OVERCOMING CATASTROPHIC FORGETTING VIA MODEL ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning multiple tasks sequentially is important for the development of AI and lifelong learning systems. However, standard neural network architectures suffer from catastrophic forgetting which makes it difficult to learn a sequence of tasks. Several continual learning methods have been proposed to address the problem. In this paper, we propose a very different approach, called *model adaptation*, to dealing with the problem. The proposed approach learns to build a model, called the *solver*, with two sets of parameters. The first set is shared by all tasks learned so far and the second set is dynamically generated to adapt the solver to suit each individual test example in order to classify it. Extensive experiments have been carried out to demonstrate the effectiveness of the proposed approach.

1 INTRODUCTION

It is well-known that neural networks (NN) suffers from *catastrophic forgetting* (CF) (McCloskey & Cohen, 1989), which refers to the phenomenon that when learning a sequence of tasks, the learning of each new task may cause the NN to forget the models learned for the old tasks. Without solving this problem, an NN is hard to adapt to continual learning, which is important for AI. We humans can learn many tasks sequentially without forgetting the knowledge learned for old tasks.

Problem Statement *Given a sequence of supervised learning tasks* $\mathbf{T} = (T_1, T_2, \ldots, T_N)$*, we want to learn them one by one in the given sequence such that the learning of each new task will not forget the models learned for the previous tasks.*

In recent years, many approaches (often called *continual learning*) have been proposed to lessen the effect of CF (Parisi et al., 2018), e.g., dynamically expandable network (DEN) (Yoon et al., 2018), learning without forgetting (LWF) (Li & Hoiem, 2016)), elastic weight consolidation (EWC) (Kirkpatrick et al., 2017), incremental moment matching (IMM) (Lee et al., 2017), gradient episodic memory (GEM) (Lopez-Paz et al., 2017), generative replay (GR) Shin et al. (2017)), etc. These existing studies except DEN and LWF focused on learning a model parameterized by a single joint set of parameters θ^* which is assumed to work well for all tasks. We call them *joint parameterization* (JP) methods. DEN and LWF required constantly increasing the number of parameters and thus can result in a huge and complex model.

However, JP methods suffer from *accuracy deterioration*. Assume we have two tasks A and B that need to be learned sequentially. Let θ^*_A and θ^*_B be the optimal parameters for Task A and Task B respectively where each of them is learned individually. Let θ^* be the joint parameters learned by the existing approaches to perform Tasks A and B in sequence. Inevitably, θ^* is different from θ^*_A and/or θ^*_B and is very likely to result in more errors for the two tasks than θ^*_A and θ^*_B individually.

In this paper, we propose a very different approach, called *model adaptation* (MA), to dealing with CF and to significantly reducing accuracy deterioration when learning a sequence of tasks. MA does not learn a joint parameter set θ^*. Instead, it learns a parameter generator $f(\cdot)$ and a shared parameter set θ_0. The key idea of MA is as follows: The overall classification network, called the *solver* S, has two disjoint subsets/parts of parameters. The first subset is θ_0, which is shared by all tasks learned so far. The second subset is just a place holder H, which will be filled by parameters generated by $f(\cdot)$ for each test instance. Specifically, in testing, given a test instance, a set of parameters will be generated by the learned parameter generator $f(\cdot)$ to replace H. Solver S then combines θ_0 and the generated parameters for the test instance to classify it. Clearly, unlike existing approaches, we do

not have a network with a set of fixed parameters for S. The idea is that θ_0 contains the common features of all tasks, and the generated parameters for each test instance adapt S for each test instance in order to classify it.

In training, when a new task T_i arrives, the system trains $f(\cdot)$ jointly with the solver S (essentially, θ_0) using the labeled training data of task T_i. The training process is such that when the parameters generated by $f(\cdot)$ for each training example combined with θ_0 of the solver S will enable S to produce the correct label for the training example. That is why in testing $f(\cdot)$ is used to generate parameters for each test instance to be used by S for classification. Note that since $f(\cdot)$ and the shared parameters θ_0 will change during training for each new task, forgetting can occur for previous tasks. To deal with forgetting, in training $f(\cdot)$ and S for each task T_i, in addition to the training data of T_i, a small number of replayed samples will be generated by a data generation network for the previous tasks to ensure that the knowledge learned for previous tasks remain stable/unforgotten. That is, the updated $f(\cdot)$ can still be used to generate parameters for examples from previous tasks and have them correctly classified by the solver S.

Although there is some resemblance between the training method of our approach and the existing generative replay methods, our method does not need labels for the replayed data because we use the replayed data only to constrain the training of S and $f(\cdot)$ to solve the forgetting problem in $f(\cdot)$ rather than to treat them as surrogates of labeled training data from previous tasks to retrain the system as in existing replay methods. Further, since in the existing replay methods, the labels are produced by the current learned network itself, the labels can be noisy and biased, which can result in errors being accumulated and propagated to subsequent tasks. Our method does not have this problem.

Apart from reducing the effect of accuracy deterioration of the existing approaches, the proposed approach also has some other advantages. First, no parameter increase or network expansion is needed to learn new tasks. Second, no previous data needs to be stored to enable the system to remember the previous learned models or knowledge.

Extensive experiments conducted using two image datasets (MNIST and CIFAR-10) and two text datasets (DBPedia ontology[1] (Lehmann et al., 2015) and THUCNews[2] (Li et al., 2006)) show that the proposed approach works well for different scenarios and different types of datasets, and outperforms the existing state-of-the art baselines markedly.

2 Proposed Model Adaptation Framework

The proposed Model Adaptation (MA) approach has three main components: Solver S, Dynamic Parameter Generator (DPG) $f(\cdot)$, and Data Generator (DG). Regression losses are used to optimize the three components. We first define some terminologies and give the problem formulation, and then present the three components in detail.

Let the sequence of supervised learning tasks be $\mathbf{T} = (T_1, T_2, \ldots, T_N)$. Each task T_i is represented by $T_i = \{x_{ij}, y_{ij} | j \in (1, \ldots, N_i)\}$, where x_{ij} is the j-th input data of T_i, and y_{ij} is its corresponding label. We use (x^t, y^t) to denote a test instance which comes from the task set T. We further define $x_i = \{x_{ij}\}_{j=0}^{J}$, $y_i = \{y_{ij}\}_{j=0}^{J}$, $X = \{x_i\}_{i=0}^{N}$ and $Y = \{y_i\}_{i=0}^{N}$.

2.1 Overall Approach

Let us work backward by describing testing first before describing training. As mentioned in the introduction section, the parameter set of S consists of two subsets, θ_0 and a parameter place holder H, which will be replaced by the parameters generated by DPG $f(\cdot)$. Given a test instance x^t, $f(\cdot)$ first generates a set of parameters P^t to replace H. Solver S then uses θ_0 and P^t to classify x^t. θ_0 is supposed to contain the common features of all tasks learned so far. The generated parameters for each test instance simply adapts S for x^t in order to classify x^t. We use this parameter split as several studies (Yoon et al., 2018; Li & Hoiem, 2016; Kirkpatrick et al., 2017) have shown that only part of parameters of a neural network need to be adjusted when learning a new task.

[1] https://github.com/guoyinwang/LEAM

[2] http://thuctc.thunlp.org/message

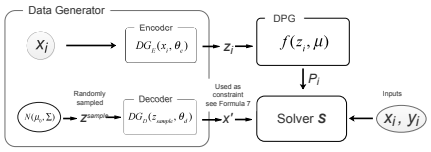


Figure 1: Sequential training in the proposed MA framework.

Figure 1 shows the pipeline of the proposed MA framework for training. Given a new task T_i with its data (x_i, y_i), solver S and DPG $f(\cdot)$ are jointly trained to learn T_i and also not to forget the previously learned tasks. In each iteration, a set of parameter sets P_i (one per training example) is generated by the current DPG $f(\mathbf{z}_i, \mu)$, where μ is the set of parameters of the DPG network, which is trained, and $\mathbf{z}_i$ is the set of compressed or dense (low dimensional) representations (one per training example) of the training data x_i of T_i and $\mathbf{z}_i = DG_E(x_i, \theta_e)$, where DG_E is the encoder of the data generator (DG) with parameters θ_e. We use $\mathbf{z}_i$ rather than the raw data x_i because x_i's dimension can be very high. DG_E compresses each example in x_i to a dense representation in $\mathbf{z}_i$ to reduce the mapping space for DPG and thus reduce the difficulty in its parameter generation.

In training/adapting solver S and DPG, $f(\cdot)$ and the shared parameters of θ_0 will change, which can cause forgetting in DPG for previous tasks. To keep DPG remembering the acquired knowledge for previous tasks, we minimize the variation of certain layers' output caused by the changes of θ_0 and $f(\cdot)$ using some replayed data x'. As noted earlier, our replay data do not need class labels, which thus avoid error accumulation and propagation suffered by existing replaying methods. We will discuss this further in Section 2.4. The replayed data $x' = DG_D(\mathbf{z}_{sample}, \theta'_d)$ is produced by the decoder of DG called DG_D. $\mathbf{z}_{sample}$ is sampled from the multivariate normal distribution.[3] θ'_d is the set of parameters of DG_D before optimizing the current task T_i.

To learn task T_i, the objective of solver S including DPG $f(z_i, \mu)$ and θ_0 is defined as:

$$\begin{aligned} \underset{\mu,\theta_0}{\text{minimize}} \quad & \mathcal{L}(S(x_i, \theta_i^*), y_i) \\ \text{s.t.} \quad & ||\mathcal{R}(x', \theta_i^*) - \mathcal{R}(x', \theta_{i-1}^*)|| < \epsilon_r \end{aligned} \tag{1}$$

where $\theta_i^* = \text{combine}(P_i, \theta_0)$ and $\mathcal{R}$ can be any layer (or a set of layers) [4] in solver S, which we will introduce later. We can see that the generated replay data x' is used in the constraints. Note that DPG is optimized together with solver S. DG has its own objective function (such as reconstruction loss and penalized form of the Wasserstein distance between the distribution of z_i and multivariate normal distribution (Tolstikhin et al., 2017)), but is trained alternately with DPG and solver S. That is because DPG takes input $\mathbf{z}_i$ (which is produced by DG) to generate parameters, and alternating training ensures consistent convergence rates for DG and DPG. Since we only have one data generator DG, it also has forgetting problem caused by incremental training of the new task data. To avoid forgetting in DG, we use constraints similar to Formula 1:

$$||\mathcal{G}(x', \theta_e, \theta_d) - \mathcal{G}(x', \theta_e, \theta'_d)|| < \epsilon_g \tag{2}$$

where $\mathcal{G}$ can be any layer (or a set of layers) in DG.

2.2 Dynamic Parameter Generator and Solver Implementation

Several existing neural networks can be used to implement DPG for generating parameters, e.g., convolutional neural network (CNN), recurrent neural network (RNN) and multilayer perceptron (MLP). The objective of this paper is not to explore all possible implementations. We found that a fairly straightforward implementation using MLP can already achieve good results. Formally, DPG can be written as:

$$\mathbf{P}_i = f(\mathbf{z}_i, \mu) = \sigma(\mathbf{w}\mathbf{z}_i + b) \tag{3}$$

where σ is the activation function, and $\mathbf{w}$ and b are the parameters of DPG and denoted by μ.

For the implementation of solver S, several deep learning networks can be used too. Let us first introduce the concept of **basic unit**, which is a simple MLP with one hidden layer. That is because most of the existing networks can be regarded as the composition of multiple basic units, even for those with complex structures, e.g., seq2seq (Sutskever et al., 2014), R-Net (Wang et al., 2017), Resnet (He et al., 2016) and Densenet (Huang et al., 2017).[5] A basic unit is an activation function

[3] In training, we force $\mathbf{z}_i$ to satisfy the multivariate normal distribution. We can then sample many $\mathbf{z}_i$ from the multivariate normal distribution.

[4] Note that in this case, the parameters in $\mathcal{R}$ is a subset of parameters for solver S. To simplify the expression and without causing confusion, we don't introduce new symbols and let the symbols unchanged.

[5] Complex structures give us more freedom to decide which part of the parameters should be generated.

plus a matrix transformation: $output = \sigma(\mathbf{w}\mathbf{x}_{input})$. Inspired by previous studies in (Yoon et al., 2018; Li & Hoiem, 2016; Kirkpatrick et al., 2017) which showed that only part of the parameters of an neural network needs to be adjusted when learning a new task, we propose to adapt the solver by replacing part of $\mathbf{w}$ by $\mathbf{P}_i$ rather than the entire set. Concatenation is then used to combine $\mathbf{P}_i$ and θ_0 (the remaining part of $\mathbf{w}$):

$$\theta_i^* = \{[\mathbf{w}_{i,k}^*]\}_{k=0}^K = \{[\theta_{0,k}; \mathbf{P}_{i,k}]\}_{k=0}^K \tag{4}$$

where K is the number of basic units that the solver has. In many cases, there is no need to adapt all the basic units in the solver. Only adapting a subset of them can already achieve good results based on our experiments. One concern about DPG might be that the proposed combination method may not have sufficient capacity to adjust the solver because only part of parameters are generated. Actually, $\mathbf{P}_{ik}$ can adapt all the dimension of the output of its corresponding basic unit k:

$$\mathbf{w}_{i,k}^* \mathbf{x}_{input} = [\theta_{0,k}; \mathbf{P}_{i,k}]\mathbf{x}_{input} = \theta_{0,k}\mathbf{x}_{input}^1 + \mathbf{P}_{i,k}\mathbf{x}_{input}^2 \tag{5}$$

where $\mathbf{x}_{input}^1$ and $\mathbf{x}_{input}^2$ are the block vectors of $\mathbf{x}_{input}$. $\mathbf{P}_{i,k}\mathbf{x}_{input}^2$ can be regarded as the bias and can adapt the output vector to any point in the vector space.

2.3 Data Generator (DG)

The main function of DG is to generate replay data for previous tasks to deal with forgetting in DPG. As mentioned earlier, DG differs from data generators in the existing generative replay (GR) method in (Shin et al., 2017) in that our approach only requires input data but not the labels.[6] The main advantage of our approach is that it can reduce error accumulation because the labeling error of the solver won't affect our model, but will damage (and accumulate in) GR.

In addition to generating replay past data, DG also performs feature extraction and dimensionality reduction for input data. The original input data can have a huge number of dimensions, which can greatly increase the mapping space of DPG and cause difficulties for $\mathbf{P}_i$ generation. DG thus also serves to compress the data from x_i to $\mathbf{z}_i$. Auto-encoders can do the job, especially, VAE-like (Variational Auto-Encoder (Kingma & Welling, 2013)) and WAE-like (Wasserstein Auto-Encoder) auto-encoders, which force $\mathbf{z}_i$ to satisfy *multivariate normal distribution*, and thus can help generate a lot of data by randomly sampling from the multivariate normal distribution. Several strategies are investigated to alleviate the forgetting problem in DG like in DPG using specially designed losses (see Section 2.4). DG has the ability to approximate the past data in a similar way as in Shin et al. (2017). We use WAE in DG since it can let different examples get a chance to stay far away from each other, which promotes a better reconstruction (Tolstikhin et al., 2017).[7] In summary, when solving task T_i, DG can be formulated by:

$$\mathbf{z}_i = DG_E(x_i, \theta_e); x_i' = DG_D(\mathbf{z}_i, \theta_d) \tag{6}$$

where DG_E is the encoder and DG_D is the decoder.

2.4 Objective Function

As discussed in Section 2.1, there are two main training components for MA: 1) DG, and 2) DPG and solver S (DPG&S for short). This section mainly discusses the constraints introduced above for the two components. Apart from those constraints, the two main components also have their own objective functions: 1) for DPG&S, cross entropy loss is used to learn new classification tasks; 2) for DG, reconstruction loss is used to enable its replay ability of past data, and penalized form of the Wasserstein distance between the distribution of $\mathbf{z}_i$ and multivariate normal distribution is added to help generate data (Tolstikhin et al., 2017). Unlike existing generative replay method, we propose to use constraints to alleviate forgetting. Specially, we extend the knowledge distillation loss (KDL) (Hinton et al., 2015) to the general situation. That is, to keep the past learned knowledge, the output

[6] The GR method in (Shin et al., 2017) uses a generative model to generate the past data x' and then obtains the label of the generated data y' through the solver learned in the past. Fake training pairs $\{x', y'\}$ are constructed and then used in the current training step to ensure that the acquired past knowledge is not forgotten.

[7] Our approach is not limited to using WAE. Other auto-encoders may also be applied.

of the basic units in the solver should not change much when learning new tasks with the help of the generated data. Formally, when learning task T_i, we use the following constraint:

$$\min \ \ ||\mathcal{R}(x', \theta_i^*) - \mathcal{R}(x', \theta_{i-1}^*)|| \tag{7}$$

where $\mathcal{R}(\cdot)$ denotes the output of any basic unit in the solver. If we do not consider the activation function, Formula 7 can also be written as:

$$\min \sum_{j=0}^{J} \sum_{k=0}^{K} ||\mathbf{w}_{i,k}^* \mathbf{x}'_{k,j} - \mathbf{w}_{i-1,k}^* \mathbf{x}'_{k,j}|| \tag{8}$$

where k denotes the number of basic units. The unit with a smaller k is at the relative lower layer of the solver network,[8] $\mathbf{w}_{i,k}^* = [\theta_{0,k}; \mathbf{P}_{i,k}]$, $\mathbf{x}'_{k,j}$ is the input of the current basic unit and is calculated through forward propagation, $\mathbf{x}'_{0,j} = DG_D(\mathbf{z}_{sample,j}, \theta'_d)$ is the initial data replayed by DG (before optimizing the current task). J denotes the numbers of sampled $\mathbf{z}$.

Two questions may be asked about Formulas 7 and 8. (1) since the replayed data x' is used in Formulas 7 and 8, does it play the same role as the original data x? The answer is positive as we will prove below. (2) what is the impact of the constraints on learning new tasks?

Question 1: Since the purpose of the constraints is to maintain the learned effect of the old data, clearly, using the original old data x in the constraints is better. However, we can prove the positive answer to question 1.

Proof: Our objective is to minimize $||\mathcal{R}(x, \theta_i^*) - \mathcal{R}(x, \theta_{i-1}^*)||$. Since $R(\cdot)$ is differentiable and its derivative function is bounded because it uses 1-Lipschitz activation function (e.g., RELU or tanh) and $R(\cdot)$ satisfies the Lipschitz condition. After some transformation, we obtain:

$$\begin{aligned} & ||\mathcal{R}(x, \theta_i^*) - \mathcal{R}(x', \theta_i^*) + \mathcal{R}(x', \theta_i^*) - \mathcal{R}(x', \theta_{i-1}^*) + \mathcal{R}(x', \theta_{i-1}^*) - \mathcal{R}(x, \theta_{i-1}^*)|| \\ \leq \ & ||\mathcal{R}(x, \theta_i^*) - \mathcal{R}(x', \theta_i^*)|| + ||\mathcal{R}(x', \theta_i^*) - \mathcal{R}(x', \theta_{i-1}^*)|| + ||\mathcal{R}(x', \theta_{i-1}^*) - \mathcal{R}(x, \theta_{i-1}^*)|| \\ \leq \ & \mathcal{L}_1||(x - x')|| + ||\mathcal{R}(x', \theta_i^*) - \mathcal{R}(x', \theta_{i-1}^*)|| + \mathcal{L}_2||(x - x')|| \end{aligned} \tag{9}$$

where the last inference is based on Lipschitz continuity, and $\mathcal{L}_1$ and $\mathcal{L}_2$ are Lipschitz constants. We can see that if the solver has a small Lipschitz constant or the DG's reconstruction error is small, minimizing Formula 7 is consistency with constraining using the original data.

Question 2: Since many replayed samples $\{\mathbf{x}'_{k,j}\}_{j=0}^{J}$ are used in calculating Formula 8, if we stack all those samples (column vector) into a matrix $\mathbf{X}'_k$, Formula 8 can be regarded as constraining $\mathbf{w}_{i,k}^* \mathbf{X}'_k$ to remain unchanged. If $\mathbf{X}'_k$ is a row low rank matrix, $\mathbf{w}_{i,k}^*$ can be trained to fit the new tasks. Otherwise, especially if $\mathbf{X}'_k$ is a row full rank matrix, $\mathbf{w}_{i,k}^*$ cannot be trained and therefore cannot learn new tasks. However, benefiting from dynamic parameter generation, our approach will not suffer from this problem. That is because $\mathbf{w}_{i,k}^*$ is specially generated through DPG to perform well with input $\mathbf{X}'_k$. And new parameters will be generated to fit new tasks while $\mathbf{w}_{i,k}^*$ is able to remain unchanged.

DG also suffers from forgetting due to the using of stochastic gradient descent. In addition to adapting the proposed DPG, as shown in Figure 2, we also investigated using loss functions to constrain DG to overcome its forgetting problem. We describe this approach here:

$$\min \sum_{j=0}^{J} ||\mathbf{z}_{sample,j} - DG_E(DG_D(\mathbf{z}_{sample,j}))|| \tag{10}$$

and

$$\min \sum_{j=0}^{J} ||DG_D(\mathbf{z}_{sample,j}, \theta_d) - DG_D(\mathbf{z}_{sample,j}, \theta'_d)|| \tag{11}$$

where we randomly sample $\mathbf{z}$ J times from multivariate normal distributions and $\mathbf{z}_{sample,j}$ is the j-th sample. As shown in Figure 2, part of training loss of DG is adapted from the original WAE. It ensures the consistency of input x_i and the reconstructed output x'_i, and minimizes the Wasserstein distance between the model distribution $\mathbb{E}(z_i)$ and the target distribution (we use multivariate standard normal distribution $N(\mu_0, \Sigma)$). Formula 10 constrains the consistency of DG's decoder and encoder over the randomly sampled $\mathbf{z}$. Formula 11 ensures that DG's decoder can still remember the old data. Using Formula 10 and 11, we can infer that the encoder's ability to process the old data is maintained.

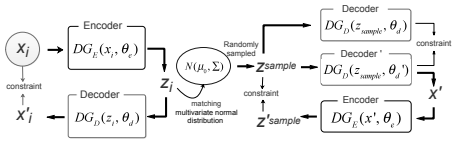


Figure 2: Data Generator training

[8] No need to constrain all units. Constraining only those in the last layer can already achieve good results.

Table 1: Experiment settings for different datasets

MNIST: We set the basic unit (hidden layer) size of the 3-layer classifier to 600 and the dropout rate to 0.3. A 3-layer convolutional network (CNN), with 2 * 2 convolutions and 64, 128, 256 filters for each layer, is used as DG's encoder. A deconvolutional network with the symmetric setting as the encoder is used as the decoder in DG. For our model, we using the parameters generated by DPG to replace 33% of the parameters in the solver's last hidden layer, and 10% of the first hidden layer.
CIFAR-10: We set the basic unit (hidden layer) size of the 3-layer classifier to 1000 and the dropout rate to 0.5. For our model, we use the parameters generated by DPG to replace 25% parameters of the solver's last hidden layer, and 10% of the first hidden layer. Since each image in this dataset contains a complex background thus is more difficult to classify, we add a 3-layer CNN below the 3-layer MLP classifier to improve the performance. 3-layer CNN has the same setting as the encoder of DG. Due to the complexity of the images in CIFAR-10, the replayed images are usually noisy. Text datasets usually cannot be exactly replayed. To alleviate this problem, we propose to replay the features (e.g. using an additional CNN to extract features) of input data which performs well in the experiment. In that case, the 3-layer CNN added to the classifier plays an important role in feature extraction and it needs to be fix; if not, the system won't get a stable input and thus will cause forgetting. Note that the feature extractor can be pre-trained using a large dataset (e.g. imagenet for image and wikipedia for text.) and thus can provide sufficient features.
Text Datasets: We set the basic unit (hidden layer) size of the 2-layer classifier to 1000 and dropout rate to 0.5. For our model, we use the parameters generated by DPG to replace 25% parameters of the solver's last hidden layer, and 10% of the first hidden layer. To get better results on text, we use pre-trained embeddings (Pennington et al., 2014; Li et al., 2018) for all experiments.

3 EXPERIMENTS

We now evaluate the proposed approach and compare it with state-of-the-art baselines using two image datasets and two text datasets.

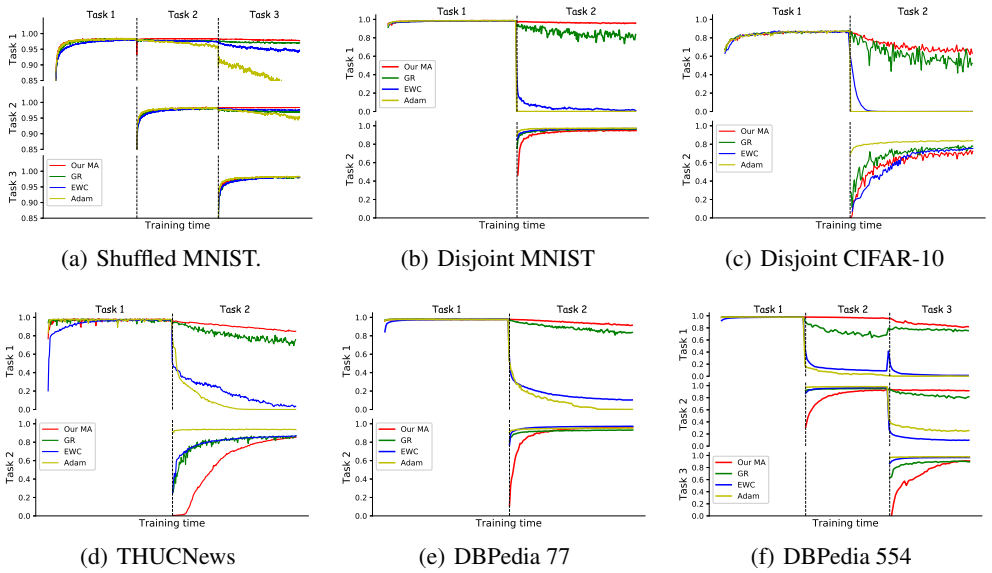
Experiment Settings

• **Two image datasets**: (1) MNIST: this dataset consists of 70,000 images of handwritten digits from 0 to 9. We use 60,000/3000/7000 images for training/validation/testing respectively. (2) CIFAR-10: this dataset consists of 60,000 32x32 color images of 10 classes, with 6000 images per class. There are 50,000/3000/7000 images for training/validation/testing respectively.
• **Two text datasets**: (1) DBPedia ontology: this is a crowd-sourced dataset (Lehmann et al., 2015) with 560,000 training samples and 70,000 test samples. Out of the 70,000 test samples, we use 10,000 for validation and 60,000 for test. (2) THUCNews: this dataset consists of 65,000 sentences of 10 classes (Li et al., 2006). We randomly select 50,000/5000/10,000 sentences for training/validation/testing respectively.

Data Preparation: To simulate sequential learning, we adopt the same two data processing methods in (Lee et al., 2017), named *disjoint* and *shuffled*.
(1) Disjoint: This method divides each dataset into several subsets according to the classes. Taking MNIST as an example, the MNIST dataset is divided into two subsets. The first subset consists of digits $\{0, 1, 2, 3, 4\}$ and the second subset consists of the remaining digits $\{5, 6, 7, 8, 9\}$. All systems learn the two subsets as two tasks in a sequential fashion and regard them together as 10-class classification. Other datasets are also processed in the same way. As DBPedia ontology has 14 classes, we randomly divide it into 3 groups. Each group contains 5 or 4 classes (14 classes in all). We call the three groups/tasks dataset DBPedia554 and the two groups/tasks dataset DBPedia77.
(2) Shuffled: Three datasets are constructed through data shuffling. The first dataset is the same as the original dataset. For the second and the third dataset, the input pixels of all images are shuffled with a fixed, random permutation. Since shuffling of words in a sentence is likely to change the sentence meaning and results in confusion, thus this experiment is not done on text datasets.

Baselines: To show the improvement of our approach, we use three state-of-the-art baselines that are representative of the current approaches: 1) **EWC** (elastic weight consolidation) (Kirkpatrick et al., 2017); 2) **IMM** (incremental moment matching) (Lee et al., 2017); 3) **GR** (generative replay) (Shin et al., 2017). We use the open source code released by the authors or the third party for comparison. We use Adam algorithm to update parameters and also use Adam as a baseline to show how serious the forgetting problem is.

Training Details: For fair comparison, our proposed approach uses the same solver (or classifier) as the baselines. That is, a multilayer perceptron is adopted as the solver/classifier (as the baselines all use this method), which is a 3-layer network (i.e., two basic units with each hidden layer as a unit) followed by a softmax layer. For our approach, the total number of parameters in the solver includes both the generated parameters $\mathbf{P}$ and the shared parameters θ_0. Due to the differences among different datasets, we adopt different settings for them, see Table 1 for details. Note that all baselines and our approach use the same setting for the same dataset. We use a 3-layer perceptron (with 2 hidden layers) network (we also call it *T-net*) for DPG and set the size of each hidden layer to 1000. Each T-net can generate 100 parameters at a time. We can parallel several T-nets as the DPG



(a) Shuffled MNIST. (b) Disjoint MNIST (c) Disjoint CIFAR-10

(d) THUCNews (e) DBPedia 77 (f) DBPedia 554

Figure 3: Accuracy curves - we test the system's accuracy per 40 training steps on the test set of each method. Y-axis shows the accuracy of different tasks as the training time (steps) increases. Note that Shuffled CIFAR-10 is not included as CNN cannot learn it directly.

to generate more parameters when needed. The network parameters are updated using the Adam algorithm with a learning rate of 0.001.

Results and Analysis

Figure 3 shows the test accuracy per 40 training steps of each method for each task as the tasks are sequentially learned (IMM is not used here, see below). Note that Shuffled CIFAR-10 is not included in the figure because our experiments show that it cannot be learned by CNN. It is also not used in any existing paper. From Figure 3, we can make the following observations:
(1). The proposed method consistently outperforms the baselines in overcoming forgetting, by a big margin in most cases.
(2). EWC does not perform well for the disjoint setting. GR is better but still poorer than our method. Adam's results show that forgetting is very serious for all datasets and settings.
(3). In general, our method converges slower than the baselines. But this is a small price to pay for the major gain in accuracy and in dealing with the catastrophic forgetting problem.

Note that the baseline IMM is not in Figure 3. That is because IMM works by combining well trained models of individual tasks to form one joint model for all tasks. Thus, we cannot draw an accuracy curve with increased training steps like others. We show the final results in Table 2.

From Table 2, we can see that our method is consistently superior to IMM on different datasets and settings. We believe that is because IMM suffers from the accuracy deterioration problem discussed in Section 1. Our method can alleviate this problem. Comparing with other baselines, we see the same pattern as in Figure 3. Our proposed MA method is clearly better.

Table 2: Average accuracy over all tasks in a sequence after the tasks have all been learned.

Model	shuffled MNIST	disjoint MNIST	disjoint CIFAR-10	THUCNews	DBPedia 77	DBPedia 554
Adam	91.46	48.64	41.99	46.78	47.66	41.10
EWC	96.70	48.96	37.75	45.02	53.95	35.89
GR	97.57	89.96	65.11	81.55	88.41	82.14
IMM	97.92	94.12	62.98	80.32	92.65	85.31
Our MA	**98.14**	**96.53**	**69.51**	**85.12**	**94.70**	**88.06**

Ablation Study

Here we study how the system behaves with less and less parameters in θ_0 or more and more parameters being replaced by parameters generated by DPG. We select the disjoint MNIST setting to conduct the experiment as it is more difficult and also more meaningful/useful than the shuffled setting.

Table 3: Average accuracy with different percentages of parameters in solver's last layer (one basic unit) being replaced with the parameters generated by DPG. Each score followed by a 95% confidence interval.

Model	GR	Our MA					
		0%	20%	40%	60%	80%	100%
Accuracy	88.71 ($\pm$2.64)	90.96 ($\pm$0.69)	92.44($\pm$1.31)	93.96($\pm$0.63)	94.51($\pm$0.87)	96.07($\pm$0.62)	96.05($\pm$0.64)

Table 3 shows the accuracy results (averaged in the same way as in Table 2). We can observe that the accuracy improves with increased percentages of parameters being replaced by the parameters generated by DPG. The best accuracy is obtained when 80% of the parameters are replaced through DPG, and the accuracy won't further improve with more replaced parameters. This observation indicates that replacing a part of parameters in solver to adapt new input tasks will be sufficient. We believe that is because part of parameters are shared among different tasks and thus there is no need to change them. In that case, further replacing those parameters won't gain much. The same conclusion can also be made by replacing the parameters in the first hidden layer of the solver (which has 2 hidden layers). We fix the replacing percentage of the last layer to 20%, and then increase the replacing percentages of the first layer. The best accuracy reaches 94.31%($\pm$0.84%) when replacing 40% parameters of the first layer, which gains 1.87% in accuracy compared with no replacement. This result also indicates that it suffices to replace the parameters in the last layer.

4 Related Work

Many approaches have been proposed to deal with catastrophic forgetting (CF). EWC (Kirkpatrick et al., 2017) quantifies the importance of weights to previous tasks, and selectively alters the learning rates of weights. Following EWC, Zenke et al. (2017) measured the synapse consolidation strength in an online fashion and used it as regularization. Learning without forgetting (LWF) (Li & Hoiem, 2016) feeds the old network with new training data in new tasks and regards the output as "pseudo-labels". In Incremental Moment Matching (IMM) (Lee et al., 2017), each trained network on one task is preserved and all networks are merged into one at the end of the sequence of tasks.

Above approaches basically focus on adjusting the network weights. Another main approach is to add some data of past tasks to the new task training to prevent forgetting the past. Gradient Episodic Memory (GEM) (Lopez-Paz et al., 2017) stores a subset of training data for every finished task, and limits the loss function on these so-called "memories". Instead of reserving some real data from previous tasks, Generative Replay (GR) (2017) followed the GANs framework to keep generators for previous tasks and learn parameters that fit a mixed set of real data of the new tasks and replayed data of previous tasks. Seff et al. (2017) also proposed to solve continual generative modeling by combining the ideas of GANs and EWC.

Some other approaches include Learn++ (Polikar et al., 2001), iCaRL (Rebuffi et al., 2017), Pathnet (Fernando et al., 2017), Memory Aware Synapses (Aljundi et al., 2017), One Big Net for Everything (Schmidhuber, 2018), Phantom Sampling (Venkatesan et al., 2017), Active Long Term Memory Networks (Furlanello et al., 2016), Conceptor-Aided Backprop (He & Jaeger, 2018), Gating Networks (Masse et al., 2018; Serrà et al., 2018), PackNet (Mallya & Lazebnik, 2017), Diffusion-based Neuromodulation (Velez & Clune, 2017), Dynamically Expandable Networks (DEN) (Yoon et al., 2018), Progress & Compress (Schwarz et al., 2018), and Incremental Regularized Least Squares (Camoriano et al., 2017).

Most of those works suffer from accuracy deterioration as discussed in Section 1 while some dynamically expanding the size of network (e.g., DEN and LWF), which results in a huge and complex model. As discussed in Section 1, our method is very different from these existing approaches.

5 Conclusion

This paper proposed a very different approach to dealing with catastrophic forgetting. The approach learns to build a model with two sets of parameters. The first set is shared by all tasks learned and the second set is dynamically generated to adapt the model to suit each individual test example. As we discussed in related work, this is different from all existing methods. Experimental results showed that the proposed approach outperformed the existing baseline methods markedly.

REFERENCES

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. *arXiv preprint arXiv:1711.09601*, 2017.

Raffaello Camoriano, Giulia Pasquale, Carlo Ciliberto, Lorenzo Natale, Lorenzo Rosasco, and Giorgio Metta. Incremental robot learning of new objects with fixed update time. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3207–3214. IEEE, 2017.

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.

Tommaso Furlanello, Jiaping Zhao, Andrew M Saxe, Laurent Itti, and Bosco S Tjan. Active long term memory networks. *arXiv preprint arXiv:1606.02355*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, pp. 3, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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*, pp. 201611835, 2017.

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pp. 4652–4662, 2017.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

Jingyang Li, Maosong Sun, and Xian Zhang. A comparison and semi-quantitative analysis of words and character-bigrams as features in chinese text categorization. In *ACL*, 2006.

Shen Li, Zhe Zhao, Renfen Hu, Wensi Li, Tao Liu, and Xiaoyong Du. Analogical reasoning on chinese morphological and semantic relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 138–143. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/P18-2023`.

Zhizhong Li and Derek Hoiem. Learning Without Forgetting. *ECCV*, 9908(1):614–629, 2016.

David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *arXiv preprint arXiv:1711.05769*, 1(2):3, 2017.

Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *arXiv preprint arXiv:1802.01569*, 2018.

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

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proc. CVPR*, 2017.

Juergen Schmidhuber. One big net for everything. *arXiv preprint arXiv:1802.08864*, 2018.

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

Ari Seff, Alex Beatson, Daniel Suo, and Han Liu. Continual learning in generative adversarial nets. *arXiv preprint arXiv:1705.08395*, 2017.

Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Roby Velez and Jeff Clune. Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks. *PloS one*, 12(11):e0187736, 2017.

Ragav Venkatesan, Hemanth Venkateswara, Sethuraman Panchanathan, and Baoxin Li. A strategy for an uncompromising incremental learner. *arXiv preprint arXiv:1705.00744*, 2017.

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 189–198, 2017.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*, 2017.