimizing L. The dataset contains pairs of samples and class annotations $G = \{X, Y\}$, where $K = \{1, 2, ..., K\}$ is a set of available output class labels. The classes are grouped into N batches of equal size and after each training iteration, one batch $i*$ is added. S_k is the set of classes from K that is used in the k-th step. X_{S_k}, Y_{S_k} is the data, which labels are in S_k and S_{k-1}^C is the set of classes not in S_{k-1}. The selection strategy is represented by the function f, which defines how a new batch of classes is selected from the untrained classes.

The set of classes at the current iterations S_k is unmasked in the dataset during the training, while the yet unavailable classes S_{k-1}^C are masked with a placeholder label, but the corresponding data instances $X_{S_{k-1}^C}$ are kept in the training data. Hence, the full data set G is used for training at every iteration. After the model is learned on the first batch, its encoder is used to initialize the encoder

for the next training step. Thereby, the final model is learned iteratively through several training runs with an increasing number of output classes. The encoder part of the model is carried further with every iteration and the output layers are re-initialized each time to account for changing output space.