Continual Learning with Neural Networks: A Review

Abhijeet Awasthi
IIT Bombay
awasthi@cse.iitb.ac.in

Sunita Sarawagi
IIT Bombay
sunita@iitb.ac.in

ABSTRACT

Continual learning broadly refers to the algorithms which aim to learn continuously over time across varying domains, tasks or data distributions. This is in contrast to algorithms restricted to learning a fixed number of tasks in a given domain, assuming a static data distribution. In this survey we aim to discuss a wide breadth of challenges faced in a continual learning setup and review existing work in the area. We discuss parameter regularization techniques to avoid catastrophic forgetting in neural networks followed by memory based approaches and the role of generative models in assisting continual learning algorithms. We discuss how dynamic neural networks assist continual learning by endowing neural networks with a new capacity to learn further. We conclude by discussing possible future directions.

KEYWORDS

Continual Learning, Lifelong Learning, Online Multitask Learning, Never Ending Learning, Catastrophic Forgetting, Dynamic Neural Networks

1 INTRODUCTION

Continual learning is strongly motivated by the principle of retaining and reusing previously learned knowledge to learn new tasks with lesser training time and resources. Continuous learning draws inspiration from lifelong learning in humans where concepts are build cumulatively as opposed to learning each concept from scratch. Other names for continuous learning include Lifelong Learning, Online Multitask Learning and Never Ending Learning. Formally, given sequentially arriving tasks T_1, T_2,…,T_K along with their respective datasets D_1, D_2,…,D_K, and a finite amount of memory M to store either partially trained models or data, our goal is to continually learn from these tasks such that while learning a task T_i, we make use of training data D_i along with the previously accumulated knowledge stored in M. The updated model is required to provide high accuracy on all previous tasks, not just the most recent task. Each task T_i is expected to benefit from previous tasks providing a forward flow of knowledge. Additionally, previous tasks are expected to improve after seeing new tasks providing a back flow of knowledge. Thus, in continuous learning the challenge is to accrue the full benefits of batch multi-task learning under the constraints that tasks arrive sequentially and we only have limited memory and limited computation cycles for retraining everything from scratch.

We list challenges faced by neural networks in a continual learning set up.

(1) **Catastrophic Forgetting** In neural networks, parameters learned by back-propagating errors adapt to the most recent task, rendering poor performance over previously learned tasks. This makes learning in neural networks forgetful, subject to evolving nature of task objectives or frequently changing training data distribution. Section 2 reviews regularization based methods to overcome these problems.
(2) **Memory constraints** A continual learning set-up assumes a modest space to store model parameters and training data from previous tasks. Hence, retraining models using previously observed data might not be feasible in some situations. Section 3 reviews some useful methods based on generative models and memory based models (using finite memory) to handle these constraints.
(3) **Augmenting the model capacity while promoting reuse of learned representations** With growing number of tasks to learn, there comes a need to increase the learning capacity of the model (number of trainable parameters).
Further, assuming an infinite capacity to store model parameters, training a separate model for each new task appears to be very straightforward. However, such an approach would still be inefficient as there is no provision to reuse the knowledge accumulated in past, which could otherwise allow for better generalization or quicker convergence. In Section 4, we discuss dynamic neural networks which aim to tackle these problems by judiciously increasing the model capacity while simultaneously promoting reuse of learned representations.
(4) **Semantic and Syntactic differences amongst tasks** Tasks arriving sequentially might differ in syntax of input or output (e.g. sentence classification vs sentence generation). Likewise, tasks sharing similar syntax might be semantically different (e.g. text generation for news articles vs text generation for poems). Thus, using a same model to learn all the tasks seem infeasible.

We categorize the existing works on continual learning into three broad categories. In the first category we discuss methods based on importance weighted parameter regularization which makes learning in neural networks less forgetful. In the second category we discuss data-replay based methods which either store examples from previous tasks or learn to generate new examples from previously observed data distributions. In the third category we discuss methods which grow a neural network architecture over time to accommodate new knowledge.

CoDS-COMAD '19, January 3–5, 2019, Kolkata, India

ACM ISBN 978-1-4503-6207-8/19/01…$15.00
https://doi.org/10.1145/3297001.3297062

2 REGULARIZATION-BASED APPROACHES

Catastrophic forgetting prevents a neural network to learn multiple tasks in a sequential order. In this section we discuss a few methods to overcome catastrophic forgetting by regularizing weights of a neural network. A key idea common to methods discussed here is to identify the parameters which played an important role in learning past experiences. These parameters are then protected from updates in future, whereas the unimportant parameters are trained further to learn new tasks. Formally, let Ω_k^t represent the estimated importance of parameter (weight) θ_k for learning all the tasks till task T_t (inclusive). Ω_k^t attains a higher value for important parameters and vice-versa. θ_k^t represents value of parameter θ_k after learning task T_t. While learning the task T_{t+1} with objective $\mathcal{L}_{t+1}$ we modify the learning objective according to Equation 1.

$$\tilde{\mathcal{L}}_{t+1} = \mathcal{L}_{t+1} + \lambda \sum_k \Omega_k^t \left(\theta_k - \theta_k^t\right)^2 \tag{1}$$

$$\theta_k^{t+1} = \underset{\theta_k}{\operatorname{argmin}}\ \tilde{\mathcal{L}}_{t+1} \tag{2}$$

While learning task T_{t+1}, the surrogate term in R.H.S. of Equation 1 associates an importance weighted penalty for deviation of parameters from their values at the start of training for task T_{t+1}. This preserves important parameters from changing significantly and simultaneously allows less important parameters to adapt and minimize the loss over the new task T_{t+1}. λ is a hyper-parameter used to specify relative importance of the surrogate term in comparison to the actual loss $\mathcal{L}_{t+1}$. Parameters after learning task T_{t+1} are given according to Equation 2.

The parameter importance term Ω_k^t reflects either the sensitivity of the loss function or some other network output w.r.t. the network parameter θ_k^t. [6] proposes to estimate parameter importance by computing Fisher information of parameters by treating loss in Equation 1 as negative of log-likelihood. Parameters with higher fisher information suggest higher sensitivity of loss function w.r.t. those parameters. Thus Ω_k^t is given by Equation 3, where $\mathbf{F}_{k,k}^t$ represent diagonal elements of Fisher information matrix computed w.r.t. modified loss $\tilde{\mathcal{L}}_t$ over training data.

$$\Omega_k^t = \mathbf{F}_{k,k}^t = \frac{\partial^2 \tilde{\mathcal{L}}_t}{\partial \theta_k^2} \tag{3}$$

Another approach suggested in [1] is to compute the partial derivative (g_k^t) of the l_2 norm of the transformation $\mathcal{F}_t$ learned by neural network after completion of task T_t. Thus Ω_k^t is given by Equation 4, where N is number of data samples used to estimate parameter importance and x_n is the n^{th} sample from the given data. Note that, unlike previous method we do not need label of the n^{th} sample y_n to calculate importance. This allows importance to be estimated in an unsupervised manner.

$$\Omega_k^t = \frac{1}{N} \sum_{n=1}^{N} \|g_k^t(x_n)\| \tag{4}$$

Approaches described above estimate parameter importance for a given task only after it has been learned, and do not consider the dynamics of parameter updates during the learning process. [16] considers these dynamics in their online approach to estimate parameter importance. Let g_k^t represent partial derivative of $\mathcal{L}_t$ w.r.t. to model parameter θ_k. A line integral of product of g_k^t and parameter updates is computed over the entire learning trajectory of a given task. The negative of the line integral, ω_k^t, can be shown to represent the contribution of parameter updates of θ_k towards decreasing the loss $\mathcal{L}_t$. Equation 5 gives the formula for ω_k^t where τ represents any point of time in the learning trajectory of task T_t, $\theta_k^{'}(\tau)$ denotes parameter update for θ_k at time τ. Parameter importance Ω_k^t, given by Equation 6 is computed by accumulating line integrals normalized with the square of final change ($\Delta_k^s = \theta_k^s - \theta_k^{s-1}$) in parameter values, for all the previous tasks. ξ denotes a very small value used to avoid division with a zero.

$$\omega_k^t = -\int_{\tau=t-1}^{\tau=t} g_k^t(\theta(\tau))\theta_k^{'}(\tau)d\tau \tag{5}$$

$$\Omega_k^t = \sum_{s \le t} \frac{\omega_k^s}{(\Delta_k^s)^2 + \xi} \tag{6}$$

Importance weighted parameter regularization could be generalized further in a Bayesian set-up by predicting a distribution over learned parameters instead of point estimates. [9] proposes a method based on approximate bayesian inference to estimate posterior distributions and propagate them as priors to learn subsequent tasks. Use of memory is encouraged to help in compensating the losses due to approximation.

[12] proposes a strategy "Progress and Compress" based on a dual model architecture. A network **KB** maintains knowledge gathered from previously learned tasks. Learning any new task is taken up by a randomly initialized network **N**. Learning here is a two phase process. In the Progress phase the representations in **KB** are fed to the network **N** to assist it in learning a new task. This encourages a positive forward transfer of existing knowledge in **KB**. In the compress phase, information contained in **N** is distilled to **KB**. Compress phase utilizes regularization based methods discussed above to retain past knowledge in **KB** while incorporating new knowledge from **N**.

Although insightful, effectiveness of these approaches over a large number of tasks is limited by the finite capacity of a network. In Section 4 we look at methods which enhance a neural network's capacity (learnable parameters), enabling them to learn further.

3 DATA REPLAY-BASED APPROACHES

In this section, we begin with a discussion of approaches which assume modest memory space to store examples from previous tasks and use these examples to prevent catastrophic forgetting while learning a new task. We then discuss the role of generative models which provide an alternative to memory based models by drawing pseudo examples from learned data distributions.

3.1 Memory assisted continual learning

Let $M_1, M_2, \ldots, M_K$ denote memory space allocated to tasks $T_1, T_2, \ldots, T_K$ for storing a diverse subset of examples from training data for each

task as it become available over time. A naive way to prevent catastrophic forgetting while learning a task T_t could be to interleave its data D_t with the stored data corresponding to all the tasks learned previously, thus simulating a batch multitask learning set-up. Examples from previous tasks help the network to retain previously learned mappings while incorporating new information. [7] utilizes examples stored in memory to guide parameter updates in a direction of non increasing losses corresponding to previously learned tasks, while minimizing the loss corresponding to the current task. The idea is to enforce a positive inner product between gradients of losses corresponding to the current and previously learned tasks (Equation 7). A positive inner product will cause gradient descent updates for the current task to minimize losses corresponding to previous tasks at best, and not affect them at worse, thus reducing catastrophic interference.

$$\langle g_t, g_s \rangle := \left\langle \frac{\partial \mathcal{L}(\theta, D_t)}{\partial \theta}, \frac{\partial \mathcal{L}(\theta, M_s)}{\partial \theta} \right\rangle \geq 0, \forall s < t \tag{7}$$

More often than not, gradients g_t for task T_t would disobey Equation 7, thus in place of gradients g_t, parameter updates are based on projected gradients $\tilde{g_t}$ which ensure a positive dot product with gradients of previous tasks. $\tilde{g_t}$ can be obtained by solving a linearly constrained quadratic program given by Equation 8.

$$\begin{aligned} \underset{\tilde{g_t}}{\text{minimize}} \quad & \frac{1}{2}\|g_t - \tilde{g_t}\|_2^2 \\ \text{subject to} \quad & \langle g_s, \tilde{g_t} \rangle \geq 0, \forall s < t \end{aligned} \tag{8}$$

Intuitively, we expect this approach to reduce the problem of overfitting in the naive approach because instead of replaying the same training examples every time, we use memory only to constrain the parameter updates for the current task to be in a specific direction. Here, the level of catastrophic forgetting depends upon the diversity of examples stored in memory. A higher diversity is expected to yield more reduction in catastrophic interference. Methods like Greedy K-center algorithm [4] can be used to return a diverse subset of examples spread well across the space of inputs. A possible limitation of memory-based approaches is that replaying the same subset of examples multiple times would very likely lead to over-fitting.

3.2 Generative Models assisted continual learning

A key idea common to many approaches which use generative models to assist continual learning is to replace the role of memory described in 3.1 by pseudo examples drawn from an approximation of data distribution over all the training examples observed so far. Referred as pseudo-rehearsals, these ideas are also motivated by the phenomenon of memory consolidation during sleep in mammalian brains [10]. More recently [13] proposed a dual model architecture consisting of a generative model G and an inference model S. Generative model G is responsible for generation of examples to perform pseudo-rehearsals. G approximates the data distribution observed by the model so far. Inference model S learns a mapping between input and output patterns for all the observed tasks. Let $H_t = \langle G_t, S_t \rangle$ represent the model trained till task T_t. Learning a new task T_{t+1} from data $D_{t+1} = (X_{t+1}, Y_{t+1})$ is a three step process.

In the first step, G_t generates a set of pseudo examples $\tilde{X}_t$ resembling the training examples related to all the tasks seen so far. S_t uses these examples as input to output pseudo labels $\tilde{Y}_t$ corresponding to $\tilde{X}_t$. Tuple $(\tilde{X}_t, \tilde{Y}_t)$ is represented as pseudo-data $\tilde{D}_t$.
In the second step, generator G_{t+1} is trained to learn the data distribution corresponding to set $\tilde{X}_t \cup X_{t+1}$. $\tilde{X}_t$ helps the generator to retain the knowledge of previous distribution while X_{t+1} updates the generator distribution with new information.
In the third step, S_{t+1} is learned using the set $\tilde{D}_t \cup D_{t+1}$ as supervision. Effectively, at each step, the generator simulates a batch multi-task learning setup described in Section 3.1
Methods using limited memory to replay a subset of previously seen data or doing pseudo-rehearsals do not scale well for an increasing number of tasks, with networks of a fixed capacity. The success of pseudo-rehearsals is further limited by the quality of samples provided by the generative model.

4 DYNAMIC NEURAL NETWORKS FOR CONTINUAL LEARNING

Ideas discussed so far assume an infinite capacity network for learning new tasks. In a practical scenario a finite capacity of a neural network would limit its ability to learn new tasks over time. Various works on dynamic neural networks [3] [11] [15] aim to solve this problem. These methods begin with a simplified architecture and when needed, augment the network incrementally with new components to attain satisfactory performance on the current task.

In [11] a neural network model is augmented with a separate subnetwork referred as column, responsible for learning a new task. Representations developed in columns of previously learned tasks are held fixed to avoid catastrophic interference. These representations are then fed as an additional input to the newly added column. Using previously learned representations as input enables the flow of existing knowledge to the new column. This method however suffer from scaling issues since a new column is added every time a new task is required to be learned. Inference in such networks become increasingly hard due to computations over previously added columns in order to pass signals to the new column.

Recently, [15] proposed an innovative method that grows the network only sub-linearly over time. Here, training for a new task T_t is a three step process.
In the first step, the algorithm identifies a set neurons in the network which can potentially contribute towards learning of the task T_t. Weights associated with this set of neurons are selectively trained while keeping other weights fixed, using parameter regularization to avoid catastrophic forgetting (Section 2). If performance over current task becomes satisfactory, the algorithm skips the second step.
The second step augments each layer of the network with a fixed number of hidden neurons. The weights connecting these neurons are randomly initialized. The augmented network is then trained further by updating only the newly added weights using group sparsity regularization [14]. Since not all the newly added hidden units are useful, group sparsity regularization allows the algorithm to selectively prune such units.

In the third step, the augmented network is further fine-tuned using parameter regularization to avoid catastrophic forgetting. The algorithm then identifies the pre-existing weights (weights before augmentation) which drifted beyond a certain threshold from their original values before start of training for the current task. Neurons connected by such weights are split into two, followed by re-initializing the connecting weight for one neuron and restoring the original value of connecting weight for other neuron. The network is finally fine-tuned again to obtain optimal parameter values for the current task. Problem of inference in such networks is solved by assigning a time-stamp s to each hidden unit added during the training of task T_s. While inference over examples of task T_t, only the hidden units with time-stamp less than or equal to t are considered. This forbids hidden units added during later stages of algorithm to interfere negatively with the mappings learned for previous tasks.

5 CONCLUSIONS

Compared to many machine learning algorithms, continuous learning aim to learn new information more efficiently by retaining and reusing past knowledge. This is particularly important in novel and resource poor settings where large training datasets may not be available at the onset of a learning phase. A model that trains continuously as training data arrives is more natural in such settings.

We began our discussion with ways to regularize parameters of neural network models which would allow new tasks to be learned sequentially while retaining the ability to perform on previously learned tasks. Then we discussed about methods which utilize a finite memory to store examples from previously observed data distributions and use them to retain the learned knowledge. We looked at the use of generative models which try to learn previously observed data distributions and serve as a replacement for finite memory. Then we reviewed methods which aim to fix the shortcomings of finite capacity models by augmenting the model's architecture to accommodate new information.

Considering growing access to low latency and large storage devices, we feel that constraint of limited memory to store data or models can be relaxed to a certain extent. Hence, more focus should be towards algorithms which exploit past experiences to provide superior performance along with reduced training time and resources. Our discussions have been limited to methods which use a single monolithic network to perform training and inference over all the tasks. Perhaps, it might be infeasible to solve tasks with different syntactic or semantic structure using a common architecture. Neural Module Networks [2] and End to End module networks [5] learn to compose question specific neural networks for Visual Question Answering. Efforts in similar directions to tailor task specific networks could be insightful. Many existing methods for continual learning focus on ways to overcome catastrophic forgetting. Overcoming catastrophic forgetting is necessary but not sufficient for scaling continual learning algorithms to a larger number of tasks. Efforts to enable re-usability of learned models at a level higher than per-trained embeddings deserve a special focus to enhance scalability. Evaluation of continual learning algorithms is mostly confined to toy datasets. Hence, applicability of many proposed algorithms in a real world setting remains largely unexplored.

Natural language Processing seems to be a promising domain to evaluate and leverage continual learning algorithms. It would be interesting to explore ways to sequentially learn a group of related NLP tasks like those in decaNLP [8], by using the knowledge obtained from previously learned tasks. Semantics and syntax of phrases in different tasks described by a same natural language remains broadly the same, thus making room for reusing learned information.
A major part of research in NLP is currently restricted to a handful languages which enjoy availability of abundant corpora for training. Efforts in continual learning should open doors for efficient ways to learn in resource scarce domains.

REFERENCES

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. 2017. Memory Aware Synapses: Learning what (not) to forget. *CoRR* abs/1711.09601 (2017). arXiv:1711.09601 http://arxiv.org/abs/1711.09601

[2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 39–48.

[3] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. 2017. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 874–883. http://proceedings.mlr.press/v70/cortes17a.html

[4] Teofilo F Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38 (1985), 293–306.

[5] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. *CoRR, abs/1704.05526* 3 (2017).

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526.

[7] David Lopez-Paz et al. 2017. Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*. 6470–6479.

[8] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730* (2018).

[9] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. 2017. Variational Continual Learning. *arXiv preprint arXiv:1710.10628* (2017).

[10] Anthony Robins. 1996. Consolidation in neural networks and in the sleeping brain. *Connection Science* 8, 2 (1996), 259–276.

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

[12] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. 2018. Progress Compress: A scalable framework for continual learning. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, StockholmsmÃďssan, Stockholm Sweden, 4528–4537. http://proceedings.mlr.press/v80/schwarz18a.html

[13] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. 2017. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*. 2994–3003.

[14] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*. 2074–2082.

[15] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2018. Lifelong Learning with Dynamically Expandable Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Sk7KsfW0-

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