Update Frequently, Update Fast: Retraining Semantic Parsing Systems in a Fraction of Time

Anonymous ACL submission

Abstract

Currently used semantic parsing systems deployed in voice assistants can require weeks to train. Datasets for these models often receive small and frequent updates, data patches. Each patch requires training a new model. To reduce training time, one can fine-tune the previously trained model on each patch, but naïve fine-tuning exhibits catastrophic forgetting – degradation of the model performance on the data not represented in the data patch.

In this work, we propose a simple method that alleviates catastrophic forgetting and show that it is possible to match the performance of a model trained from scratch in less than 10% of a time via fine-tuning. The key to achieving this is supersampling and EWC regularization. We demonstrate the effectiveness of our method on multiple splits of the Facebook TOP and SNIPS datasets.

1 Introduction

Semantic parsing is the task of mapping a natural language query into a formal language. Semantic parsing models are at the core of commercial dialogue systems like Google Assistant, Alexa, and Siri. Typically, for a given query, the model should identify the requested action (intent) and the associated values specifying parameters of the action (slots). For example, if the query is *Call Mary*, the action could be *callAction* and *Mary* could be a value of the slot *contact*.

Real-world datasets for semantic parsing get frequent updates, especially for the long tail of low-resource classes 1, and after each update, a new model needs to be trained. Considering that such datasets can contain millions of examples (Damonte, Goel, and Chung, 2019), currently used neural semantic parsers can take days or even weeks of training. To avoid training a model from scratch every time, we can fine-tune the current model on the new examples. However, this leads to catastrophic forgetting (French, 1999) and to a performance drop on the original dataset.

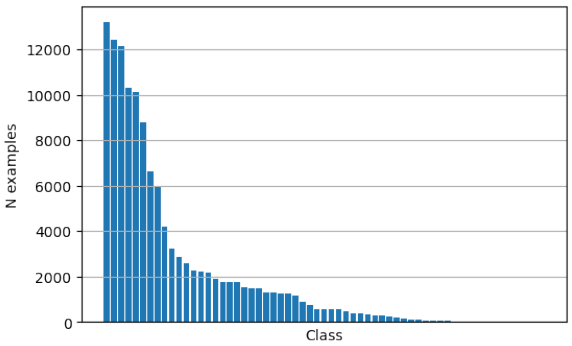

Figure 1: Class distribution in TOP dataset. The most frequent class (on the left) is SL:DATE_TIME with 13214 examples and the least frequent are IN:NEGATION and SL:GROUP (or the right) with just one example available.

Methods that mitigate catastrophic forgetting have been developed (Kirkpatrick et al., 2017; Sun, Ho, and yi Lee, 2020). However, there has been limited work studying their applicability to complex tasks such as semantic parsing (Lee, 2017) and to modern neural network models (de Masson d'Autume et al., 2019). We also argue that a more practical data-incremental (as opposed to task-incremental) scenario has not received enough attention.

In this paper, we investigate the applicability of such methods to semantic parsing and demonstrate that using a simple combination of data supersampling and EWC (Kirkpatrick et al., 2017) allows us to (1) achieve approximately the same test-set performance as training a new model (2) reduce catastrophic forgetting effect to almost negligible and (3) to do so in less than 10% of the time needed to train a model from scratch. We also propose a new semantic parsing metric that is well-suited for per-class evaluation.

We evaluate our method in multiple settings,

varying the amount of data available for each category initially and in subsequent increments. We show that our method achieves higher accuracy than training from scratch in some of the more realistic settings, e.g., when less than 30 examples are available for a class in the original training data and less than 300 examples are added incrementally.

2 Incremental training

We formalize our task as follows. Suppose we have a network $\mathbb{M}_{prev}$ trained on the dataset D_1 as our current model. We are then given a dataset D_2 as additional "data patch" data. Let T_1 be the time to train $\mathbb{M}_{from\_scratch}$ from scratch on $D_1 \cup D_2$ and T_2 is the time to fine-tune the previous model. Our goal is to fine-tune $\mathbb{M}_{prev}$ and produce $\mathbb{M}_{from\_scratch}$, resulting in comparable performance with $\mathbb{M}_{finetuned}$ with $T_2 \ll T_1$. In this work, we study the case where D_2 has a data distribution that is skewed and significantly different from the distribution in D_1. Note that we do not explicitly study scenarios where conflicting training data is being corrected using D_2 as in (Gaddy et al., 2020).

Peculiarities of semantic parsing: While standard classification tasks assume a single label for each training example, the output for semantic parsing is a tree with multiple nodes (Fig. 2). Tree structure can automatically account for the class correlations as the correlated classes are likely to occur in a single training example as different tree nodes. Another potential benefit of having structured outputs is that the difference between the class distribution in D_1 and D_2 likely to be smaller than in unconstrained classification, since D_2 is likely to include some frequent classes from D_1 in its parsing trees.

Another characteristic of commercial semantic parsing is high number of classes. For example, the publicly available TOP dataset (Gupta et al., 2018) uses 25 intents and 36 slots for just two domains *navigation* and *events*. Real production systems can contain hundreds of classes for a single domain (Rongali et al., 2020). Thus, incremental training is particularly interesting as the long tail of the class distribution calls for frequent model updates.

3 Fine-tuning approaches

In continual learning, it is generally assumed that the previous data is not accessible at the next learning iteration (de Masson d'Autume et al., 2019). However, this is usually not the case for commercial semantic parsing systems. Hence, we assume that we have access to all data.

Our main reason to apply continual learning methods to semantic parsing is to reduce training times. Thus, we only use the methods that do not require significant additional computation. We investigate several fine-tuning approaches using data-reuse and regularization.

3.1 Reusing old data

We consider two main ways to reuse the old data (D_1) during fine-tuning: **replay** and **sampling**. Experience replay (Lin, 1992; de Masson d'Autume et al., 2019) of size N saves N training examples into the replay buffer, allowing reuse during the next continual learning iteration. Usually, N is small, but we experiment with various N ranging from 0 to the size of the dataset. To better parametrize experience replay for our setup, instead of $N \in \mathbb{N}$ we select $p \in [0, 1]$ – where p is the fraction of D_1 dataset that is used during fine-tuning.

Note that experience replay only uses a part of the dataset. This may be beneficial in some setups, but we expect that using the whole dataset can reduce forgetting even further. Thus, our second method samples batches from both "old" and "new" subsets. We weigh examples from the "old" and "new" data differently and always weight "new" data more. To simplify comparison with experience replay, we also parametrize this method with p, where p is an expected proportion of old dataset sampled by the end of the epoch.

3.2 Regularization-based methods

Another approach to mitigate forgetting is to regularize the model in a way so that the model's predictions on the old data are changed minimally.

Move Norm Regularization We introduce a simple regularization method similar to weight decay but one that explicitly targets incremental training / continual learning. Move norm is a regularizer that prevents the model with weights Θ from diverging from the previous model weights Θ^{prev} (Equation 1). It is added to the loss function analogous to weight decay and is parametrized by λ.

$$R(\Theta, \Theta^{prev}) = \lambda||\Theta - \Theta^{prev}||_2 \qquad (1)$$

Elastic Weight Consolidation EWC was introduced in Kirkpatrick et al. (2017) and can be de-

scribed as a parameter-weighted move norm:

$$R(\Theta, \Theta^{prev}) = \lambda||F \circ (\Theta - \Theta^{prev}))||_2 \quad (2)$$

where $\circ$ is element-wise multiplication, and F is a vector of positive parameter importance values. Theoretically, F is the inverse of the diagonal of the Fisher matrix, but in practice, the squared gradients of the model trained on D_1 are used as an approximation.

The issues of applying EWC to modern neural networks are known in the literature (Lee, 2017) and come from the very flat loss surface at the end of the training. To mitigate this, we use the average values of the squared gradients over training steps, starting at the very first step. This method is similar to (Zenke, Poole, and Ganguli, 2017) and we found it to significantly improve the exponential smoothing of the gradient norm proposed by Lee (2017).

3.3 Sampling EWC

Even though the original EWC formulation specifically targeted the case with no old data available, we propose to combine EWC with sampling/replay. We expect to see additional improvements in terms of test-set performance improvement and reduced amount of forgetting.

4 Experimental setup

The model used in the study is a sequence-to-sequence Transformer model with a pointer network and BERT encoder, as in Rongali et al. (2020). We use the same hyperparameters[1]. For fine-tuning with EWC we use regularization strength of 100 (regular experiments) and 1000 ("high EWC" experiments).

To mimic an incremental learning setup we split the training set into two segments: "old data" (D_1) and "new data" (D_2) . The splitting procedure aims to mimic the real-world iterative setup when a trained model already exists and we want to incorporate new data into this model.

4.1 Data splits

We experiment with multiple data splits, summarized in Table 1. For each split, we chose a split class C and a split percentage P_C. We then randomly select P_C percentage of the training examples with class C for the "new data" subset. We selected mid- and low-frequency classes in our experiments. In such cases, a 95%+ split leaves a handful of examples in the D_1 subset and moves the rest to D_2.

To ensure that the "new data" does not contain classes absent from "old data", we add a small set of examples that contain all the classes to the "old data".

[1]Details in the Appendix A

Split	Dataset	# old	# new
Name Event 95	TOP	60	1.1k
Path 99	TOP	15	1.5k
Organizer Event 95	TOP	15	301
Get Loc. School 95	TOP	9	171
Get Weather 95	SNIPS	95	1.8k
Served Dish 90	SNIPS	25	230

Table 1: Data splits used in the experiments, showing the number of examples of the target class contained in the "old" and the "new" segments of the data. The first column shows the class name, followed by the proportion of training examples moved to the the D_2 subset.

4.2 Datasets

We use two popular task-oriented semantic parsing datasets: Facebook TOP (Gupta et al., 2018) and SNIPS (Coucke et al., 2018). SNIPS consists of flat queries containing a single intent and simple slots (no slot nesting), while the TOP format allows tree structures. More specifically, each node of the parse tree is an intent name (e.g., IN:GET_WEATHER), a slot name (SL:DATE), or a piece of input query text. The tree structure is represented as a string of labeled brackets with text, and the model predicts this structure using the input query in a natural language (see Fig. 2).

The TOP dataset consists of 45K hierarchical queries, about 35% of which have tree depth > 2. SNIPS dataset has a simpler structure (only a single intent for each query) and the train part consists of 15K examples. To unify the datasets, we reformat SNIPS to fit the TOP structure.

4.3 Metrics

We propose a new metric for semantic parsing, Tree-Path F_1 score. This metric that evaluates the accuracy of "subcommands" in the parsing tree – paths from the root to a leaf. It is more fine-grained than EM and allows to compute class-specific scores.

We use exact match (EM) to evaluate model performance across all classes and Tree Path F_1

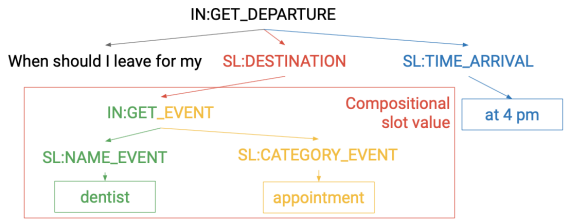

Figure 2: Semantic parsing example. Each tree path starts at the IN:GET_DEPARTURE node and finishes with a slot value (the values are in the boxes). Tree paths are colored.

score (TP-F_1) to evaluate performance on a specific class. To estimate the uncertainty of our metrics, we divide our test set into 5 folds and compute the mean and standard deviation of model performance across the folds.

Tree Path Score To compute Tree Path F_1 Score (TP-F_1), the parse tree is converted into tree paths from the root to node that is either a valid slot or an intent without slots. Tokens that do not correspond to any slot value are ignored. This procedure is performed for both correct and predicted trees and then the F_1 score is computed on the paths.

With precision $P = \frac{\#\text{correctly predicted paths}}{\#\text{predicted paths}}$ and recall $R = \frac{\#\text{correctly predicted paths}}{\#\text{expected paths}}$.

For example, for the query, "When should I leave for my dentist appointment at 4 pm", the parse tree looks as in Figure 2 and has four tree paths. Every path starts at the root (IN:GET DEPARTURE) and goes down to the slot value. For the SL:DESTINATION slot, the value is compositional and equal to the string [IN:GET_EVENT [SL:NAME_EVENT: DENTIST] [SL:CATEGORY_EVENT: APPOINTMENT]] .

For example, if the predicted tree has a different value for the slot SL:NAME_EVENT, two paths in it would differ from the correct ones. A path to the value of the SL:NAME_EVENT slot and a path to the value of the SL:DESTINATION slot – because SL:DESTINATION value is compositional and contains NAME_EVENT as a part of it. In this case, the number of correctly predicted paths would be two (SL:TIME_ARRIVAL and SL:CATEGORY_EVENT slots), the number of predicted paths would be four, and the number of expected paths also four. TP-F_1 in this case equals 1/2.

To compute a per-class score only the paths containing the target class are considered. In the example above, NAME_EVENT's TP-F_1 is zero, as there are no correctly predicted paths for it.

5 Results

5.1 Performance

Achieving approximately the same as a trained from scratch model performance is an important requirement for the applicability of our method in practice. To assess whether the proposed fine-tuning method performs on par with a model trained from scratch on a combined "old"+"new" dataset, we compare the following metrics: TP-F_1 for the target class and exact match. The former evaluates the gain of using the additional data, the latter evaluates the model across all classes.

The Table 2 shows that our methods that reuse old data are able to achieve the performance of a fully retrained network. At the same time, naïve fine-tuning sometimes performs *worse than the current (not fine-tuned) model even on the target class*. A popular continual learning approach, EWC, significantly improves upon naïve fine-tuning, but lags behind full retraining in performance.

The Figure 3 shows five different fine-tuning setups for the ORGANIZER_EVENT 95 split. In each, we vary the amount of old data the method uses (parameter p, Section 3.1). We can see that naïve fine-tuning (the leftmost points for "replay" and "sample") fails to match the performance of full retraining even on the target class which is well represented in the fine-tuning data. Same holds true for EWC without data sampling (the leftmost points for "EWC+replay", "EWC+sample", "high EWC+replay"). Additional experimental results are provided in the Appendix B.

We also plot an auxiliary x-axis (top) that shows the number of training steps used for fine-tuning. To simplify the comparison with the model trained from scratch, we convert them to relative training steps. For the TOP dataset, 100% is 28,000 steps that are needed to train a new model.

5.2 Speedup

The main advantage that justifies using continual learning instead of training a new model from scratch, is the speedup. In our experiments, we have observed a large and consistent decrease in training steps required to match the performance of a model trained from scratch.

The best improvement was a 30x speedup (in the number of training iterations) for the GET LOCATION SCHOOL 95 and ORGANIZER_EVENT 95

Split	from scratch		no finetunig		naïve		EWC		ours	
	TP-F1	EM	TP-F1	EM	TP-F1	EM	TP-F1	EM	TP-F1	EM
Name Event 95	**0.84**	**0.82**	0.73	0.80	0.50	0.76	0.62	0.79	0.81	**0.82**
Path 99	**0.64**	**0.82**	0.19	0.79	0.30	0.72	0.49	0.78	0.63	0.81
Organizer Event 95	0.60	**0.82**	0.35	0.81	0.23	0.78	0.52	0.80	**0.74**	**0.82**
Get Loc. School 95 [2]	0.65	**0.82**	0.15	0.82	0.66	0.81	0.62	0.80	**0.79**	**0.82**
Get Weather 95	**0.97**	**0.95**	0.93	0.92	0.59	0.67	0.65	0.83	**0.97**	0.94
Served Dish 90 [2]	0.90	**0.95**	0.81	0.92	0.59	0.90	0.84	0.91	**0.94**	0.94

Table 2: Target class TP-F1 and exact match for the baselines and our methods. The baselines include training from scratch, fine-tuning the model on new data only without any regularization (naïve) and with EWC. For "ours" we select the best combination of method (Sec. 3) and old data amount. In all splits we are able to achieve comparable to a from scratch performance while naïve and EWC fail to do so in all splits except GET_LOC_SCHOOL 95.

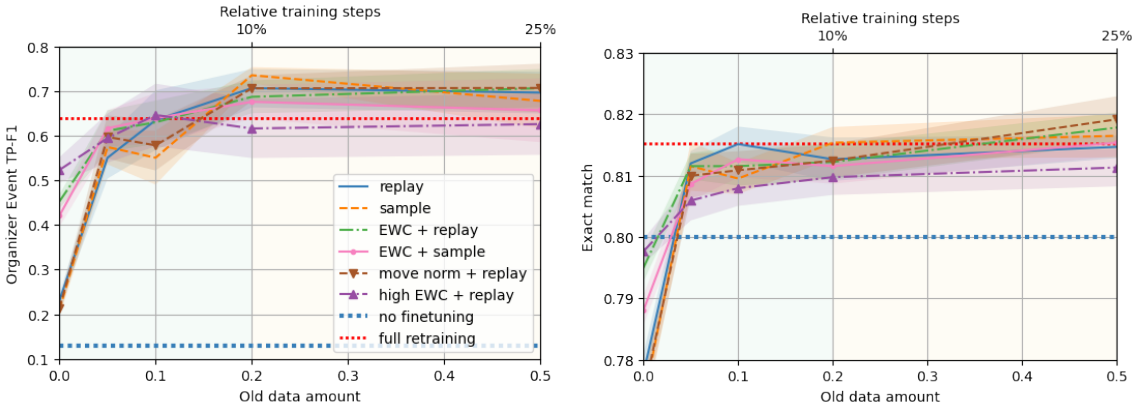

Figure 3: We plot class-specific TP-F_1 and exact match metrics against the fraction of original data sampled/replayed for the ORGANIZER_EVENT 95 split. Both sampling and replay from the old data help significantly. Similar performance to a full retraining can be achieved with sampling only around 10% of the original data (using either of the methods). This comes at a fraction of computational cost of a training a new model.

Split	Replay	Sample	MoveNorm+Replay	EWC+Replay	EWC+Sample
Name Event 95	**26%**	**26%**	**26%**	**26%**	**26%**
Path 99	40%	**29%**	40%	N/A	N/A
Organizer Event 95	6%	10%	10%	**3%**	**3%**
Get Loc. School 95 [2]	**3%**	**3%**	25%	5%	5%
Get Weather 95	**10%**	**10%**	**10%**	28%	**10%**
Served Dish 95 [2]	**16%**	**16%**	**16%**	**16%**	**16%**

Table 3: Relative number of training steps needed to reach the full retraining performance (within two standard deviations). 100% training steps means full retraining. N/A means the performance has not been reached by either variation of the method. Different methods perform similarly.

splits (Table 3). It is worth noting that both of these splits model a more realistic scenario with a moderately-sized data patch (hundreds of examples) compared to the PATH 99 split (1.5 thousand examples). Time-wise, in our setup we observed a twenty-fold reduction in training time and we believe this may be pushed even further using a separate device (GPU/TPU) for evaluation.

5.3 Amount of forgetting

Looking at Table 3, one can notice that there is no clear winner in this speed battle. All methods perform quite similar and it may be hard to decide which one to use. However, another question worth asking is, "how much does the model degrade after the fine-tuning?". While low exact match usually indicates catastrophic forgetting, high EM values may be misleading. A significant improvement over several classes can increase EM just enough

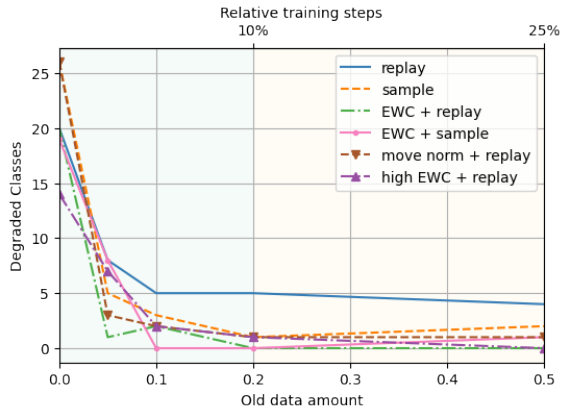

Figure 4: Number of classes with 2σ-degraded tree path scores.

to hide some degraded classes.

It is questionable to deploy a model that is better than the previous one overall, but performs significantly worse for some specific scenarios. To quantify this, we evaluate the number of classes that degraded significantly. We define "significant" as more than 2σ where σ is a standard deviation evaluated using bootstrapping (Section 4.3).

Figure 4 shows that the methods that use EWC and sampling exhibit less forgetting. For the case of ORGANIZER_EVENT 95 split EWC completely eliminates forgetting starting at 10% of old data.

In summary, while different methods may yield similar results when looking at EM and class-specific TP-F_1 (Fig. 3), they *vary significantly in terms of the effects of catastrophic forgetting* (Fig. 4). In all of our experiments we saw decreased amounts of forgetting when using EWC and old data sampling. In 5/6 of our splits the method with the lowest forgetting was EWC+sampling.

5.4 Negative Results: Layer Freezing

One of the approaches we initially investigated was inspired by Howard and Ruder (2018). Hypothetically, layer freezing during the fine-tuning stage could restrain the model from diverging too far and mitigate the forgetting. We experimented with freezing encoder/decoder/logits+pointer network and their combinations. In all experiments freezing performed worse than regular fine-tuning. Freezing encoder and decoder and only updating the logits and pointer networks turned out to be a particularly bad method which suggests that improving low-resource classes that we targeted benefits mainly from constructing new features and not just reusing the learned ones.

[2]We observe high variance in metrics for the "Served Dish 95" and "Get Loc. School 05" splits. It could have been caused by a low number of test examples with the target class.

6 Related Work

Practical applications of Continual Learning (McCloskey and Cohen, 1989; Parisi et al., 2019) to the NLP field has been rather limited (Li et al., 2019; Lee, 2017). A usual task-incremental setup (Caccia et al., 2020; de Masson d'Autume et al., 2019) is far from the real-world applications. However, a simpler data-incremental setup (Cossu, Carta, and Bacciu, 2020; Mi et al., 2020) becomes extremely useful considering quickly growing size and computational requirements to train currently used NLP networks (Devlin et al., 2019; Radford et al., 2019; Raffel et al., 2020).

The main issue of continual learning is catastrophic forgetting (French, 1999; McCloskey and Cohen, 1989). Data-based (de Masson d'Autume et al., 2019; Sun, Ho, and yi Lee, 2020) and regularization-based (Kirkpatrick et al., 2017; Zenke, Poole, and Ganguli, 2017) methods has shown promising results in continual learning and applied to NLP tasks, including goal-oriented dialogue. (Lee, 2017) applies continual learning methods for a goal-oriented dialogue task. Nonetheless, their setup is significantly different from ours as they do not thoroughly study the reduction of computational requirements using continual learning. Another difference is the model scale which is an important parameter in modern deep learning and NLP (Kaplan et al., 2020). We argue that the effects of speedup in continual learning are the best motivation to study and apply these methods.

7 Conclusion

In this work, we consider a practical side of continual learning that has not received enough attention from the NLP researchers – the ability to quickly update an existing model with new data. Nowadays, training time becomes a more challenging issue every year. We anticipate that in the near future of billion-parameter-sized models, incremental and continual learning settings can not only lead to a significant advantage in terms of resource efficiency but also become a necessity.

We demonstrate that in the data-incremental scenario, fine-tuning is a viable alternative to training a new model. First, we show that simple data sam-

pling techniques such as experience replay or supersampling allow to match the target-class and overall performance of a model trained from scratch. Then, we demonstrate up to 30x speedup over training a new model. Finally, we explicitly measure catastrophic forgetting and show that EWC and sampling from the whole dataset (opposed to experience replay) significantly reduce the forgetting effect.

We suggest using EWC with supersampling and monitoring specialized metrics like the number of degraded classes to track forgetting for the practical applications. While different methods and different amounts of forgetting yielded the best results for different splits, EWC + 20% sample generally performed well across all splits and registered low degradation.

To extend this work to a broader set of practical cases, the future work will be concentrated on the class-incremental setup. This will allow the additional data to have classes not represented in the original dataset.

References

Caccia, M.; Rodriguez, P.; Ostapenko, O.; Normandin, F.; Lin, M.; Page-Caccia, L.; Laradji, I. H.; Rish, I.; Lacoste, A.; Vázquez, D.; et al. 2020. Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems* 33.

Cossu, A.; Carta, A.; and Bacciu, D. 2020. Continual learning with gated incremental memories for sequential data processing. *2020 International Joint Conference on Neural Networks (IJCNN)* 1–8.

Coucke, A.; Saade, A.; Ball, A.; Bluche, T.; Caulier, A.; Leroy, D.; Doumouro, C.; Gisselbrecht, T.; Caltagirone, F.; Lavril, T.; Primet, M.; and Dureau, J. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces.

Damonte, M.; Goel, R.; and Chung, T. 2019. Practical semantic parsing for spoken language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, 16–23.

de Masson d'Autume, C.; Ruder, S.; Kong, L.; and Yogatama, D. 2019. Episodic memory in lifelong language learning. In *Advances in Neural Information Processing Systems*, 13143–13152.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

French, R. M. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3(4):128–135.

Gaddy, D.; Kouzemtchenko, A.; Reddy, P. K.; Kolhar, P.; and Shah, R. 2020. Overcoming conflicting data for model updates. *arXiv preprint arXiv:2010.12675*.

Gupta, S.; Shah, R.; Mohit, M.; Kumar, A.; and Lewis, M. 2018. Semantic parsing for task oriented dialog using hierarchical representations. *arXiv preprint arXiv:1810.07942*.

Howard, J., and Ruder, S. 2018. Universal language model fine-tuning for text classification. *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)* 1:328–339.

Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; and Amodei, D. 2020. Scaling laws for neural language models. *ArXiv* abs/2001.08361.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; Hassabis, D.; Clopath, C.; Kumaran, D.; and Hadsell, R. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America* 114(13):3521–3526.

Lee, S. 2017. Toward continual learning for conversational agents. *arXiv preprint arXiv:1712.09943*.

Li, Y.; Zhao, L.; Wang, J.; and Hestness, J. 2019. Compositional generalization for primitive substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 4293–4302. Hong Kong, China: Association for Computational Linguistics.

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8(3-4):293–321.

McCloskey, M., and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24. Elsevier. 109–165.

Mi, F.; Chen, L.; Zhao, M.; Huang, M.; and Faltings, B. 2020. Continual learning for natural language generation in task-oriented dialog systems. *arXiv preprint arXiv:2010.00910*.

Parisi, G. I.; Kemker, R.; Part, J. L.; Kanan, C.; and Wermter, S. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113:54–71.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21(140):1–67.

Rongali, S.; Soldaini, L.; Monti, E.; and Hamza, W. 2020. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. In *Proceedings of The Web Conference 2020*, 2962–2968.

Sun, F.-K.; Ho, C.-H.; and yi Lee, H. 2020. Lamol: Language modeling for lifelong language learning. In *ICLR*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. *Proceedings of machine learning research* 70:3987.

A Hyperparameters and training setup

Table 4 contains hyperparameters used for pretraining. We used the Noam schedule (Vaswani et al., 2017) for the learning rate. Note that it involves $\frac{1}{\sqrt{d_{decoder}}} = \frac{1}{\sqrt{256}}$ learning rate scaling.

For fine-tuning, the same parameters were used unless otherwise stated in the experiment description. We searched for the best EWC regularization from `[0.1, 10, 100, 100]` and for the best move norm in `[0.01, 0.05, 0.1]`. If we say "high EWC" on our plots, it means EWC regularization factor to be 1000. Model, optimizer, and learning rate scheduler states were restored from the checkpoint with the best EM.

encoder model	BERT-Base cased
decoder layers	4
decoder n heads	4
decoder model size	256
decoder ffn hidden size	1024
label smoothing	0.0
batch size	112
optimizer	ADAM
ADAM betas	$\beta_1 = 0.9, \beta_2 = 0.98$
ADAM ϵ	10^{-9}
decoder lr	0.2
encoder lr	0.02
warmup steps	1500
freezed encoder steps	500
dropout	0.2
early stopping	10
ranodm seed	1

Table 4: Training and model hyperparameters

B Additional Experimental Results

Additional plots In this section we present additional experimental data, analogous to the one presented in the paper.

These plots exhibit a similar to their counterparts from Section 5 behavior.

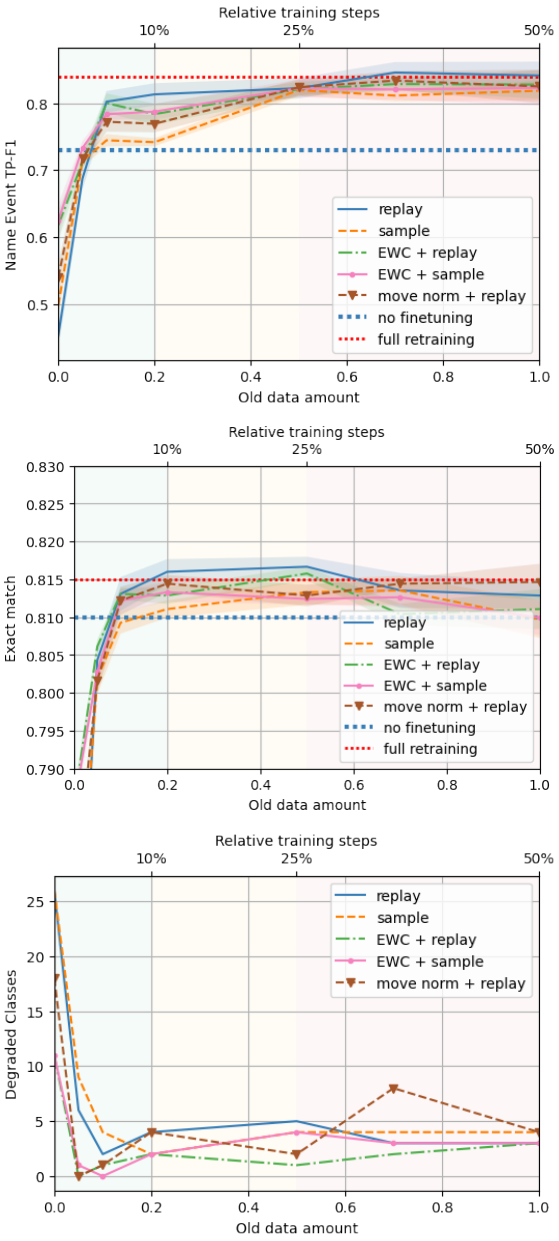

Figure 5: TOP NAME EVENT 95 results.

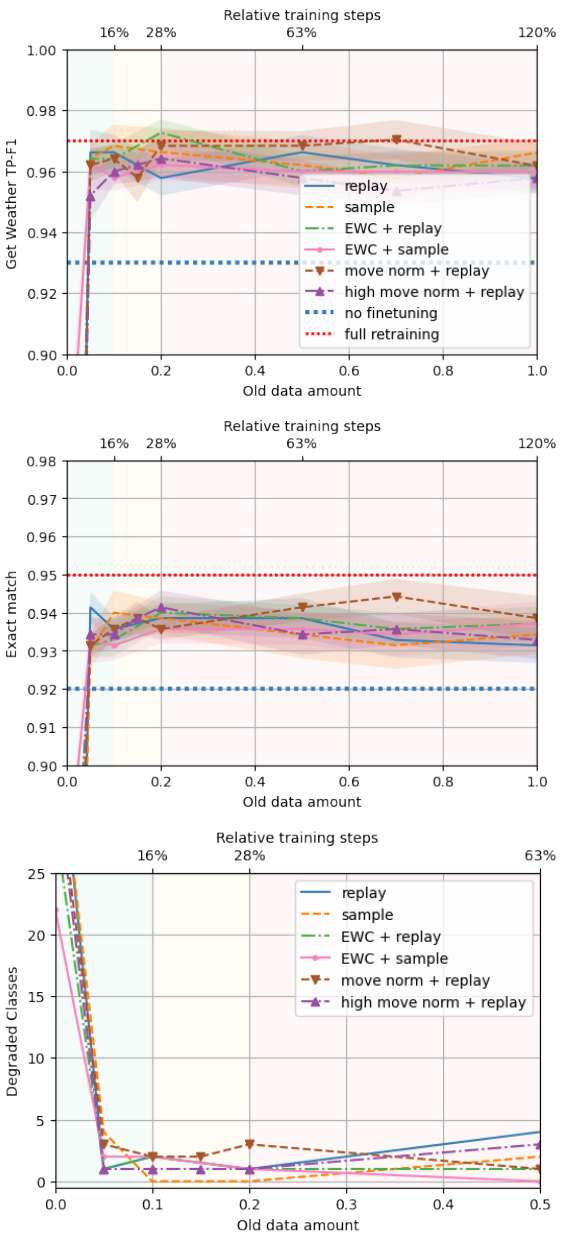

Figure 6: SNIPS GETWEATHER 95 results.

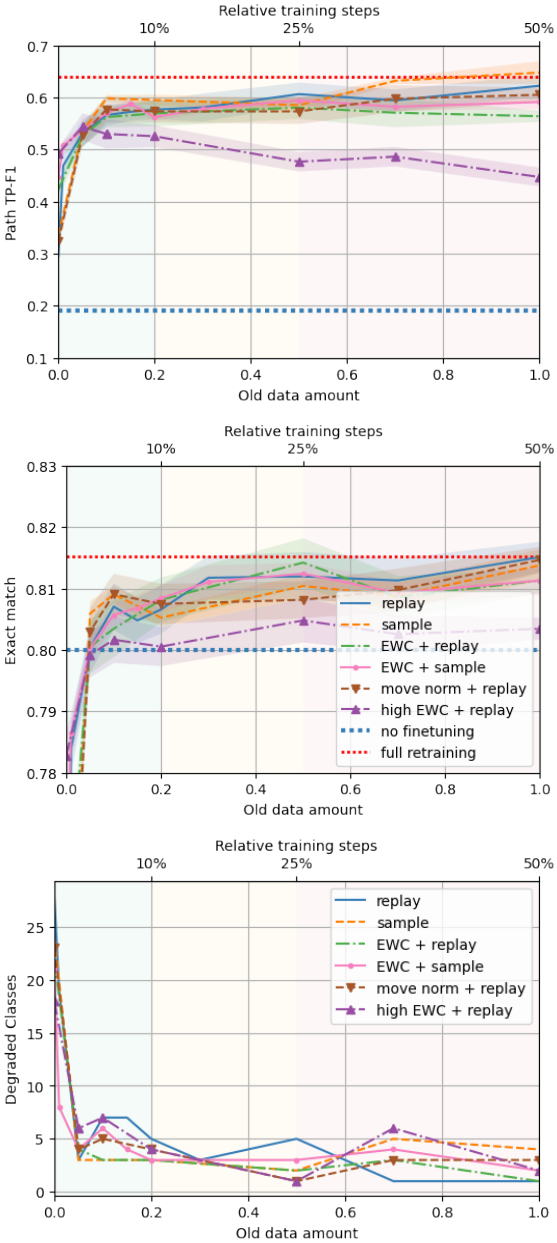

Figure 7: PATH 99 results.