Continual learning with direction-constrained optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

This paper studies a new design of the optimization algorithm for training deep learning models with a fixed architecture of the classification network in a continual learning framework, where the training data is non-stationary and the non-stationarity is imposed by a sequence of distinct tasks. This setting implies the existence of a manifold of network parameters that correspond to good performance of the network on all tasks. Our algorithm is derived from the geometrical properties of this manifold. We first analyze a deep model trained on only one learning task in isolation and identify a region in network parameter space, where the model performance is close to the recovered optimum. We provide empirical evidence that this region resembles a cone that expands along the convergence direction. We study the principal directions of the trajectory of the optimizer after convergence and show that traveling along a few top principal directions can quickly bring the parameters outside the cone but this is not the case for the remaining directions. We argue that catastrophic forgetting in a continual learning setting can be alleviated when the parameters are constrained to stay within the intersection of the plausible cones of individual tasks that were so far encountered during training. Enforcing this is equivalent to preventing the parameters from moving along the top principal directions of convergence corresponding to the past tasks. For each task we introduce a new linear autoencoder to approximate its corresponding top forbidden principal directions. They are then incorporated into the loss function in the form of a regularization term for the purpose of learning the coming tasks without forgetting. We empirically demonstrate that our algorithm performs favorably compared to other state-of-art regularization-based continual learning methods, including EWC [1] and SI [2].

1 Introduction

Humans are equipped with complex neurocognitive mechanisms that enable them to continually learn over time by accommodating new knowledge and transferring knowledge between correlated tasks while retaining previously learned experiences. This ability is often referred to as *continual or lifelong learning*. In a continual learning setting one needs to deal with a continual acquisition of incrementally available information from non-stationary data distributions (online learning) and avoid *catastrophic forgetting*, i.e., a phenomenon that occurs when training a model on currently observed task leads to a rapid deterioration of the model's performance on previously learned tasks. In the commonly considered scenario of continual learning the tasks come sequentially and the model is not allowed to inspect again the samples from the tasks seen in the past [3]. Within this setting, there exist two types of approaches that are complementary and equally important in the context of solving the continual learning problem: i) methods that assume fixed architecture of deep model and focus on designing the training strategy that allows the model to learn many tasks (note that a human brain stops growing at a certain age [4], which further motivates these methods from the biological perspective) and ii) methods that rely on existing training strategies (mostly SGD [5] and its variants,

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

which themselves suffer catastrophic forgetting [6]) and focus on expanding the architecture of the network to accommodate new tasks. In this paper we focus on the first framework.

Training a network in a continual learning setting, when the tasks arrive sequentially, requires solving many optimization problems, one per task. A space of solutions (i.e., network parameters) that correspond to good performance of the network on all encountered tasks determine a common manifold of plausible solutions for all these optimization problems. In this paper we seek to understand the geometric properties of this manifold. In particular we analyze how this manifold is changed by each new coming task and propose an optimization algorithm that efficiently searches through it to recover solutions that well-represent all already-encountered tasks. Our contribution to the existing literature relies on developing a continual learning algorithm that explicitly relies on the characteristics of the manifold shared between tasks. What is new in this paper? To the best of our knowledge, the analysis of the deep learning loss landscape that determines the shape of this manifold, the algorithm, and the experimental results are all new here.

The paper is organized as follows: Section 2 reviews recent progress in the research area of continual learning, Section 3 provides empirical analysis of the geometric properties of the deep learning loss landscape and builds their relation to the continual learning problem, Section 4 introduces our algorithm that we call DCO since it is based on the idea of direction-constrained optimization, Section 5 contains empirical evaluations, and finally Section 6 concludes the paper. Additional results are contained in the Supplement.

2 Related Work

Continual learning and the catastrophic forgetting problem has been addressed in a variety of papers. A convenient literature survey dedicated to this research theme was recently published [3]. The existing approaches can be divided into three categories [3, 7]: regularization-based methods, dynamic architecture methods, and replay techniques. We discuss here the first family of methods and defer the description of dynamic architecture methods and replay methods to the Supplement as they are not directly related to the setting considered in this paper (in our setting we do not allow the architecture of the classifier to dynamically change and we do not use replay).

Regularization-based methods modify the objective function by adding a penalty term that controls the change of model parameters when a new task is observed. In particular these methods ensure that when the model is being trained on a new task, the parameters stay close to the ones learned on the tasks seen so far. EWC [1] realizes that using tasks' Fisher information matrices to measure the overlap of tasks. SI [2] introduces the notion of synaptic importance, enabling the assessment of the importance of network parameters when learning sequences of classification tasks, and penalizes performing changes to the parameters with high importance when training on a new task in order to avoid overwriting old memories. Relying on the importance of the parameters of a neural network when learning a new task is also a characteristic feature of another continual learning technique called MAS [8]. The RWALK method [9] is a combination of an efficient variant of EWC and a modified SI technique that computes a parameter importance score based on the sensitivity of the loss over the movement on the Riemannian manifold. Additionally, RWALK stores a small subset of representative samples from the previous tasks and uses them while training the current task, which is essentially a form of a replay strategy (replay strategies are discussed in the Supplement). Finally, the recently proposed OGD algorithm [10] relies on constraining the parameters of the network to move within the orthogonal space to the gradients of previous tasks. This approach is memory-consuming and not scalable as it requires saving the gradient directions of the neural network predictions on previous tasks. All methods discussed so far constitute a family of techniques that keep the architecture of the network fixed. The algorithm we propose in this paper also belongs to this family.

Another regularization method called LwF [11] optimizes the network both for high accuracy on the next task and for preservation of responses on the network outputs corresponding to the past tasks. This is done using only examples for the next task. The encoder-based lifelong learning technique [12] uses per-task under-complete autonecoders to constraint the features from changing when the new task arrives, which has the effect of preserving the information on which the previous tasks are mainly relying. Both these methods fundamentally differ from the aforementioned techniques and the approach we propose in this paper in that they require a separate network output for each task. Finally, P&C [13] builds upon EWC and takes advantage of the knowledge distillation mechanism to preserve and compress the knowledge obtained from the previous tasks. Such a mechanism could as well be incorporated on the top of SI, MAS, or our technique.

3 Loss landscape properties

The experimental observations provided in this section extend and complement the behavior characterization of SGD [14] connecting its dynamics with random landscape theory that stems from physical systems. The details of the experimental setup of this section can be found in the Supplement (Section 8). Consider learning only one task. We analyze the top principal components of the trajectory of SGD *after convergence*, i.e., after the optimizer reached a saturation level[1]. Let x^* denotes the value of the parameters in the beginning of the saturation phase. The convergence trajectory will be represented as a sequence of optimizer steps, where each step is represented by the change of model parameters that the optimizer induced (gradient). We consider n steps after model convergence and compute the gradient of the loss function at these steps that we refer to as $\nabla L(x_1;\zeta_1), \nabla L(x_2;\zeta_2), \ldots, \nabla L(x_n;\zeta_n)$ (x_i denotes the model parameters at the i^{th} step and ζ_i denotes the data mini-batch for which the gradient was computed at that step). We use them to form a matrix $G \in \mathbb{R}^{d\times n}$ (i-th column of the matrix is $\nabla L(x_i;\zeta_i)$) and obtain the eigenvectors $\{v_i\}$ of GG^T. We furthermore define the averaged gradient direction $\bar{g} = \frac{1}{n}\sum_{i=1}^{n} \nabla L(x_i;\zeta_i)$. We first study the landscape of the deep learning loss function along directions v_i and $\bar{g}$, i.e., we analyze the function

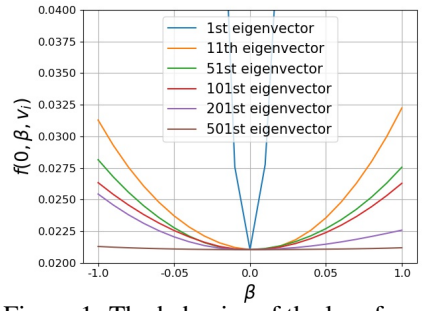

Figure 1: The behavior of the loss function for $\alpha = 0$ and varying β when moving along different eigenvectors (zoomed plot; the original plot is in the Supplement, Figure 5).

$$f(\alpha, \beta, v_i) = L(x^* - \alpha\bar{g} + \beta v_i; \zeta), \tag{1}$$

where α and β are the step sizes along $-\bar{g}$ and v_i respectively and ζ is the entire training data set.

Remark: Below, the eigenvector with the lower-index corresponds to a larger eigenvalue.

Observation 1: Behavior of the loss for $\alpha = 0$ and changing β For each eigenvector v_i, we first fix α to 0 and change β in order to study the behavior of $f(0, \beta, v_i)$. Figure 1 captures the result. It can be observed that as the model parameters move away form optimal point x^* the loss gradually increases. At the same time, the rate of this increase depends on the eigendirection that is followed and grows faster while moving along eigenvectors with the lower-index. Thus we have empirically shown that *the loss changes more slowly along the eigenvectors with the higher-index, i.e., the landscape is flatter along these directions.*

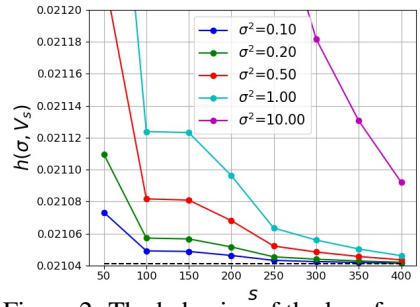

Figure 2: The behavior of the loss function when varying σ and s (zoomed plot; the original plot is in the Supplement, Figure 6).

Observation 2: Behavior of the loss in the subspaces spanned by groups of eigenvectors Here we generalize Observation 1 to the subspaces spanned by a set of eigenvectors. For the purpose of this observation only we consider the following metric instead of the one given in Equation 1:

$$h(\sigma, V_s) = \mathbb{E}_{\delta\sim\mathcal{N}(0,\frac{\sigma^2}{d}I)} f(x^* + V_s V_s^T \delta), \tag{2}$$

where δ is the random perturbation, σ is the standard deviation, and $V_s = [v_{s-49}, v_{s-48}, \cdots, v_s]$ is the matrix of eigenvectors of 50 consequtive indexes. To be more concrete, we locally (in the ball of radius σ around x^*) sample the space spanned by the eigenvectors in V_s. The expectation is computed over 3000 random draws of δ. In Figure 2 we examine the behavior of $h(\sigma, V_s)$ for various values of σ and s. The plot confirms what was shown in Observation 1 that the loss landscape becomes flatter in the subspace spanned by the eigenvectors with high index.

Observation 3: Behavior of the loss for changing α and β We generalize Observation 1 and examine what happens with $f(\alpha, \beta, v_i)$ when both α and β change. Figure 3 captures the result. We can see that as α increases, or in other words *as we go further along the averaged gradient direction, the loss landscape becomes flatter. This property holds for an eigenevctor with an arbitrary index.* Thus for larger values of α we can go further along eigenvector directions without significantly

[1]The optimization process is typically terminated when the loss starts saturating but we argue that running the optimizer further gives benefits in the continual learning setting.

changing the loss. This can be seen as a *cone* that expands along $-\bar{g}$. Furthermore, the findings of Observation 1 are also confirmed in Figure 3. For the eigenvectors with higher index the loss changes less rapidly (the cone is wider along these directions). These properties underpin the design of new continual learning algorithm proposed in this work. When adding the second task, the algorithm constrains the optimizer to stay within the cone of the first task. Inuititively this can be done by first pushing the optimizer further into the cone along $-\bar{g}$ and then constraining the optimizer from moving along eigenvectors with low indexes in order to prevent forgetting the first task. This procedure can be generalized to an arbitrary number of tasks as will be shown in the next section.

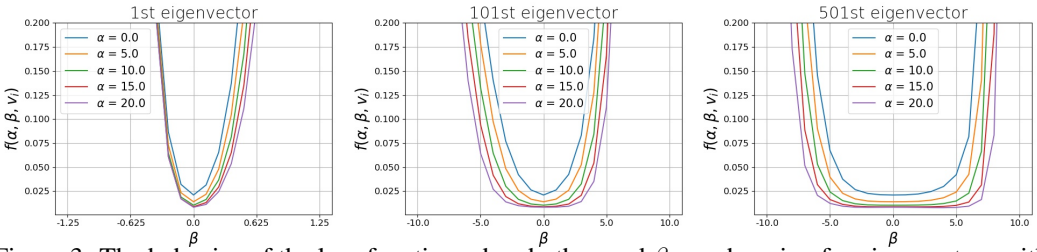

Figure 3: The behavior of the loss function when both α and β are changing for eigenvectors with different index.

4 Algorithm

In Section 3 we analyzed the loss landscape for a single task and discovered the existence of the cone in the model's parameter space where the model sustains good performance. We then discussed the consequence of this observation in the continual learning setting. In this section we propose a *tractable* continual learning algorithm that for each task finds its cone and uses it to constrain the optimization problem of learning the following tasks. We refer to the model that is trained in the continual learning setting as $\mathcal{M}$. The proposed algorithm relies on identifying the top directions along which the loss function for a given task increases rapidly and then constraining the optimization from moving along these directions (we will refer to these directions as "prohibited") when learning subsequent tasks. Note that each new task adds "prohibited" directions. In order to efficiently identify and constrain the "prohibited" directions we use compressed autoencoders whose design was tailored for the purpose of the proposed algorithm. We train separate autoencoders for each learned task. The j^{th} autoencoder admits on its input gradients of the loss function that are obtained when training the model $\mathcal{M}$ on the j^{th} task. The intuitive idea behind this approach is that autoencoder with small feature vector will capture the top directions of the gradients it is trained on. We refer to our method as *direction-constrained optimization* (DCO) method.

4.1 Loss function

In this section we explain the loss function that is used to train the model $\mathcal{M}$ in a continual learning setting. We incorporate a regularization term into the loss function that penalizes moving along the "prohibited" directions. The loss function that is used to train the model on the i^{th} task takes the form:

$$L_i(x;\xi) = L_{ce}(x;\xi) + \lambda \sum_{j=1}^{i-1} \left\| ENC^j(x - x_j^*) \right\|_2^2, \tag{3}$$

where ξ is a training example, L_{ce} is a cross-entropy loss, λ is a hyperparameter controlling the regularization, $ENC^j(\cdot)$ denotes the operation of the encoder of the autoencoder trained on task j, and x_j^* are the parameters of model $\mathcal{M}$ obtained at the end of training the model on the j^{th} task.

4.2 Compressed linear autoencoders

In our algorithm, the role of autoencoder is to identify the top k directions of the optimizer's trajectory after convergence, where this trajectory is defined by gradient steps, obtained during training the model $\mathcal{M}$. A traditional linear autoencoder, consisting of two linear layers, would require $2 \times d \times k$ number of parameters, where d denotes the number of parameters of the model $\mathcal{M}$. Commonly used deep learning models however contain millions of parameters [15, 16, 17], which makes a traditional autoencoder not tractable for this application. In order to reduce the memory footprint of the autoencoder we propose an architecture that is inspired by the singular value decomposition. The proposed autoencoder admits a matrix on its input and is formulated as

$$AE(M) = Udiag(U^\top MV)V^\top, \tag{4}$$

where $diag(U^\top MV)$ is a matrix formed by zeroing out the non-diagonal elements of $U^\top MV$, M is an autoencoder input matrix of size $m \times n$, and U and V are autoencoder parameters of size

Algorithm 1 DCO Algorithm

Require:
η and η_a: learning rates of the model and autoencoders respectively. $\gamma \in (0, 1]$: pulling strength that controls the searching scope of the model parameters. N: number of additional epochs used to train the model after saturation. m: the size of the batch of gradients fed into autoencoders. τ: the period of updates of the model parameters in step 3. n: number of tasks. $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$: training data from task $1, 2, \ldots, n$.

for $i = 1$ **to** n **do**
  # step 1: train model until convergence
  repeat
    $\xi \leftarrow$ randomly sample from $\mathcal{T}_i$
    $x \leftarrow x - \eta \nabla_x L_i(x; \xi)$
  until convergence
  $x^* \leftarrow x$ # Store model parameters

  # step 2: continue training model for N additional epochs
  for $j = 1$ **to** N **do**
    repeat
      $\xi \leftarrow$ randomly sample from $\mathcal{T}_i$
      $x \leftarrow x - \eta \nabla_x L_{ce}(x; \xi) - \gamma(x - x^*)$
    until all samples are iterated
  end for
  $x_i^* \leftarrow x$ # Store model parameters

  # step 3: train autoencoder until convergence
  repeat
    $g \leftarrow 0, G \leftarrow \{\}$
    for $j = 1$ **to** m **do**
      $\xi \leftarrow$ randomly sample from $\mathcal{T}_i$
      $g \leftarrow g + \nabla_x L_{ce}(x; \xi)$ # Accumulate gradients
      $G \leftarrow G \cup \nabla_x L_{ce}(x; \xi)$ # Add gradients to the batch
      if τ divides j **then**
        $x \leftarrow x - \eta g$
        $g \leftarrow 0$
      end if
    end for
    $G \leftarrow \frac{G}{\sqrt{\|G\|_2^2/m}}$ # Normalize batch of gradients
    $W \leftarrow W - \frac{\eta_a}{m} \nabla_W L_{mse}(W; G)$ # Update autoencoder parameters
    $x \leftarrow x - \gamma(x - x_i^*)$
  until convergence
  $W_i \leftarrow W$ # Store autoencoder parameters
  $x \leftarrow x_i^*$ # Restore model parameters
end for
Output x^*

$m \times k$ and $n \times k$ respectively. Thus, the total number of parameters of the proposed autoencoder is $k(n + m)$, which is significantly lower than in case of traditional autoencoder (knm), especially when n and m are large.

We use a separate encoder ENC_l and decoder DEC_l for each layer l of the model $\mathcal{M}$. We couple them between layers using a common "feature vector" which is created by summing outputs of all encoders. The proposed autoencoder is then formulated as

$$AE(G) = \{DEC_1(ENC(G)), \ldots, DEC_L(ENC(G))\}, \tag{5}$$

where

$$ENC(G) = \sum_l ENC_l(G_l), ENC_l(G_l) = diag(U_l^\top G_l V_l), DEC_l(ENC(G)) = U_l ENC(G) V_l^\top,$$

$G = \{G_1, G_2, \ldots, G_L\}$ is a set of matrices such that each matrix contains gradients of the model for a given layer, and L is number of layers in the model. Finally, in order to enable processing the gradients of the convolutional layers we reshape them from their original size $o \times i \times w \times h$ to $o \times iwh$, where o is number of output channels, i is number of input channels, and w and h are width and height of the kernel of the convolutional layer. We train the autoencoder with standard mean square error loss $L_{mse}(W; G) = \|AE(G) - G\|_2^2$, where $W = \{U_1, V_1, \ldots, U_L, V_L\}$ is set of autoencoder's parameters.

4.3 Algorithm description

The proposed algorithm comprises of three steps. In the first step we train the model $\mathcal{M}$ using the loss function proposed in Equation 3 until convergence. In the second step, we continue to train the model for additional N epochs to push its parameters deeper into the cone. Finally, in step 3 we train the autoencoder. The algorithm's pseudo code is captured in Algorithm 1.

5 Experiments

In this section we compare the performance of DCO with state-of-the-art continual learning methods: EWC [1], SI [2], RWALK [9] and A-GEM [18], as well as SGD [5]. We use open source codes[2]. Note that A-GEM was proposed in a single-epoch setup originally. For a fair comparison, we run A-GEM for multiple epochs on training data. We consider three commonly used continual learning data sets: Permuted MNIST, Split MNIST, and Split CIFAR-**100**. The details of data sets, data processing, hyperparameter selection, and network architectures can be found in the Supplement.

We consider the **average error** as our metric. If we denote e_j as *test* classification error of the model on j^{th} task, then the average error E_i on i^{th} task ($j \leq i$) is defined as $E_i = \frac{1}{i}\sum_{j=1}^{i} e_j$. In Table 1 we demonstrate that DCO performs favorably compared to the baselines in terms of the final average error. In Figure 4 we show how the average error behaves when adding new tasks. The figure reveals that DCO eventually achieves better performance than other techniques on Split MNIST data set and it consistently outperforms other methods for Permuted MNIST and Split CIFAR-100.

Table 1: Average Error E_n (%)

Method	Permuted MNIST	Split MNIST	Split CIFAR-100
SGD	40.35	24.37	46.36
EWC	5.61	0.71	32.86
SI	6.66	0.82	32.25
RWALK	5.76	1.85	33.4
A-GEM	5.12	1.02	32.19
DCO	**3.81**	**0.57**	**28.22**

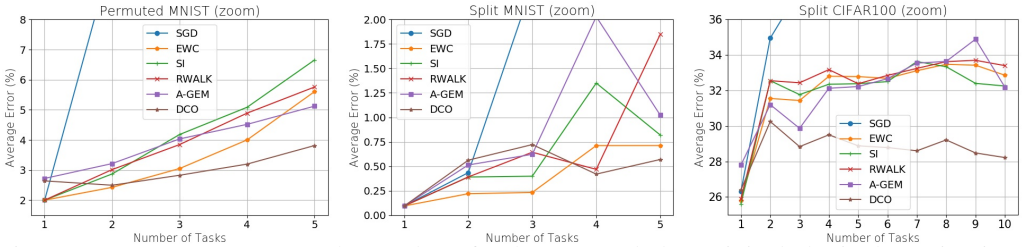

Figure 4: Average error versus the number of tasks. Zoomed plot (original plot is shown in Figure 7 in the Supplement); **Left:** Permuted MNIST **middle:** Split MNIST **right:** Split CIFAR-100.

6 Conclusion

This paper elucidates the interplay between the local geometry of a deep learning optimization landscape and the quality of a network's performance in a continual learning setting. We derive a new continual learning algorithm counter-acting the process of catastrophic forgetting that explores the plausible manifold of parameters on which all tasks achieve good performance based on the knowledge of its geometric properties. Experiments demonstrate that this online algorithm achieves improvement in performance compared to more common approaches, which makes it a plausible method for solving a continual learning problem. Due to explicitly characterizing the manifold shared between the tasks, our work potentially provides a tool for better understanding how quickly the learning capacity of the network with a fixed architecture is consumed by adding new tasks and identifying the moment when the network lacks capacity to accommodate new coming task and thus has to be expanded. This direction will be explored in the future work.

[2]https://github.com/facebookresearch/agem

References

[1] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 114(13):3521–3526, 2017.

[2] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017.

[3] G. Parisi, R. Kemker, J. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2018.

[4] N. Gogtay, J. N. Giedd, L. Lusk, K. M. Hayashi, D. Greenstein, A. C. Vaituzis, T. F. Nugent, D. H. Herman, L. S. Clasen, A. W. Toga, et al. Dynamic mapping of human cortical development during childhood through early adulthood. *PNAS*, 101(21):8174–8179, 2004.

[5] L. Bottou. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[6] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *ICLR*, 2014.

[7] S. Farquhar and Y. Gal. Towards robust evaluations of continual learning. *CoRR*, abs/1805.09733, 2018.

[8] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, 2018.

[9] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, 2018.

[10] M. Farajtabar, N. Azizan, A. Mott, and A. Li. Orthogonal gradient descent for continual learning. *CoRR*, abs/1910.07104, 2019.

[11] Z. Li and D. Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[12] A. Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars. Encoder based lifelong learning. In *ICCV*, 2017.

[13] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. In *ICML*, 2018.

[14] Y. Feng and Y. Tu. How neural networks find generalizable solutions: Self-tuned annealing in deep learning. *CoRR*, abs/2001.01678, 2020.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[18] A. Chaudhry, M. A. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019.

[19] R. Aljundi, P. Chakravarty, and T. Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *CVPR*, 2017.

[20] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[21] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. In *ICLR*, 2018.

[22] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *ICML*, 2019.

[23] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, 2018.

[24] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018.

[25] C.-Y. Hung, C.-H. Tu, C.-E. Wu, C.-H. Chen, Y.-M. Chan, and C.-S. Chen. Compacting, picking and growing for unforgetting continual learning. In *NeurIPS*, 2019.

[26] D. Isele and A. Cosgun. Selective experience replay for lifelong learning. In *AAAI*, 2018.

[27] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio. Gradient based sample selection for online continual learning. In *NeurIPS*, 2019.

[28] D. Lopez-Paz and M. A. Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, 2017.

[29] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *ICLR*, 2019.

[30] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *NeurIPS*, 2017.

[31] M. Rostami, S. Kolouri, and P. K. Pilly. Complementary learning for overcoming catastrophic forgetting using experience replay. In *IJCAI*, 2019.

[32] D. Rao, F. Visin, A. Rusu, R. Pascanu, Y. W. Teh, and R. Hadsell. Continual unsupervised representation learning. In *NeurIPS*. 2019.

[33] X. He, J. Sygnowski, A. Galashov, A. A. Rusu, Y. W. Teh, and R. Pascanu. Task agnostic continual learning via meta learning. *CoRR*, abs/1906.05201, 2019.

[34] C. Zeno, I. Golan, E. Hoffer, and D. Soudry. Task agnostic continual learning using online variational bayes. *CoRR*, abs/1803.10123, 2018.

[35] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[36] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 and cifar-100 datasets. `https://www.cs.toronto.edu/kriz/cifar.html`, 2009.

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.