A Class Incremental Learning Algorithm for a Compact-Sized Probabilistic Neural Network and Its Empirical Comparison with Multilayered Perceptron Neural Networks

Shunpei Morita, Hiroto Iguchi, and Tetsuya Hoya(✉)

Department Computer Engineering, CST, Nihon University, 7-24-1, Narashino-dai, Funabashi-City, Chiba 274-8501, Japan
houya.tetsuya@nihon-u.ac.jp

Abstract. It is well known that class incremental learning using deep learning is difficult to achieve, since deep learning approaches inherently suffer from the catastrophic forgetting in the training mode. In contrast, a probabilistic neural network is capable of performing classification tasks based upon a set of local spaces, each composed of a training pattern, and thereby class incremental learning can be robustly performed. In this paper, we propose a class incremental learning method by exploiting the property of a probabilistic neural network, while reducing effectively the number of the training patterns to be stored within the hidden layer, and compare the performance of the class incremental learning tasks obtained using a multilayered perceptron model with that using a probabilistic neural network. Simulation results using seven publicly available datasets show that both the classification accuracies of an original probabilistic neural network and the proposed incremental learning method are 2.59 to 26.58 times higher than that of the deep learning in class incremental learning. Moreover, we observed that the class incremental learning performed using a probabilistic neural network exhibited a robust performance compared to the deep neural networks with iCaRL. In addition, it was observed that the proposed learning method was able to reduce effectively the number of the units in the hidden layer, while with the decrease in accuracy by only 1.77% to 7.06%, compared to the original one.

Keywords: Probabilistic neural network · Multilayered perceptron · Deep learning · Class incremental learning · Pattern classification

1 Introduction

Deep learning (DL) [1] is one of the most widely used learning methods in the field of machine learning, and it has made significant contributions to the development of pattern recognition technologies. In particular, deep neural networks

H. Lu et al. (Eds.): ACPR 2023, LNCS 14406, pp. 288–301, 2023.
https://doi.org/10.1007/978-3-031-47634-1_22

(DNNs) [2] are nowadays prevalent in both academic studies and industrial applications, and many image classifier models, including VGG [3], ResNet [4] and AlexNet [5], for instance, have been used across a wide range of fields. In many of the cases, the performance of a classification model is typically evaluated after the training performed in an iterative fashion, using the entire set of the training patterns. In practice, however, it is often the case that the training patterns are not ready to be available for all the classes but given only partially at a time and those of the remaining classes come later. Therefore, it is desirable to perform a continual learning (CL) [6], where the network training continues with the incoming new patterns. Within the CL principle, a number of class incremental learning (CIL) [7] approaches have been proposed to date [8–10]. In CIL, a pattern classifier is augmented with new classes, using only an additional set of the patterns for the new classes. However, it has been increasingly acknowledged, as pointed out in [11–13], that catastrophic forgetting of previously learned classes is a major problem in DL for performing the CIL/CL.

According to the taxonomy proposed in [14], the CIL approaches for DL proposed to date can be classified into the three categories of i) data-centric, ii) model-centric, and iii) algorithm-centric ones. For i), the CIL method in [8] introduces maximum entropy regularization (MER) into the loss function in order to prevent overfitting to the uncertain knowledge, and a confident fitting is given as penalty. In contrast, the method in [9] utilizes the dynamically expandable representation (DER) and can be categorized as a type ii) approach, based upon the classification in [14]. Within the method, the representation previously learned by a DNN is frozen and new additional feature extractors are learned for the additional tasks. For iii) the method in [10], i.e. weight aligning (WA), maintains the fairness between the previously learned and new classes. One of the other model-centric CIL methods is elastic weight consolidation (EWC) [15], and it is claimed in [15] that the old tasks can be memorized by slowing down the learning pace of some selected network parameters. However, the work in [14] reports rather counterfactual simulation results. In addition, replay methods have received much attention in the CIL for DNNs, and iCaRL [11] is particularly widely used among them. The iCaRL attempts to get over the catastrophic forgetting by adding some previously trained data to the newly incoming data. Although several methods have proposed to date, it is said that the catastrophic forgetting inherent to DNNs has still not been fully overcome.

In contrast to the DNN approaches described above, probabilistic neural network (PNN) [16] is another artificial neural network model, originally proposed for pattern classification problems. The original PNN model has only a single hyperparameter to be given a priori, and its training is straightforwardly completed in a one-shot manner; each training pattern is accommodated as the attribute vector of a second layer unit. Moreover, it is also shown in a recent study [17] that the parallel implementation of a PNN using k-means clustering enables its reference (testing) mode to be performed superior to a DNN, by exploiting the independence property of the arithmetic operations for each unit in the hidden layer, as well as those in each output layer unit representing a single class.

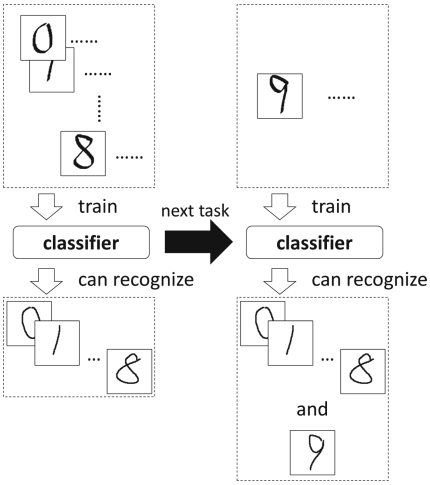

Fig. 1. The flow of the class incremental learning.

In this paper, we propose a novel class incremental learning algorithm for a PNN with a reduced number of the hidden layer units (a.k.a. compact sized PNN; CS-PNN) and empirically verify its effectiveness by comparing a DNN and PNN with the original training scheme. The organization of the paper is as follows: Sect. 2 describes what CIL is, followed by a brief introduction of the PNN, and comparison between the CIL on a PNN and DNN. Section 3 proposes a new CIL algorithm for PNN. Section 4 is devoted to the simulation study, i) comparing a DNN and the original PNN under the CIL situations, each yielding the baseline, and then ii) evaluating the classification performance of the CS-PNN. Section 4 provides the summary of the present work and suggests some future directions.

2 Preliminary Studies

2.1 Class Incremental Learning

In [11], CIL is meant to be the task of learning incrementally some additional classes for an already learned model, so that the newly added classes can be also identified by the model, besides the classes already learned. For instance, provided that a model has been trained for the classification task of handwritten digits and is capable of performing the task only partially, e.g. the digits from zero to eight, an additional learning is performed on the model, so that a new digit nine can also be classified, in addition to the nine digits already learned, as illustrated in Fig. 1.

2.2 Probabilistic Neural Network

A PNN is a feed-forward neural network composed of an input and output layer, with linear-sum activation units for the latter, and a single hidden layer with the

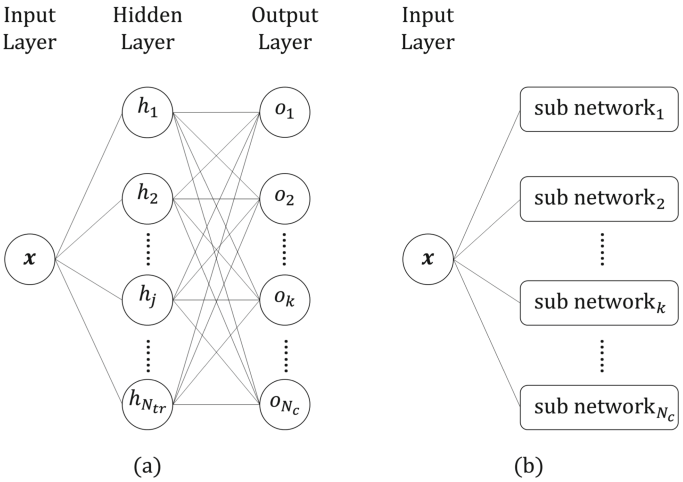

Fig. 2. (a) The structure of a PNN with a total number of training patterns and N_c classes, and (b) The PNN represented as a composite structure by a set of the N_c class-independent sub-networks.

nonlinear units. Figure 2 (a) illustrates the structure of a PNN. In Fig. 2, each of the hidden layer units has a radial basis function (RBF) in terms of Gaussian response function as an activation function:

$$h_j(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c_j}\|_2^2}{\sigma^2}\right) \tag{1}$$

where $\|\cdots\|_2$ denotes $L2$-norm, h_j, $\boldsymbol{x}$, and $\boldsymbol{c_j}$ are respectively the output value of the j-th hidden layer unit, the input vector, and the attribute vector representing one of the training patterns, and where σ is the radius unique to all the RBFs. Each hidden layer unit is connected to an output layer unit corresponding to the class to which the training pattern belongs. In PNN, the weight between a hidden layer and output layer units is set as follows:

$$w_{j,k} = \begin{cases} 1 & \text{if } \boldsymbol{c_j} \text{ belongs to class } k, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In the output layer, each unit yields the activation value:

$$o_k = \frac{1}{U_k} \sum_{j=1}^{U_k} w_{k(j),k} h_{k(j)}(\boldsymbol{x}) \tag{3}$$

$$N_{tr} = \sum_{k=1}^{N_c} U_k \tag{4}$$

where $w_{k(j),k}$ is the weight between the j-th RBF belonging to class k and the k-th output layer unit, U_k is the number of the training vectors for class k, N_{tr} is

the total number of the training vectors for all the classes, and $k(j)$ is the ID of the RBFs that belong to class k (which is the same as o_k). In a PNN, since both the hidden and output layer units are completely separated from class to class, the network can be eventually regarded as a composite network consisting of its class-independent subnetworks, as shown in Fig. 2 (b). Moreover, the training of the original PNN is completed in only 2 steps: i) storing each training vector as the attribute of a hidden layer unit and ii) determining σ in (1) appropriately.

2.3 Class Incremental Learning on DNN and PNN

For training a DNN, DL aims to minimize the output error for $\boldsymbol{w}$, i.e. a certain set of the network parameters, where the input vector $\boldsymbol{x}$ is given:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial E(\boldsymbol{x}, \boldsymbol{w})}{\partial \boldsymbol{w}} \tag{5}$$

where E is the error function, for which a categorical cross entropy is often used in classification problems, and η is the learning rate. Then, in the situation where the network parameters are optimized for a new set of the training patterns, it is quite often the case that the knowledge acquired from new training patterns can easily override the weight and bias parameters optimized using previous patterns. This is the main cause of the catastrophic forgetting occurring in a DNN. Therefore, in the situation where the CIL is performed on a DNN, the problem arises in the pattern classification tasks, as the classification accuracy for a new class is apt to decline dramatically. In other words, this is due to the fact that the model can be optimized *only* for a particular set of the training patterns for a new class.

In contrast to the DNNs, PNN does not suffer from such forgetting, as the network training on a PNN essentially proceeds by only adding each hidden layer unit with storing a training vector as an attribute vector within the RBF, as described earlier. Moreover, in the situation where the dataset given is well-balanced, it is considered that the feature space can be reasonably covered by a set of the local subspaces spanned by the respective attribute vectors, each stored in a hidden layer unit, and thereby it is considered that an additional learning such as the CIL can be effectively and straightforwardly carried out.

3 Class Incremental Learning Algorithm for a Compact-Sized Probabilistic Neural Network

Here, we propose a new method applied for training the PNN, i.e. a compact-sized probabilistic neural network (CS-PNN) for CIL, with reducing the number of units in the hidden layer. In the CS-PNN, each unit in the hidden layer memorizes the location of a selected training vector in the feature space. Also, the number of the hidden layer units can be suppressed by updating the location of some existing vector, if a new training vector is located nearby the existing one, without adding it as a new attribute vector. The training algorithm for the CS-PNN is summarized as follows.

step 1. For the first training pattern belonging to a new class (k=1,2,...,N_c), add $h_{k(j=1)}$ into the hidden layer with $\boldsymbol{c_{k(i=1)}} = \boldsymbol{x_{tr(k,i=1)}}$, and o_k into the output layer.

step 2. For $i = 2, 3, \ldots, U_k$, perform the following for all the remaining training patterns:

If $\exists h_k; h_k(\boldsymbol{x_{tr(k,i)}}) > \theta_k$ then update $\boldsymbol{c_j}$ for the unit h_j which yields a maximum output value among the existing units belonging to class k to the average between $\boldsymbol{x_{tr(k,i)}}$ and $\boldsymbol{c_j}$ as follows:

$$\boldsymbol{c_j} \leftarrow \frac{\boldsymbol{x_{tr(k,i)}} + \boldsymbol{c_j}}{2} \tag{6}$$

Otherwise, add $h_{k(i)}$ into the hidden layer with $\boldsymbol{c_{k(i)}} = \boldsymbol{x_{tr(k,i)}}$, as well as o_k to the output layer, if it does not exist yet, and connect $h_{k(i)}$ to o_k (i.e. with the weight unity in between);

Note that applying the CS-PNN algorithm in the above will automatically generate the units in both the hidden and output layers, where necessary. Also, since the parameters $\boldsymbol{c_j}(j = 1, 2, \ldots, U_k$; U_k is the number of training patterns belonging to class k) are updated only for the subnet representing a single class, the CIL is performed without affecting to the attribute vectors stored in other subnets (i.e. representing the respective classes). Moreover, during the operation of the CIL algorithm in the above, only the attribute vector of the unit which is the nearest to the training is updated, while the stored vectors in other units remain intact, with only a small and necessary change in the feature space spanned. In addition, since only the attribute vectors of the units with their activations larger than the threshold are updated, a smaller value of θ can lead to a further reduction in the number of the hidden layer units, depending upon the situations. Note, however, that there needs a trade-off between the performance, since, with a small setting of θ, the chance that the attribute located far from the primary feature space for a particular class can be selected for the update increases. For this reason, a care should be taken for the choice of the value θ, besides the unique radius σ in (1).

4 Simulation Study

We conducted a series of the simulations, aimed for comparing the CIL capabilities of the DNN, original PNN, and the proposed CS-PNN.

In the simulation study, we used the seven publicly available datasets: abalone, isolet, letter-recognition, MNIST, optdigits, pendigits, and wdbc; one obtained from the MNIST [18], and the remaining six from the UCI machine learning repository [19]. Then, each pattern vector of a dataset was normalized within the range of $[-1, 1]$:

$$x_i \leftarrow 2\left(\frac{x_i - x_{i\mathrm{MIN}}}{x_{i\mathrm{MAX}} - x_{i\mathrm{MIN}}} - 0.5\right) \tag{7}$$

Table 1. Summary of the seven datasets used for the simulation study.

Dataset	#Training	#Testing	#Classes	#Features per pattern
abalone	2088	2089	3	7
isolet	2252	1559	26	617
letter-recognition	16000	4000	26	16
MNIST	60000	10000	10	784
optdigits	3823	1797	10	64
pendigits	7494	3498	10	16
wdbc	398	171	2	30

where i is the ID of an attribute vector. The properties of the seven datasets used for the simulations are summarized in Table 1. For the unique hyper-parameter of a PNN, σ was set by the following equation, and the setting was used through all the simulations, which yielded relatively a reasonable performance for each dataset:

$$\sigma = \frac{d_{\mathrm{MAX}}}{N_c} \tag{8}$$

where d_{MAX} is the maximum distance computed using all the pairs of the training patterns across all the classes used for training, and N_c is the number of classes.

In the case of training a DNN, an entire set of the training vectors was divided into those of the respective class, and each set of the vectors was presented to the network as one batch at a time for the training. For the DNN, once an iterative training session using the batch belonging to a certain class was completed, the batch for other class was used for the next iterative training session. This manner of the subsequent training sessions continued till the last class.

4.1 Comparison Between the DNN and Original PNN Under the CIL Situation

In this simulation, we first compared the DNN and PNN for the baseline tasks of the CIL. For the baseline tasks, each batch for the class till the last was subsequently presented to a DNN/PNN and the CIL at each presentation was performed. Then, the classification accuracy after completing the CIL was evaluated for each of the two models, using the testing vectors. In the tasks, therefore, each training pattern vector was stored, as described earlier, as an attribute vector of a unit in the hidden layer of a PNN, while the parameter setting, as well as the training, of the DNN was done based upon the following manner:

- Number of the Hidden Layers: varied from 1 to 3.
- Training Algorithm: Adam [20] (learning rate = 0.01, $\alpha = 0.9$, $\beta = 0.999$).
- Number of Epochs in an Iterative Training Session:
  - Patterns for the first class (i.e. the first CIL task): 20 epochs.

Table 2. Simulation results comparing the classification accuracies, obtained after the completion of the training for all the classes in the CIL situation, and showing the baseline performance of DNN and PNN, under the CIL situation.

Dataset	Acc. of DNN [%]						Acc. of PNN[%]
	1 hidden layer		2 hidden layers		3 hidden layers		
	1 epoch	20 epochs	1 epoch	20 epochs	1 epoch	20 epochs	
abalone	36.05	36.05	36.05	36.05	31.02	36.05	**57.25**
isolet	11.61	7.7	3.85	3.85	3.85	3.85	**86.34**
letter-recognition	3.62	3.95	3.95	3.95	3.95	3.95	**96.23**
MNIST	10.09	10.09	10.09	10.09	10.09	10.09	**96.50**
optdigits	10.52	10.02	10.91	10.02	10.02	10.02	**98.39**
pendigits	18.7	17.64	15.04	9.61	9.61	9.61	**94.25**
wdbc	37.43	37.43	37.43	37.43	37.43	37.43	**97.08**

- Patterns for other classes (i.e. other remaining tasks): once epoch/20 epochs.

In the setting for the DNN above, the three hyper-parameters for the Adam, which is considered as one of the state-of-the-art DL algorithms, were those given as the default values used in PyTorch [21] except for the learning rate, whereas the numbers of the epochs chosen were based upon the preliminary simulations; it was empirically confirmed that 20 epochs were sufficient to reach a convergent state for each run. The simulation results are summarized in Table 2. In this table, we obviously see that a DNN is not able to perform properly the CIL tasks at all, where the number of classes was increased one-by-one. On the other hand, the classification accuracies obtained using a PNN were always higher than those obtained using the DNNs for all the seven datasets used in the simulation study. As shown in Table 2, it was observed that the classification accuracy by a PNN was around 2.5 times as high as those by DNNs for the wdbc case, whereas 26.5 times higher for the letter-recognition case. For the CIL tasks using the MNIST, we then analyzed the classification accuracies in each class obtained by a DNN with three hidden layers, upon the iterative training of 20 epochs in each additional training task. In the analysis, we confirmed that the accuracies for the preciously learned classes were all dropped to zero, except that of 100% for each new class just learned. Therefore, it is said that the catastrophic forgetting did occur in the early stage of learning in each case of the DNNs, since there was no significant difference between the classification accuracies for the one-epoch cases and those of the twenty epochs, as shown in Table 2. From these observations, it can be therefore concluded that a PNN is capable of performing the CIL, without any serious forgetting occurred, as reported in [22], while the DNNs are not.

Table 3. Simulation results comparing the classification accuracy and the number of the RBFs using the original PNN and CS-PNN for each of the seven datasets.

Dataset		Original PNN	CS-PNN			
			$\theta = 0.6$	$\theta = 0.7$	$\theta = 0.8$	$\theta = 0.9$
abalone	Acc. [%]	**57.25**	36.96	44.23	44.71	44.47
	Num. RBFs	2088	**11**	14	23	37
isolet	Acc. [%]	**86.34**	83.64	85.12	86.14	86.14
	Num. RBFs	2252	**2126**	2161	2187	2240
letter-recognition	Acc. [%]	**96.23**	90.68	92.40	94.20	95.60
	Num. RBFs	16000	**9703**	10667	11635	12809
MNIST	Acc. [%]	**96.50**	91.14	95.38	96.12	96.48
	Num. RBFs	60000	**47240**	47476	47778	50177
optdigts	Acc. [%]	98.39	98.11	98.22	**98.44**	98.33
	Num. RBFs	3823	**2819**	2986	3208	3471
pendigits	Acc. [%]	94.25	93.62	94.23	93.42	**95.57**
	Num. RBFs	7494	**1873**	2404	3200	4487
wdbc	Acc. [%]	**97.08**	82.46	82.46	83.63	**97.08**
	Num. RBFs	398	**9**	19	26	65

4.2 Evaluation of Classification Accuracy and Reduction Rate of Hidden Layer Units for the Proposed CS-PNN

We then conducted another set of simulations, in order to validate the classification performance in the case where the CS-PNN was applied. The simulation results are shown in Tables 3 and 4 and Figs. 3 and 4.

Table 3 summarizes both the classification accuracies and number of hidden layer units after performing the incrementally training of a PNN by applying the proposed algorithm (i.e. CS-PNN) in Sect. 3.

In Table 4, a performance comparison of both the relative difference in terms of classification accuracy and reduction rate of RBFs between the original PNN and CS-PNN, calculated using the values in Table 3, is shown. The relative difference and reduction rate were calculated, respectively, as follows:

$$\text{Relative Difference} = \text{Acc. of original PNN} - \text{Acc. of CS-PNN} \tag{9}$$

$$\text{Reduction Rate} = 1 - \frac{\text{Num. RBFs in each CS-PNN}}{\text{Num. RBFs in original PNN}} \tag{10}$$

Here, the relative difference in (9) is introduced for a straightforward comparison of the difference between the classification accuracy of the original PNN and CS-PNN; a minus value shows the accuracy of CS-PNN inferior to that of the original PNN. As shown in the bottom two rows in Table 4, the average

Table 4. Comparison of relative difference in classification accuracy between the original PNN and CS-PNN, and reduction rate of hidden layer units (RBFs).

Dataset		CS-PNN			
		$\theta = 0.6$	$\theta = 0.7$	$\theta = 0.8$	$\theta = 0.9$
abalone	Relative Difference [%]	−20.30	−13.02	**−12.54**	−12.78
	Reduction rate of RBFs [%]	**99.47**	99.33	98.90	98.23
isolet	Relative Difference [%]	−2.69	−1.22	**−0.19**	**−0.19**
	Reduction rate of RBFs [%]	**5.60**	4.04	2.89	0.53
letter-recognition	Relative Difference [%]	−5.55	−3.82	−2.03	**−0.63**
	Reduction rate of RBFs [%]	**39.36**	33.33	27.28	19.94
MNIST	Relative Difference [%]	−5.36	−1.12	−0.38	**−0.02**
	Reduction rate of RBFs [%]	**21.27**	20.87	20.37	16.37
optdigts	Relative Difference [%]	−0.28	−0.17	**0.06**	−0.06
	Reduction rate of RBFs [%]	**26.26**	21.89	16.09	9.21
pendigits	Relative Difference [%]	−0.63	−0.03	−0.83	**1.32**
	Reduction rate of RBFs [%]	**75.01**	67.92	57.30	40.13
wdbc	Relative Difference [%]	−14.62	−14.62	−13.45	**0.00**
	Reduction rate of RBFs [%]	**97.74**	95.23	93.74	83.67
Avg.	Relative Difference [%]	−7.06	−4.86	−4.19	**−1.77**
	Reduction rate of RBFs [%]	**52.10**	48.95	45.18	38.30
Med	Relative Difference [%]	−5.36	−1.22	−0.83	**−0.06**
	Reduction rate of RBFs [%]	**39.36**	33.33	27.28	19.94

over the relative difference for each value of θ for the CS-PNN was −7.06%, −4.86%, −4.19%, and −1.77%, respectively, whereas the corresponding median, which was computed over these results, was respectively as −5.63%, −1.22%, −0.83%, and −0.06%. Therefore, overall, a dramatic performance degradation as in the DNNs was not observed for the CS-PNN, compared to the original PNN. In Table 4, it is, however, observed that the range of the decrease in terms of the classification accuracy for both the abalone and wdbc datasets is relatively larger than that for other datasets. The possible reason for such a deterioration is ascribed to the relatively higher reduction rate of the hidden layer units, as shown in Table 4, and/or the removal of the units yielding a significant impact on the classification performance. In contrast, the accuracies obtained using the CS-PNN for the other five datasets, i.e. isolet, letter-recognition, MNIST, optdigits, and pendigits, are all shown to remain almost intact, each with a relatively high reduction rate of the RBFs (except the isolet case with $\theta = 0.6$), as shown in Table 4.

On the other hand, the reduction rates for both the abalone and wdbc (except for θ=0.09) were over 90%, while a significant decrease in terms of the classification accuracy of over 10% was also observed. In contrast, the relative difference in

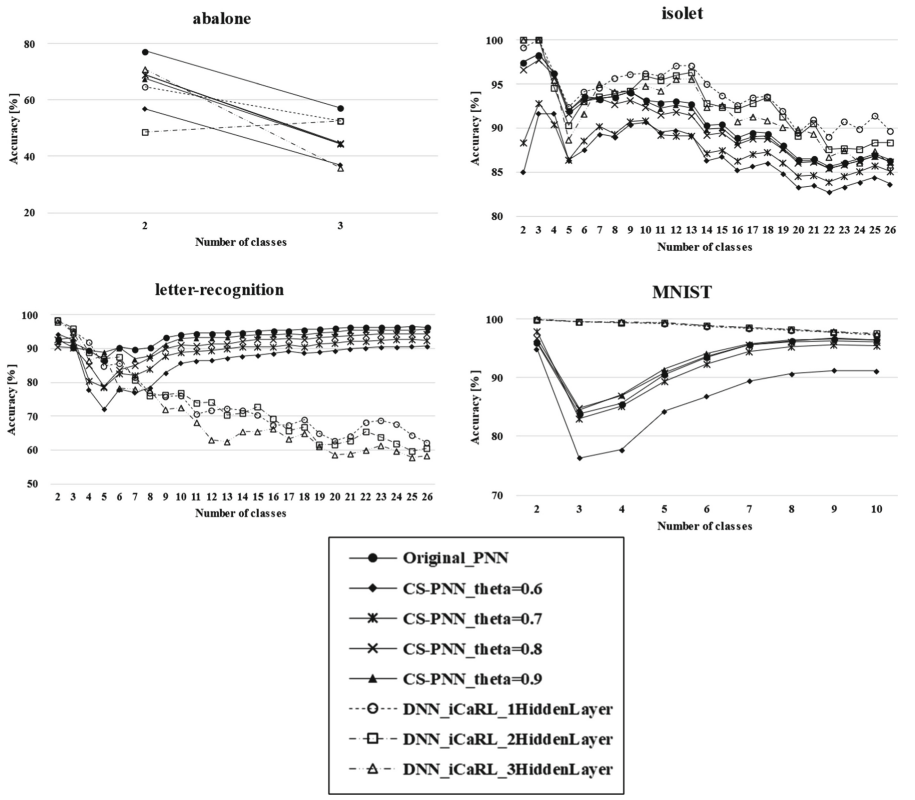

Fig. 3. Classification accuracies obtained using original PNN, CS-PNN (θ=0.6,0.7,0.8,0.9), and iCaRL on DNN ($K = 0.2 \times$ the number of all training data), with the number of classes varied for the abalone, isolet, letter-recognition, and MNIST datasets.

classification accuracy for the five cases of the isolet, letter-recognition, MNIST, optdigits, and pendigits almost always stayed below the corresponding averaged one, while the reduction rates of the RBFs were varied greatly with the setting of θ as shown in the bottom in Table 4. This indicates that an appropriate setting of θ by somehow taking into account the overall distribution of the distances between the input and attribute vectors is necessary, so as to effectively reduce the number of the RBFs. Therefore, it is considered, as a rule of thumb in practice, that the value of the unique radius σ is first tuned to yield a higher classification accuracy, then θ is varied for an effective reduction in the number of the RBFs.

In sum, from these observations, it is said that the CS-PNN can effectively select the training pattern vectors to be accommodated within the hidden units, while maintaining relatively well-separated class boundaries in between, as compared to the original PNN approach.

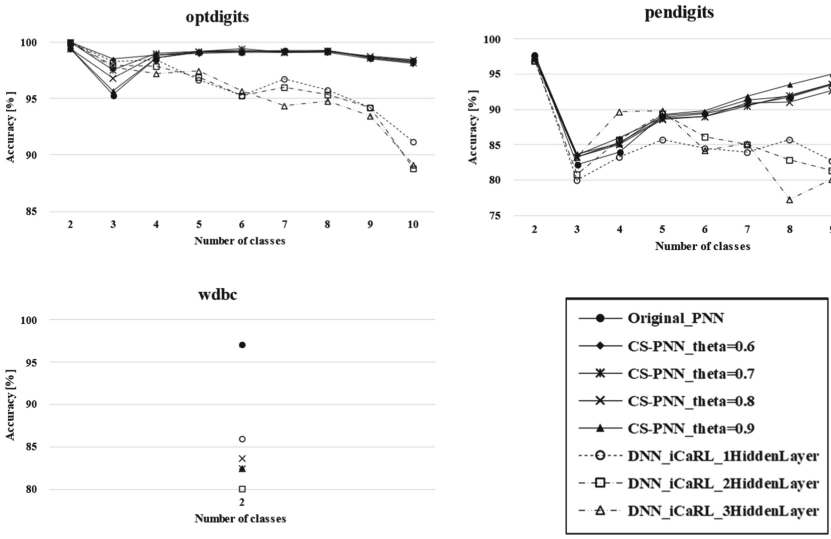

Fig. 4. Classification accuracies obtained using original PNN, CS-PNN (θ=0.6,0.7,0.8,0.9), and iCaRL on DNN ($K = 0.2 \times$ the number of all training data), with the number of classes varied for the optdigits, pendigits, and wdbc datasets.

Figures 3 and 4 show the changes in the classification accuracy of the CIL with an original PNN, CS-PNN, and the DNN with iCaRL. In this simulation, the memory size K for the iCaRL, i.e., the number of stored data for the subsequent additional training task, was set at 0.2 times the number of all training data in each dataset. In Figs. 3 and 4, we observe that for all the datasets except the abalone the classification accuracy almost always stays above 70–80%. Moreover, as shown in Figs. 3 and 4, the accuracy for all the four datasets but the abalone, isolet, and wdbc was improved steadily, albeit sometimes exhibiting a sudden drop at an earlier CIL task, and remained relatively high afterwards, unlike the DNN without applying the iCaRL. In addition, it was confirmed that the accuracy of the iCaRL consistently showed a decrease with each additional training for the letter-recognition, optdigits, and pendigits cases, compared to the CS-PNN. In the isolet case, however, the accuracy of CS-PNN was lower than iCaRL at the early stages, though the classification accuracy of the CS-PNN approached that of the iCaRL as the additional training tasks proceeded. A similar trend was observed for the MNIST case. In contrast, for the abalone and wdbc, the CS-PNN and iCaRL exhibited no significant difference in the accuracy. Therefore, it is said that the CS-PNN performed effectively for all the seven datasets used for the simulation study compared to the CIL of the DNN with iCaRL.

5 Conclusion

In this work, we have firstly shown that a PNN is capable of performing the CIL, through the simulation study using seven publicly available datasets in comparison with the DNNs. We have then proposed the CS-PNN, for the purpose of effectively reducing the number of the hidden layer units in a PNN, while maintaining a reasonably high classification performance. It has also been observed that performing a CIL is virtually not possible by a bare DNN approach using the Adam algorithm for all the cases, due to the catastrophic forgetting occurred during the simulation, while the CS-PNN can cope moderately well with the CIL. Compared to the iCaRL, CS-PNN can also perform robustly in the CIL tasks. Moreover, it is worth mentioning that,unlike DNNs, the training of a PNN is fast, as the training does not require iterative training of the network parameters at all but can be simply done by assigning some selected training data to the hidden layer unit's attributed vectors as described in this work. It is also notable that, beside the hidden layer units, the network obtained via the CS-PNN has a varying number of output units during a CIL task, unlike conventional, fixed-sized DNN models. In addition, it is also reported in [23] that a PNN exhibits the high robustness against an adversarial attack.

Future work is directed to the investigation of the effective choice of the unique radius σ of a PNN, as well as that applicable to the CS-PNN.

References

1. Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
2. Schmidhuber, D.E., Hinton, G.E., Williams, R.J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
3. Karen, S., Andrew, Z.: Very deep convolutional networks for large-scale image recognition. The 3rd International Conference on Learning Representations, https://arxiv.org/pdf/1409.1556. Accessed 29 May 2023
4. Kaiming, H., Xiangyu, Z., Shaoqing, R. Jian, S.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
5. Alex, K., Ilya, S., Geoffrey, E, H.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25 (2012)
6. Sebastian, T., Tom, M.M.: Lifelong robot learning. Robot. Auton. Syst. **15**(1–2), 25–46 (1995)
7. McClelland, J.L., McNaughton, B.L., O'Reilly, R.C.: Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. Psychol. Rev. **102**(3), 419–457 (1995)
8. Dahyun, K., Jhwan, B., Yeonsik, J., Jonghyun C.: Incremental learning with maximum entropy regularization: rethinking forgetting and intransigence. https://arxiv.org/abs/1902.00829. Accessed 29 May 2023
9. Shipeng, Y., Jiagwei, X., Xuming, H.: Der: dynamically expandable representation for class incremental learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3013–3022 (2021)

10. Bowen, Z., Xi, X., Guojun, G., Bin, Z., Shu-Tao, X.: Maintaining discrimination and fairness in class incremental learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 13205–13224 (2020)
11. Sylvestre-Alivise, R., Alexander, K., Georg, S., Christoph, H. L.: iCaRL: incremental classifier and representation learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2001–2010 (2017)
12. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: the sequential learning problem. Psychol. Learn. Motiv. **24**, 109–165 (1989)
13. Ratcliff, R.: Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychol. Rev. **97**(2), 285–308 (1990)
14. Da-Wei, Z., Qi-Wei, W., Zhi-Hong, Q., Han-Jia, Y., De-Chuan, Z., Ziwei, L.: Deep class-incremental learning: a survey. https://arxiv.org/pdf/2302.03648.pdf. Accessed 29 May 2023
15. James, K., et al.: Overcoming catastrophic forgetting in neural networks. PANS **114**(13), 3521–3526 (2017)
16. Specht, D.F.: Probabilistic neural networks. Neural Netw. **3**(1), 109–118 (1990)
17. Takahashi, K., Morita, S., Hoya, T.: An analytical comparison between the pattern classifiers based upon a multilayered perceptron and probabilistic neural network in parallel implementation. Int. Conf. Art. Neural Netw. **3**, 544–555 (2022)
18. LeCun, Y., Cortes, C., Burges, C. J. C.: The MNIST database. http://yann.lecun.com/exdb/mnist/. Accessed 19 Aug 2021
19. Dua, D., Graff, C.: UCI machine learning repository. Univ. California Irvine, Irvine, CA. http://archive.ics.uci.edu/ml. Accessed 29 May 2023
20. Diederik, P. K., Jimmy, B.: Adam: a method for stochastic optimization. https://arxiv.org/abs/1412.6980. Accessed 29 May 2023
21. Pytorch, Team.: PyTorch: An imperative style, high-performance deep learning library. https://pytorch.org. Accessed 01 June 2023
22. Hoya, T.: On the capability of accommodating new classes within probabilistic neural networks. IEEE Trans. Neural Netw. **14**(2), 450–453 (2003)
23. Ian J. G., Jonathon, S., Christian, S.: Explaining and harnessing adversarial examples. International Conference on Learning Representations. https://arxiv.org/abs/1412.6572. Accessed 29 May 2023