Modular Deep Belief Networks that do not Forget

Leo Pape, Faustino Gomez, Mark Ring and Jürgen Schmidhuber

Abstract— **Deep belief networks (DBNs) are popular for learning compact representations of high-dimensional data. However, most approaches so far rely on having a single, complete training set. If the distribution of relevant features changes during subsequent training stages, the features learned in earlier stages are gradually forgotten. Often it is desirable for learning algorithms to retain what they have previously learned, even if the input distribution temporarily changes. This paper introduces the M-DBN, an unsupervised modular DBN that addresses the forgetting problem. M-DBNs are composed of a number of modules that are trained only on samples they best reconstruct. While modularization by itself does not prevent forgetting, the M-DBN additionally uses a learning method that adjusts each module's learning rate proportionally to the fraction of best reconstructed samples. On the MNIST handwritten digit dataset module specialization largely corresponds to the digits discerned by humans. Furthermore, in several learning tasks with changing MNIST digits, M-DBNs retain learned features even after those features are removed from the training data, while monolithic DBNs of comparable size forget feature mappings learned before.**

I. INTRODUCTION

DEEP BELIEF NETWORKS (DBNs; [1]) are popular for learning compact representations of high-dimensional data. DBNs are neural networks consisting of a stack of Boltzmann machine layers that are trained one at a time, in an unsupervised fashion to induce increasingly abstract representations of the inputs in subsequent layers. This layer-by-layer training procedure facilitates supervised training of deep networks, which are in principle more efficient at learning compact representations of high-dimensional data than shallow architectures [2], but are also notoriously difficult to train with traditional gradient methods (e.g., [3]).

DBNs can be particularly useful as sensory preprocessors for learning agents that interact with an environment that requires learning complex action mappings from high-dimensional inputs. Often these input spaces are embedded within a vast state space where the input distribution may vary widely between regions. Rather than assemble a single, monolithic training set covering all eventualities, it is more efficient to train an agent incrementally such that it can build upon what it learned previously. This *continual learning* paradigm [4] demands that the underlying learning algorithms support the retention of earlier training. While DBNs (and the related approach of stacked autoencoders) have been successfully applied to many tasks [1, 2, 5–7], most approaches rely on training data that are sampled from a stationary distribution. However, in continual learning, where the statistics of the training data change over time, DBNs, like most connectionist approaches, gradually forget previously learned representations as new input patterns overwrite those patterns that become less probable.

A possible remedy to forgetting is to split a monolithic network into a number of expert modules, each of which specializes on a subset of the task. In such an ensemble approach, expert modules are trained to improve their performance only on those subtasks they are already good at, ignore the subtasks of other experts, and thereby protect their own weights from corruption by unrelated input patterns. Jacobs et al. [8] introduced a supervised method for training local experts in which a gating network is trained to assign each input pattern to the expert that produced the lowest output error. An unsupervised version of this algorithm described in [9] uses the reconstruction error of each module on a sample to train the modules with the best reconstruction to become even better, and, optionally, to train the other modules to become worse in reconstructing that sample. While these methods divide a task over multiple modules, they contain no mechanism for preventing an expert module from shifting its expertise when the statistics of the training data change over time — even a single deviating sample can significantly alter a module's expertise when the subtask in which the module specialized disappears from the training data.

This paper presents the modular DBN (M-DBN), an unsupervised method for training expert DBN modules that avoids catastrophic forgetting when the dataset changes. Similar to [9], only the module that best reconstructs a sample gets trained. In addition, M-DBNs use a batch-wise learning scheme in which each module is updated in proportion to the fraction of samples it best reconstructs. If that fraction is less than a small threshold value, the module is not trained at all. The experimental results demonstrate that these modifications to the original DBN are sufficient to facilitate module specializa-

All authors are at IDSIA, University of Lugano, SUPSI, Lugano, Switzerland. Email: {pape, tino, mark, juergen}@idsia.ch.

This work was supported by the EU under contract numbers FP7-ICT-IP-231722 and FP7-NMP-228844.

tion, and prevent the forgetting of learned features in tasks in which the statistics of the training data change over time.

In section II we briefly introduce DBNs, give the details of the M-DBN, and discuss related approaches. Next, we demonstrate module specialization on the MNIST handwritten digit dataset in section III-A. Section III-B shows that, in tasks where the digit classes change over time, the M-DBN retains the digits it has learned, while a monolithic DBN of similar size does not. We discuss our findings in section IV.

II. Modular Deep Belief Networks

A. Deep Belief Networks

The modular DBN introduced here is based on the DBN presented in [1], which consists of a stack of restricted Boltzmann machines (RBMs; [1]) trained one at a time (see Figure 1a). Each RBM has an input layer (visible layer) and a hidden layer of stochastic binary units. Visible and hidden layers are connected with a weight matrix and no connections exist between units in the same layer. Signal propagation can occur in two ways: *recognition*, where visible activations propagate to the hidden units; and *reconstruction*, where hidden activations propagate to visible units. The same weight matrix (transposed) is used for both recognition and reconstruction. By minimizing the difference between the original input and its reconstruction (i.e. reconstruction error) through a procedure called contrastive divergence [10], the weights can be trained to generate the input patterns presented to the RBM with high probability.

In DBNs, subsequent layers usually decrease in size in order to force the network to learn increasingly compact representations of its inputs. The training procedure is sometimes augmented to optimize additional terms, such as the L1 and L2 norms of the weight matrices, or sparsity constraints on the unit activations. Weights are initialized from a normal distribution with zero mean and small standard deviation. Weight updates are applied after the presentation of a number of samples in a minibatch. After a number of training cycles through the full training dataset, the stack of RBMs is unfolded, such that first recognitions are computed through all subsequent layers, and next reconstructions through all layers in reverse order. The recognition and reconstruction weights are uncoupled, and can then be fine-tuned with gradient descent, either to become better at reconstructing the inputs, or — in combination with other supervised or reinforcement learning methods — to form features relevant to the task at hand.

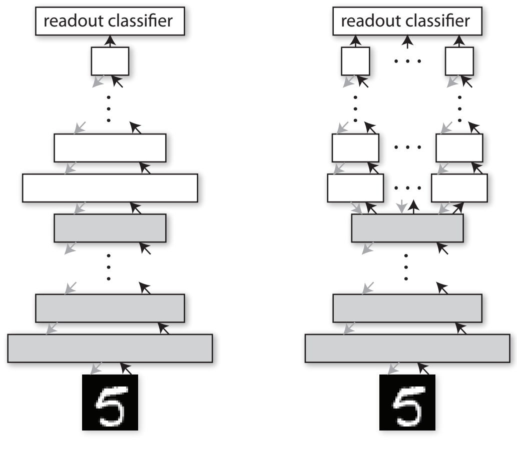

Fig. 1. Network architectures: (a) monolithic DBN and (b) modular M-DBN. Both architectures share the first few layers (gray boxes). Each set of two subsequent layers is trained as an RBM. Layers (boxes) compute their activations from the lower layer during the recognition step (black arrows), and reconstructions from the higher layer in the reconstruction step (gray arrows).

B. Modular DBNs that do not forget

The M-DBN consists of an ensemble of DBN modules, where all modules receive the same input pattern (see Figure 1b). The modules propagate their input through all layers, and from the final layer back through the network to reconstruct the input. Next, the reconstruction error for each module on its inputs is computed, and only the module with the smallest reconstruction error is trained one layer at the time for a small number of epochs. This procedure is repeated several times, until some predefined criterion is met.

Training is performed in a batch-wise fashion with a number of samples in a minibatch (as in [1]), times the number of modules. Thus, if the samples are equally distributed over the modules, each module is trained on the number of samples in a minibatch. Each module uses a learning rate that is proportional to the number of samples best reconstructed by that module. A threshold value is set for the modular learning rate, such that for small learning rates no weight updates are computed and applied. This training procedure facilitates specialization and prevents forgetting in three ways: (1) modules do not specialize in samples that are occasionally assigned to different modules in subsequent epochs, (2) a module is only updated if it specializes in features relevant to a sufficient number of samples (and only by updating can modules further specialize), and (3) a module will no longer be updated if the samples in which it specializes are removed from the training data.

The M-DBN can consist of completely separate DBNs, or some lower layers can be shared between modules (an in [11], Figure 1b). Especially in tasks where the first few layers learn primitive features that are present in all samples, sharing those layers among the modules can considerably reduce computational efforts. Of course, splitting a DBN into modules can be done multiple times in different layers. The reconstruction error in the first layer of the modules (which is not necessarily the input layer of the entire network) determines which module will be trained.

The computational cost of the M-DBN is similar to a monolithic DBN with the same number of weights. The forward and backward passes needed for computing the reconstruction errors are required for the training procedure of both architectures, so they incur no additional cost in the M-DBN. However, since the M-DBN is no longer trained one layer at a time, the number of epochs needed for convergence increases. Every time a layer is updated, the layers that receive input from the updated layer need to be adjusted as well. Also, the M-DBN incurs a very slight extra cost for comparing the module reconstruction errors in order to determine which module should be trained. On the other hand, weight updates are much cheaper in the M-DBN because only a single module is trained per sample. Additionally, the M-DBN can be easily parallelized, because each module can be treated independently.

C. Related work

Other methods have been proposed to train modular DBN architectures, usually based on modified weight update equations that take into account the modular reconstruction error. [9] showed that mixtures of autoencoders are able to specialize in different handwritten digit classes. [12] used the reconstruction error of modular autoencoders in the weight updates to force the output of at least one module to fit the input, and to force the rest of the modules to increase the error between the input and the output. [13] used a neural gas mixture of local principal component analyses. [14] showed successful specialization of modular DBNs on limited low-dimensional data.

Compared to the M-DBN presented here, these methods are more complex, involve additional adjustable parameters and are mostly used to demonstrate module specialization or to achieve increased performance on fixed datasets. Most existing modular DBN approaches use module competition to update the weights in all modules instead of just the best module. While updating all modules for each sample may lead to faster and better specialization, it comes at considerable computational cost and makes the network prone to forgetting. To prevent expert modules from forgetting their skills, module updates must be truly local. However, updating one module for each sample is necessary, but not sufficient to prevent forgetting: even a single sample can cause complete readjustment of the module's expertise, in case the class in which the module specialized is no longer present.

Another related method to prevent forgetting is to use distinct modules for short term and long-term memory [15]. In this approach, newly learned information is transferred slowly to the long-term memory, with the help of so-called pseudopatterns (basically pairs of input-output data for random input patterns). For an overview of related ideas, see [16]. The modular DBN resembles these approaches to some extent, because it stores samples for one training epoch in short term memory, determines some properties of the sample distribution, and uses these properties to train the modules, the equivalent of long-term memory.

III. Experiments

We performed a number of experiments to study the ability of the modular DBN (M-DBN) to specialize and retain learned skills on several handwritten digit recognition tasks, and compared it to a non-modular architecture (simply referred to as DBN) of comparable size. Figure 1 shows the two competing architectures used in the experiments. The M-DBN and DBN share the first few layers, but differ in the remaining upper layers. Whereas the upper layers in the DBN are fully connected to each other, just like the shared layers, the M-DBN has a set of separate smaller upper layers for each of its modules. The upper DBN layers were sized such that the total number of weights in the two networks is roughly equivalent, and its final-layer size is equal to the combined final-layer sizes of the modules.

All experiments used the MNIST dataset [17] which contains 70 000, 28×28, pixel images of handwritten digits 0-9. To reduce computation time and to simplify the presentation of the results, the experiments were restricted to digits 0-5 (two thirds of the total dataset). Following common practice, 10 000 images were used as test set.

Both the DBN and M-DBN used four shared layers with layer sizes 1000-500-500-100. These shared layers were first trained for 50 epochs on the MNIST training dataset and then frozen, which greatly reduced computation time, as the subsequent layers did not have to adjust to changes in the shared weights. Training the shared layers with a subset of the MNIST digits only slightly changed the final outcome, because the shared layers capture only local feature primitives such as small dots and stripes that are common to all digit classes.

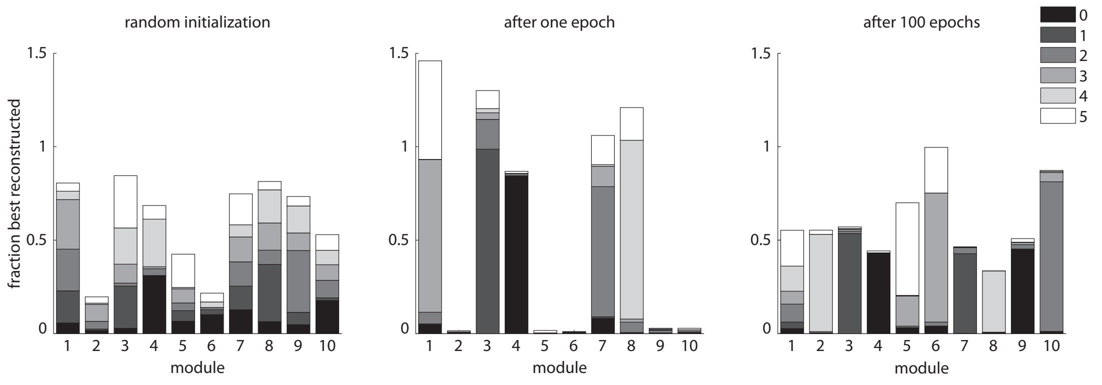

Fig. 2. Fraction of best reconstructed samples per module and digit class during different stages of training: immediately after random weight initialization (left), after one training epoch (center) and after 100 training epochs (right). Note that fractions for each digit class add up to 1, and are stacked per module.

The M-DBNs used in the experiments consisted of 10 modules ($n_{\mathcal{M}} = 10$) with increasingly smaller upper layer sizes of 70-40-20-10-4-1. Weight updates and reconstruction errors for the M-DBNs were computed and applied in minibatches of size $100\, n_{\mathcal{M}}$, yielding a modular minibatch size of 100 in case the samples are equally distributed over all modules. The learning rate $\alpha^{\mathcal{M}}$ of each module $\mathcal{M}$ was set proportional to the fraction of samples $x^{\mathcal{M}}$ that module $\mathcal{M}$ reconstructed best:

$$\alpha^{\mathcal{M}} = \alpha\, x^{\mathcal{M}}\, n_{\mathcal{M}}.$$

The global learning rate α was set to 0.1, and the modular learning rate $\alpha^{\mathcal{M}}$ was not allowed to become larger than α. If the fraction of best reconstructed samples by a module was less than $0.3/n_{\mathcal{M}}$, the module was not trained. The effect of different layer sizes and number of modules is discussed below.

The non-modular DBNs had the same number of layers as the M-DBNs, roughly the same number of weights, and a final-layer size equal to the combined final-layer sizes of the M-DBNs. For example, a ten-module M-DBN with upper layers sizes 70-40-20-10-4-1 (108 440 weights), was compared to a DBN with upper layer sizes 314-179-90-45-18-10 (108 756 weights). Note that a large portion of the M-DBN weights ($\approx$70%) is found between the last shared layer and the first module layers. The DBN used a minibatch size of 100, and learning rate of 0.1. For a fair comparison with the modular approach, each DBN layer was trained for 10 consecutive epochs before training the next layer. The other DBN learning parameters are the same as in [1]. As the purpose of this work is to study module specialization and forgetting, not to achieve best performance on MNIST classification, no fine-tuning of the weights with gradient descent was performed.

To test the degree to which the modules specialized in a digit class, we computed a specialization index for each module $\mathcal{M}$ as:

$$s^{\mathcal{M}} = \max_i \frac{x_i^{\mathcal{M}}}{x^{\mathcal{M}}}$$

where $x_i^{\mathcal{M}}$ is the fraction of samples of digit class $i = 0 \ldots 5$ that were reconstructed best by module $\mathcal{M}$. The higher the index, the more a module prefers a single digit class. The average specialization index:

$$\overline{s} = \sum_{\mathcal{M}} x^{\mathcal{M}} s^{\mathcal{M}}$$

indicates the degree to which modules overlap, and how much the M-DBN is specialized as a whole. An average specialization index of 1 implies that each module specialized in exactly one digit class, and a specialization value lower than 1 indicates that one or more modules specialized in more than one digit class.

To determine utility of the learned compact representations during the experiments, a support vector machine (SVM, [18]) was trained every 10 epochs to classify digits 0-5, based on the M-DBN and DBN final-layer representations of the MNIST test set. Two-thirds of the MNIST test set was used to train the SVM. Classification performance was measured as the fraction of correctly classified samples $\overline{r}$ on the remaining one-third of the test dataset. For the M-DBN, the activations in the final module layers were concatenated with a vector containing a 1 at the element of the best module, and zeros for all other modules. The SVM used the implementation from [19] for multi-class classification with a radial basis kernel. Using a grid-search for finding the values of adjustable SVM parameters (as described in [19]), C and γ were set to 16 and 2, respectively. Changing these values several orders of magnitude did not significantly alter the results. Other parameters were set to the defaults from [19].

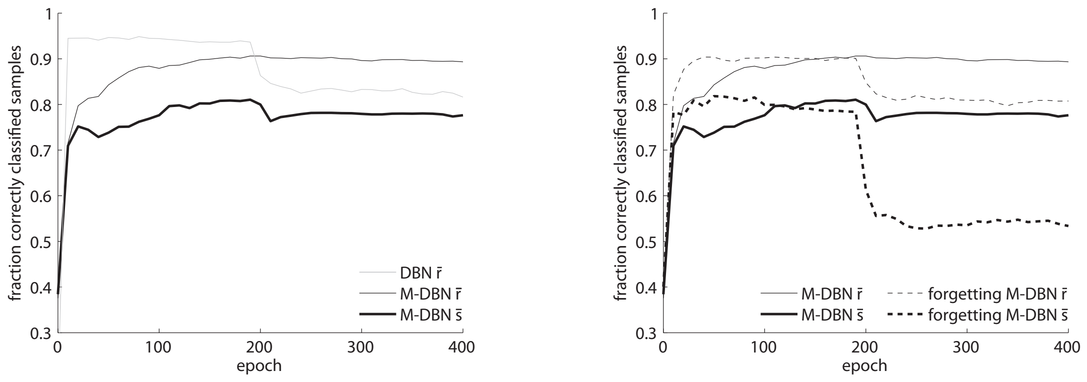

Fig. 3. Performance measures of the DBN and M-DBN (left); and the forgetting M-DBN and M-DBN (right) in the digit removal task. Training starts with digits 0-5 and continues with digits 0-2 after epoch 200. Classification performance $\overline{r}$ and specialization index $\overline{s}$ are based on digits 0-5 in the MNIST test data.

A. *Module specialization*

We first investigate the ability of the M-DBN to specialize its modules to different digit classes. Figure 2 shows the fraction of digits best reconstructed digits by each module during several stages of the training, for an example run. Initially, the modules do not exhibit obvious digit preferences, but after just one training epoch, some of the modules have already specialized in one or two digit classes. Apparently, the slight initial differences in the distribution of the digits over the modules are sufficient to seed significant module specialization. However, some of the modules specialize in more than one (usually two) digit classes (e.g., module 1 after one epoch). In many experiments we found that there was a module that specialized in both digits 3 and 5. Often several tens of epochs were needed for further separation, and complete separation of these digit classes was not always achieved (e.g., modules 5 and 6 after 100 epochs). While this might be a problem for tasks in which these digits are not always available in the training data at the same time, the performance of the readout classifier was not affected by the lower specialization indices — distinctions between digits were still possible based on the final-layer representations of the unspecialized modules.

These findings reveal that modules specialize in two stages: (1) rapid adjustment toward one particular class in the first few epochs, and (2) slow forgetting of classes that are less frequently assigned to a module in case initial module specialization was not fully achieved. If samples from multiple classes are assigned to a single module, the module will eventually be tuned toward the class with the highest number of samples for that module. However, if there are no other modules specializing in one of the multiple classes that a module reconstructs best, no further specialization will occur. For example, if digits 1, 3 and 5 are supplied to three modules, two of the modules might initially specialize in upright and slanted '1's respectively, and the third will specialize in both digits 3 and 5. This means that there are not enough modules to make the same distinctions made by humans, and indicates that some aspects of the task should be known beforehand in order to solve it optimally. Increasing the number of modules (up to 20) improved specialization, but, as mentioned above, the focus here is not on absolute performance, but on the ability of modular DBNs to specialize and not forget.

B. *Nonstationary digit tasks*

Next, we investigate the capacity of the M-DBN to retain learned features in two nonstationary digit tasks: (1) removal of digits classes from the training data, and (2) replacement of one set of digit classes with another set of digit classes. In the removal task, the networks are trained for 200 epochs on digits 0-5, and then another 200 epochs on digits 0-2 only. This task tests whether the networks ability to retain learned features after these features have disappeared from the training data. In the replacement task, the training set alternated between digit classes 0-2 and 3-5 every 200 epochs. This task tests how well the networks retain learned features after they disappear from the training data, while simultaneously learning new features. Five differently initialized DBNs and M-DBNs were trained in each task. As described before, a readout SVM was trained each ten epochs to classify digits 0-5 in the MNIST test data, using the final-layer representations.

Figure 3 shows the performance of the DBN and M-DBN on the removal task. For comparison, we also included the results for an M-DBN that used the fixed, global learning rate α for all modules, instead of the module-specific learning rate $\alpha^{\mathcal{M}}$. As this version uses specializing modules, but has no mechanism to prevent forgetting, it is called the "forgetting M-DBN".

Initially, the readout classifier of the DBN produces the best classification, but after several hundred epochs, the performance for the M-DBN catches up. At epoch 200, classes 3-5 are removed from the training data, and training continues with only classes 0-2. Shortly after, the performance for the DBN starts to decrease. However, about 85% of the samples are still correctly classified by the DBN after epoch 200, even though only 50% of the digit classes are present in the training data. This is either because the DBN retains some of the learned features, or because the representations for the remaining classes can still be used to correctly classify the missing classes. To verify which hypothesis is correct, we trained five DBNs on classes 0-2, and determined the classification performance of readout SVMs based on final-layer representations for digits 0-5 in the MNIST test data. We found a similar classification rate of about 85% as for a DBN that *was* trained on classes 0-5 before (Figure 3), so the conclusion is that the DBN does forget the relevant features for distinguishing digits that are no longer present in the training data.

In contrast, the readout classifier's performance for the M-DBN did not decrease after the removal of classes 3-5. We observed that the distribution of the removed digit classes 3-5 over the modules did not change, indicating that these modules retain their specialization. The specialization index of the forgetting M-DBN, on the other hand, sharply decreased after removing digits 3-5 from the training data. As module specialization was never completely perfect, modules that largely specialized in digits 3-5 were often also best at reconstructing a very small number of digits 0-2. Without an adjusted learning rate, the small number of best reconstructed samples from other classes eventually caused the modules to adjust their expertise in the forgetting M-DBN to the remaining classes. Just as for the DBN, the forgetting M-DBN forgot the relevant features needed to identify the digits no longer present in the training data, causing a decrease in the performance of the readout classifier.

Figure 4 shows the results for the replacement task. As in the removal task, the M-DBN outperforms the DBN after replacement of the classes. Even after several presentations of the different sets, the DBN still forgets the learned features after the dataset changes. However, limiting training to classes 0-2 or 3-5 does not cause the same drop in performance for the DBN. Again, the representations learned for one set of digit classes could to a certain extent be used by the readout classifier to distinguish digit classes that are not present in the training data. Sometimes the specialization index of the M-DBN decreases (e.g., around epoch 900). Inspection of the distribution of digit classes over the modules revealed that this is caused by a redistribution of part of the samples to a different module, eventually leading to a better performance of the readout classifier.

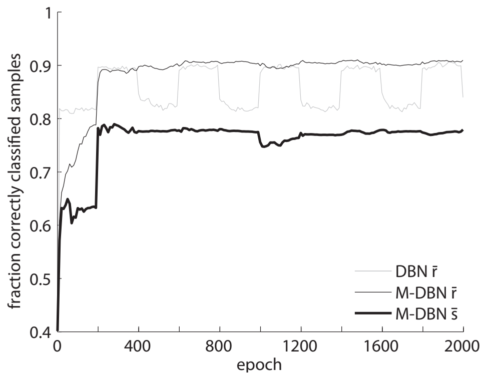

Fig. 4. DBN and M-DBN performance on the digit replacement task. Ticks on the horizontal axis indicate changes in the training dataset. Classification performance $\overline{r}$ and specialization index $\overline{s}$ are based on digits 0-5 in the MNIST test data.

IV. Discussion

The M-DBN is able to learn expert modules that each specialize in a subset of the MNIST handwritten images, roughly corresponding to the digits discerned by human beings. While the modular approach of [9] was already able to specialize in different digit classes, as we have shown, modularization alone is not sufficient to prevent forgetting. The M-DBN presented here is able to retain module specialization after some of the digits are removed from the training data, or are replaced with a different set of digits. The learned features of the M-DBN can be used by a subsequent readout classifier, even after removal or replacement of some of the digits, while a non-modular network of similar size forgets what it learned before.

We tried different network architectures with more or less shared layers (no shared layers, shared layer sizes 500-100, 1000-100, 1000-500-100 and 2000-500-500-100). Smaller shared layers slightly decreased performance of the readout classifiers, probably because not all the relevant variability could be captured. Larger shared layers did not significantly change the final classification results. We also tried different sizes for the last shared layer (10, 20, 40, 80, 100, 150, 400 and 500), and found that for small layer sizes (<80) no performance difference occurred between the DBN and M-DBN. Since shared layers were trained on the MNIST training set and then frozen, small last-shared-layers already learned distinctive representations for digit classes, making modular specialization obsolete. Larger last-shared-layers represent more local features found in many digits, causing specialization in different digit classes only in subsequent modular layers. Increasing the size of the final module layers hardly effected specialization. However, monolithic networks with large final-layers achieved results comparable to modular networks in the nonstationary tasks. This is probably because there is no need to represent global image features, such as digit class, in larger layers. The more local image features represented by larger layers can still be used by the readout classifier to determine the digit class (though at considerable computational cost for the readout classifier).

The performance of both architectures on the nonstationary tasks is lower than most classification methods on MNIST (e.g., linear classifiers achieve 88% correct; more advanced approaches without preprocessing achieve above 95% correct classification rates [17, 20]; the current published record is 99.65%, by [21]). The tasks considered here are different than tasks where all samples are available to the learning algorithm, and are equally distributed over the digit classes. Fine-tuning the DBNs with gradient descent, increasing the number of modules, module sizes and training epochs might further improve classification results. However, the purpose of this study was not to obtain the best results on MNIST digit classification, but to demonstrate a modular DBN architecture that does not forget.

Some extensions to the M-DBN may be worth considering. One is to make the number of modules flexible, creating and merging modules dynamically. For example, two modules that achieve almost the same best reconstruction error for a certain subset of the samples, could be merged by resetting or removing one of the two. Finding a good criterion for module creation is less straightforward. It seems natural that new modules should be created to reduce reconstruction error, yet neither large reconstruction error nor large variance in reconstruction error implies that a module is failing to specialize. Rather, these values often indicate that a module is adapting slowly or is specializing in a class that is intrinsically difficult to reconstruct. However, it might be possible to use feedback from a subsequent supervised or reinforcement learning algorithm as a criterion for module creation. For example, in an reinforcement learning task where the M-DBN's output is used for predicting state or action values, a large variance in temporal difference error could suggest that further specialization is required, and that more modules should be created.

A possible method to speed up learning is to randomly reinitialize the weights of the modules that never reconstructed any of the samples well during the entire training process. The module's new weights might be closer to some subtask of the training data on which it can subsequently specialize. An additional method to facilitate specialization and prevent forgetting is to maintain an excess number of randomly initialized modules that are never trained, so that samples that are better reconstructed by a random module than by a trained module will not affect the trained modules.

We expect the M-DBN approach to be particularly useful for continual learning, especially in tasks where the behavior of a learning agent drives it to previously unexplored regions of the environment with a different distribution over observations. In reinforcement learning, for example, using adaptive unsupervised preprocessors have become increasingly popular (e.g., [22]). However, features generated by algorithms that forget what they learned before, may no longer be valid for parts of the environment the agent has not visited recently. In these cases, the M-DBN can be used as a building block for dimensionality reduction in a learning agent that does not forget what it learned before.

V. Conclusion

This paper has introduced the M-DBN, a modular DBN architecture and learning algorithm for unsupervised feature generation in continual learning tasks. The M-DBN allows features to be developed that are retained when the distribution over input patterns changes — a critical property for continual learning. Experiments using the MNIST handwritten-character database demonstrated that M-DBN modules become expert at distilling features for the patterns in which they specialized. Furthermore, that expertise was not lost when the input distribution changed. In contrast, a monolithic DBN of comparable size quickly forgot the features it had learned for input patterns that disappeared from the training set.

References

[1] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[2] Y. Bengio, P. Lamblin, P. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 19. Cambridge, MA.: MIT Press, 2007.

[3] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[4] M. B. Ring, "Continual learning in reinforcement environments," Ph.D. dissertation, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas, August 1994.

[5] M. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Proceedings*

of Computer Vision and Pattern Recognition Conference, Minneapolis, Minnesota, 2007.

[6] I. Sutskever and G. E. Hinton, “Learning multilevel distributed representations for high-dimensional sequences,” in *AI and Statistics*, Puerto Rico, 2007.

[7] G. W. Taylor, G. E. Hinton, and S. Roweis, “Modeling human motion using binary latent variables,” in *Advances in Neural Information Processing Systems (NIPS)*, vol. 19. Cambridge, MA.: MIT Press, 2007.

[8] R. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive mixtures of local experts,” *Neural Computation*, vol. 3, pp. 79–87, 1995.

[9] G. E. Hinton, M. Revow, and P. Dayan, “Recognizing handwritten digits using mixtures of linear model,” *Advances in Neural Information Processing Systems*, vol. 7, pp. 1015–1022, 1995.

[10] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural Computation*, vol. 14, pp. 1771–1800, 2002.

[11] A. Erkan, R. Hadsell, P. Sermanet, K. Kavukcuoglu, M. Ranzato, U. Muller, and Y. LeCun, “Selfsupervised learning from high dimensional data for autonomous offroad driving,” in *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[12] H. Ando, S. Suzukia, and T. Fujitaa, “Unpervised visual learning of three-dimensional objects using a modular network architecture,” *Neural Networks*, vol. 7-8, pp. 1037–1051, 1999.

[13] B. Zhang, M. Fu, and H. Yan, “A nonlinear neural network model of mixture of local principal component analysis: application to handwritten digits recognition,” *Pattern Recognition*, vol. 34, pp. 203–214, 2001.

[14] K. J. Kurtz, “The divergent autoencoder (DIVA) model of category learning,” *Psychonomic Bulletin & Review*, vol. 14, pp. 560–576, 2007.

[15] R. M. French, B. Ans, and S. Rousset, *Pseudopatterns and dual-network memory models: advantages and shortcomings*. Berlin: Springer, 2001, pp. 13–22.

[16] A. Robins, “Sequential learning in neural networks: A review and a discussion of pseudorehearsal based methods,” *Intelligent Data Analysis*, vol. 8, pp. 301–322, 2004.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[18] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.

[19] C. Chang and C. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[20] R. Salakhutdinov and G. E. Hinton, “Learning a nonlinear embedding by preserving class neighbourhood structure,” *AISTATS*, vol. 11, 2007.

[21] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep big simple neural nets excel on handwritten digit recognition,” *Neural Computation*, vol. 22, pp. 3207–3220, 2010.

[22] S. Lange and M. Riedmiller, “Deep auto-encoder neural networks in reinforcement learning,” in *International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain, 2010.