ACCUMULATING CONVERSATIONAL SKILLS USING CONTINUAL LEARNING

Sungjin Lee

Microsoft Research, Redmond, WA, USA

ABSTRACT

While neural conversational models have led to promising advances in reducing hand-crafted features and errors induced by the traditional complex system architecture, training neural models from scratch requires an enormous amount of data. If pre-trained models can be reused when they have many things in common with a new task, we can significantly cut down the amount of required data. To achieve this goal, we adopt a neural continual learning algorithm to allow a conversational agent to accumulate skills across different tasks in a data-efficient way. We present preliminary results on conversational skill accumulation on multiple task-oriented domains.

Index Terms— conversational agents, continual learning, neural conversational model

1. INTRODUCTION

Conversational bots become increasingly popular in a wide range of business areas. In order to support the rapid development of bots, a number of bot building platforms have been launched, for example, Microsoft Bot Framework [1], Alexa Skills Kit [2] and so on. But the development of a business-critical bot still requires a serious effort from the design to actual implementation of several components such as language understanding, state tracking, action selection, and language generation. Not only does this complexity prevent casual developers from building quality bots but also introduces an unavoidable degradation in performance due to some non-trivial problems including unclear state representation design, insufficient labeled data and error propagation down the pipeline. Recently, neural approaches have shown the potential to solve such problems – the neural networks induce a latent representation in the course of the joint optimization of all components without requiring any labeling on internal state. Now neural networks are at the center of services like Conversation Learner [3] and Rasa [4] which allow developers to interactively build bots with much less hand-crafted features.

Despite such appealing aspects, neural approaches have own challenges to overcome: training a neural conversational model from scratch requires numerous dialog examples. The data-intensiveness problem intensifies when it comes to a composite task that consists of multiple subtasks. If one can repurpose already built models by composing them for a new task, we can significantly cut down the amount of required data. For example, one can add a payment handling capability to a new bot by repurposing any model that is already trained on payment-related conversations. However, it is not straightforward to compose neural networks due to lack of semantic modularity. Furthermore, one cannot simply train a model on a series of different tasks due to the *Catastrophic Forgetting* [1] problem. In order to address such difficulties, we propose to build on recent developments in the space of continual learning for neural models. Specifically, we adopt a variant of *Elastic Weight Consolidation* (EWC) algorithm to continuously learn a new task without forgetting valuable skills that are already learned. We present preliminary results on conversational skill accumulation on multiple task-oriented domains.

The rest of this paper is organized as follows. In Section 2 we present a brief summary of related work. In Section 3 we describe our approach for continual learning for conversational agents. In Section 4 we discuss our experiments. We finish with conclusions and future work in Section 5.

2. RELATED WORK

Broadly there are two lines of work addressing the data-intensiveness problem. The first makes use of domain-specific information and linguistic knowledge to abstract out the data [2, 3, 4, 5]. The second line of work adopts data recombination approaches to generate counterfeit data that mimics target domain dialogs [6, 4]. The prior approaches, however, partly bring us back to the downside of traditional approaches: The difficulty in maintenance increases as the number of rules grows; The system quality depends on external expertise; It is hard to scale out over different domains. The data recombination approaches increase the size of training, leading to a significant increase in training time. There are various types of continual learning methods [7, 8, 9, 10, 11, 12, 13], though, in this work, we focus on model-based algorithms such as EWC [14, 15], since variants of EWC allow us to efficiently consider the importance measure of each parameter for different tasks without

[1] https://dev.botframework.com/

[2] https://developer.amazon.com/alexa-skills-kit

[3] https://labs.cognitive.microsoft.com/en-us/project-conversation-learner

[4] http://rasa.com/

expensive architectural changes.

3. CONTINUAL LEARNING FOR CONVERSATIONAL AGENTS

In this section, we describe our conversation model and an adaptive online algorithm for continual learning which together allow us to sequentially train a conversation model over multiple tasks without forgetting earlier tasks.

3.1. Conversation Model

Our conversational model builds upon the Hybrid Code Network [3] which servers as a key component for the recent interactive bot development technologies such as Conversation Learner and Rasa. At a high level, our model have five components: sentence RNNs [5]; a context RNN; domain-specific software; domain-specific action templates; and a conventional entity extraction module for identifying entity mentions in text. Both the context RNN and the developer code maintain state. Each action template can be a textual communicative action or an API call. Figure 1 shows the overall operational loop. The cycle begins when the user provides an utterance, as text (step 1). Second, an utterance embedding is formed, using the sentence RNN (step 2). Third, an entity extraction module identifies entity mentions (step 3), for example, identifying "Seattle" as a $\langle$`city`$\rangle$ entity. The text and entity mentions are then passed to "Entity tracking" code provided by the developer (step 4), which grounds and maintains entities, for example, mapping the text "Seattle" to a specific row in a database. This code can optionally return "context features" which are features the developer thinks will be useful for distinguishing among actions, such as which entities are currently present and which are absent. The feature components from steps 1-4 are concatenated to form a feature vector (step 5). This vector is passed to the context RNN. The context RNN computes a hidden state (step 6), which is retained for the next timestep (step 7), and passed to a bilinear layer which projects the hidden state to the response space (step 8), yielding a response embedding. A set of embeddings for each distinct system action template are generated, using the sentence RNN (step 9). This set of embeddings get ranked according to the similarity to the response embedding (step 10). With a softmax, a probability distribution over action templates is generated (step 11). From the resulting distribution (step 12), the best action is selected (step 13). The selected action is next passed to "Entity output" developer code that can substitute in entities (step 14) and produce a fully-formed action, for example, mapping the template "$\langle$city$\rangle$, right?" to "Seattle, right?". In step 15, control branches depending on the type of the action: if it is an API action, the corresponding API call in the developer code is invoked (step 16), for example, to render rich content to the user. APIs can act as sensors and return features relevant to the dialog, so these can be added to the feature vector in the next timestep (step 17). If the action is text, it is rendered to the user (step 18), and cycle then repeats. The action taken is provided as an embedding to the context RNN in the next timestep (step 19). Note that there are a few improvements in our model over the HCN model, we encode system actions using RNNs rather than just featurizing the system action taken with a binary vector which is all zero values except for the index of the taken action; we rank a set of candidate system actions by matching them with the context rather than performing classification without looking into the actions.

[5] All sentence RNNs share the parameters.

3.2. Adaptive Elastic Weight Consolidation

In order to achieve continual learning, we need to minimize the total loss function summed over all tasks, $\mathcal{L} = \Sigma_\mu L_\mu$, without access to the true loss functions of prior tasks. A catastrophic forgetting arises when minimizing L_μ leads to an undesirable increase in the loss on prior tasks L_ν with $\nu < \mu$. Variants of the EWC algorithm tackle this problem by optimizing a modified loss function:

$$\tilde{L}_\mu = L_\mu + \underbrace{c \sum_k \Omega_k^\mu (\bar{\theta}_k - \theta_k)^2}_{\text{surrogate loss}} \tag{1}$$

where c represents an weighting factor between prior and current tasks, θ all model parameters introduced in Section 3.1, $\bar{\theta}_k$ the parameters at the end of the previous task and Ω_k^μ regularization strength per parameter k. The bigger Ω_k^μ, the more influential is the parameter. EWC defines Ω^μ, for example, to be a point estimate which is equal to the diagonal entries of the *Fisher information matrix* at the final parameter values. Since EWC relies on a point estimate, we empirically noticed that sometimes Ω^μ fails to capture the parameter importance when the loss surface is relatively flat around the final parameter values as Ω^μ essentially decreases to zero.

In contrast to EWC, [15] computes an importance measure online by taking the path integral of the change in loss along the entire trajectory through parameter space. Specifically, the per-parameter contribution ω_k^μ to changes in the total loss is defined as follows:

$$\omega_k^\mu = -\int_{t^{\mu-1}}^{t^\mu} g_k(\theta(t)) \theta_k'(t) dt \tag{2}$$

where $\theta(t)$ is the parameter trajectory as a function of time t, $g(\theta) = \frac{\partial L}{\partial \theta}$ and $\theta_k'(t) = \frac{\partial \theta}{\partial t}$. Note that the minus sign indicates that we are interested in decreasing the loss. In practice, we can approximate ω_k^μ as the sum of the product of the gradient $g_k(t)$ with the parameter update $\Delta_k(t)$. Having defined ω_k^μ, Ω_k^μ is defined such that the regularization term carries the same units as the loss by dividing ω_k^μ by the total displace-

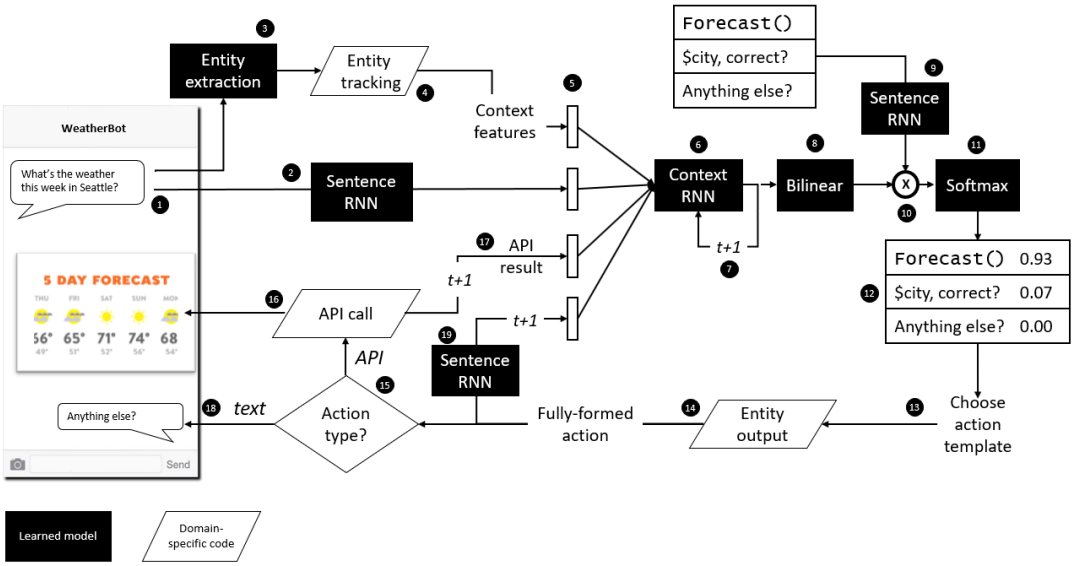


Fig. 1. The overall operational loop. Trapezoids refer to programmatic code provided by the software developer, and shaded boxes are trainable components.

ment in parameter space:

$$\Omega_k^\mu = \sum_{\nu<\mu} \frac{\omega_k^\nu}{(\Delta_k^\nu)^2 + \zeta} \tag{3}$$

where Δ_k^ν quantifies how far the parameter moved during the training process for task ν. ζ is introduced to keep the expression from exploding in cases where the denominator gets close to zero. Note that, with this definition, the quadratic surrogate loss in (1) yields the same change in loss over the parameter displacement Δ_k as the loss function of the previous tasks.

However, note that unlike prior studies on weight consolidation, where knowledge transfer is conducted between similar tasks, our tasks have a large difference in the number of distinct action templates. This leads to a huge difference in the scale of the loss and, in turn, makes the value of importance measure incomparable between different tasks. Thus, to make the importance measures comparable, we rescale the losses of each task by dividing it by $-\log \frac{1}{n}$ where n is the number of distinct action templates for each task, assuming the initial loss is governed by random guesses. We call this version of EWC *Adaptive Elastic Weight Consolidation* (AEWC).

4. EXPERIMENTS

4.1. Data

To test our method, we used two task-oriented dialog datasets: `bAbI6` [16] and Google multi-domain dialog datasets (`GMD`) [17]. Basic statistics of the datasets are shown in Table 1 and Table 2. `bAbI6` deals with restaurant finding tasks. `GMD` contains two different tasks – buying a movie ticket and reserving a restaurant table. [6] We generated distinct action templates by replacing entities with slot types and consolidating based on dialog act annotations.

Metric	bAbI6		
	Train	Dev	Test
Dialogs	1618	500	1117
Avg. turns per dialog	20.08	19.30	22.07
Avg. tokens per user turn	3.14	3.17	3.10
Avg. tokens per system turn	9.94	9.95	10.59

Table 1. Data statistics of `bAbI6`. The number of distinct system actions is 58.

Metric	GMD		
	Train	Dev	Test
Dialogs	1478	460	1027
Avg. turns per dialog	9.00	7.16	7.32
Avg. tokens per user turn	6.09	6.42	6.36
Avg. tokens per system turn	22.38	22.09	22.48

Table 2. Data statistics of `GMD`. The number of distinct system actions is 441.

[6] The user's goal in `bAbI6` is to obtain specific pieces of restaurant information such as phone number or address, whereas the user's goal in `GMD` is focused on booking a table. This makes dialogs in these two datasets significantly different.

4.2. Comparative models

To test if conversational skills are successfully accumulated, we continuously train a model on two tasks, task A and B. The resulting model should work well on both task A and B. We compare four models trained by different training schemes: 1) *Weight Transfer* (WT): we train a model on task A, and continue to train the model on task B, 2) AEWC: the same as WT except for AEWC being applied. 3) L2: we use L2 loss instead of the EWC surrogate loss in Eq. 1 4) *No Transfer* (NT): for reference, we train a model just on task B. We first take `bAbI6` as task A and `GMD` as B and then switch the order.

4.3. Training details

For the sentence RNN, we use a bidirectional LSTM [18] with 100 hidden units for each direction. The context RNN is a unidirectional LSTM with 200 hidden units. We initialized all LSTM-RNNs using the *Xavier* uniform distribution [19]. The word embedding weight matrix was initialized with the GloVe embeddings with 100 dimension [20]. We used the Adam optimizer [21], with gradients computed on mini-batches of size 1 and clipped with norm value 5. The learning rate was set to 1×10^{-3} throughout the training and all the other hyperparameters were left as suggested in [21]. We performed early stopping based on the performance of the evaluation data to avoid overfitting. As a simple transfer mechanism, we initialized all weight parameters with prior weight parameters when there is a prior model. The ω_k and Δ_k are updated continuously during training, whereas the importance measure Ω_k and the prior weight $\tilde{\theta}$ are only updated at the end of each task. After updating the Ω_k, the ω_k are set to zero. We set the trade-off parameter c to 0.025, chosen from $\{0.1, 0.05, 0.025, 0.01\}$ based on validation performance.

4.4. Results

The result is shown in Table 3. Since there are multiple action templates that are appropriate for a given dialog context, we use *Recall@3* as performance metric. In both cases, i.e. B $\rightarrow$ G and G $\rightarrow$ B, AEWC successfully manages to learn task B while retaining the performance on task A. In contrast, though WT reaches a higher performance on task B, the performance on task A sharply drops down, meaning WT fails to accumulate the conversational skills learned from prior tasks. L2 places somewhere in between. This result is natural because WT has no regularization, AEWC allows for smart regularization depending on importance measure, and L2 is a simple importance-agnostic regularization. In Fig. 2, we provide a plot for G $\rightarrow$ B that shows how the performance on task A changes as we train on task B. This curves clearly demonstrate that AEWC tries to optimize on task B in such a way that it doesn't hurt task A, whereas WT only optimizes on task B, keeping decreasing the performance on task B. To obtain a better understanding of the implicit skill accumulation during training, we provide a visualization on the importance measures for different tasks in Appendix A.

		NT	WT	L2	AEWC
B→G	G	55.09	**52.91**	24.97	48.42
	B	3.60	57.49	65.69	**65.99**
G→B	B	68.38	**68.27**	55.09	64.08
	G	2.82	19.78	38.31	**53.66**

Table 3. Experimental results on conversational skill accumulation. "B" and "G" stand for `bAbI6` and `GMD`, respectively. The first column represents the ordering of training datasets. The second column indicates test datasets.

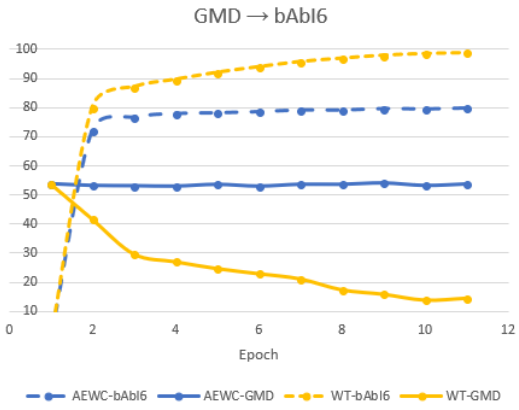


Fig. 2. Performance curves of AEWC and WT on `GMD` and `bAbI6` when training on `bAbI6` after `GMD`.

5. CONCLUSION

We have presented a continual learning-based approach to repurposing already trained neural conversational models to address the data-intensiveness problem. We tested our method through conversation skill accumulation across different task-oriented domains. Future work includes experiments with real users to evaluate actual dialog success for a complex task with multiple subtasks.

6. REFERENCES

[1] Robert M French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[2] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young, "A network-based end-to-end trainable task-oriented dialogue system," *arXiv preprint arXiv:1604.04562*, 2016.

[3] Jason D Williams, Kavosh Asadi, and Geoffrey Zweig, "Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning," *arXiv preprint arXiv:1702.03274*, 2017.

[4] Tiancheng Zhao, Allen Lu, Kyusong Lee, and Maxine Eskenazi, "Generative encoder-decoder models for task-oriented spoken dialog systems with chatting capability," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, 2017, pp. 27–36.

[5] Arash Eshghi, Igor Shalyminov, and Oliver Lemon, "Bootstrapping incremental dialogue systems from minimal data: the generalisation power of dialogue grammars," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2210–2220.

[6] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young, "Multi-domain neural network language generation for spoken dialogue systems," in *Proceedings of the 2016 Conference on North American Chapter of the Association for Computational Linguistics (NAACL)*, June 2016.

[7] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[8] Sang-Woo Lee, Chung-Yeon Lee, Dong-Hyun Kwak, Jiwon Kim, Jeonghee Kim, and Byoung-Tak Zhang, "Dual-memory deep learning architectures for lifelong learning of everyday human behaviors.," in *IJCAI*, 2016, pp. 1669–1675.

[9] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[10] Zhizhong Li and Derek Hoiem, "Learning without forgetting," in *European Conference on Computer Vision*. Springer, 2016, pp. 614–629.

[11] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim, "Less-forgetting learning in deep neural networks," *arXiv preprint arXiv:1607.00122*, 2016.

[12] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[13] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber, "Compete to compute," in *Advances in neural information processing systems*, 2013, pp. 2310–2318.

[14] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, p. 201611835, 2017.

[15] Friedemann Zenke, Ben Poole, and Surya Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*, 2017, pp. 3987–3995.

[16] Antoine Bordes and Jason Weston, "Learning end-to-end goal-oriented dialog," *arXiv preprint arXiv:1605.07683*, 2016.

[17] Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck, "Building a conversational agent overnight with dialogue self-play," *arXiv preprint arXiv:1801.04871*, 2018.

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

[19] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

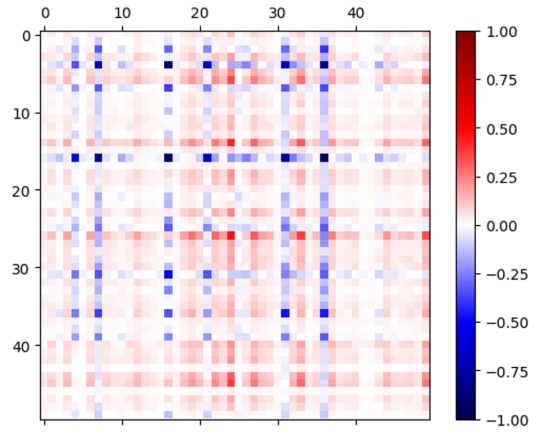
[20] Jeffrey Pennington, Richard Socher, and Christopher D Manning, "Glove: Global vectors for word representation," *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, vol. 12, pp. 1532–1543, 2014.

[21] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *The International Conference on Learning Representations (ICLR).*, 2015.

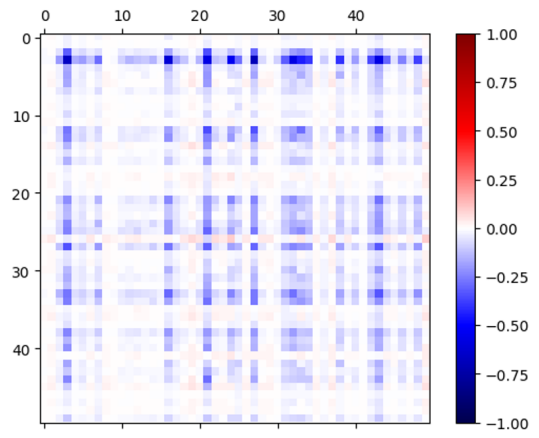
Appendices

A. WEIGHT IMPORTANCE VISUALIZATION

To obtain a better understanding of the implicit skill accumulation during training, we visualized the difference of normalized values of the importance measures across different tasks. Due to space limitations, we sampled a set of parameters from the model that was continuously trained on `GMD` and `bAbI6`, and then plotted a heatmap of the difference,



(a) AEWC



(b) No AEWC

Fig. 3. Parameter importance visualization. The intensity of red color indicates the strength of the importance of each parameter for GMD whereas blue color for bAbI6.

i.e., $\Omega^{\texttt{bAbI6}} - \Omega^{\texttt{GMD}}$ in Fig. 3. The intensity of red color indicates the strength of the importance of each parameter for GMD, whereas blue color for bAbI6. Fig. 3a shows that, with AEWC, parameters that contribute to decreasing the loss of different tasks are clearly separated, thus avoiding the catastrophic forgetting problem. On the contrary, there are much fewer parameters in red in Fig. 3b. This means that the learning process does not consider the importance of each parameter for GMD when it trains on bAbI6.