ENSEMBLENET: A NOVEL ARCHITECTURE FOR INCREMENTAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks are used in many state-of-the-art systems for machine perception. Once a network is trained to do a specific task, it cannot be easily trained to do new tasks as it leads to catastrophic forgetting of the previously learned tasks.We propose here a novel architecture called *EnsembleNet* that accommodates for newer classes of data without having to retrain previously trained sub-models. The novelty of our model lies in the fact that only a small portion of the network has to be retrained which makes it computational efficient and also results in higher performance compared to other architectures in the literature. We demonstrated our model on MNIST Handwritten digits, MNIST Fashion and CIFAR10 datasets. The proposed architecture was benchmarked against other models in the literature on $\Omega_{new}, \Omega_{base}, \Omega_{all}$ metrics for MNIST- Handwritten dataset. The experimental results show that Ensemble Net on overall outperformed every other model in the literature.

1 INTRODUCTION

Many times in the real world setting, it becomes essential for an existing neural network to learn a new class. For example: A Deep Neural Network (DNN) that is trained to recognize cats might have to be trained to recognize a new species of cat. However, current DNNs are not capable of meeting the need. Whenever a DNN is trained on a new subset of data, the weights that are responsible for retaining the old information are disturbed. This leads to forgetting of the old data that it learnt and this is called as Catastrophic Forgetting. Human brains are very good at this and make it look easy, but emulating this behaviour in machine learning models has proven to be a challenging task. To solve this problem, many techniques have been proposed in the past . Most of them include freezing the weights that are important for the previous data (Fernando et al. (2017)) (Kirkpatrick et al. (2017)) or retraining the entire neural network on the previous data. Retraining the entire model on entire previous data is a computationally intensive task and might require weeks of retraining in few cases. It would be computationally very efficient if we can retrain only a small portion of the model instead of the entire neural network.

In this paper we proposed a model that will address this issue. The proposed architecture performs rehearsal of the previous data on only a small portion of the entire model. The proposed architecture contains two layers of neural networks and only the neural network that is present in the final layer is exposed to rehearsal. We have shown that this partial rehearsal technique is sufficient to achieve state of the art performance on $\Omega_{base}, \Omega_{new}, \Omega_{old}$ metrics Kemker et al. (2018) where Ω_{base} is the ability of a model to retain the base knowledge, Ω_{new} is the ability of a model to retain the new knowledge and Ω_{all} is the overall knowledge retention capacity of a model.

As a consequence of the proposed models architecture, it can be trained on a distributed computing network in an asynchronous manner on heterogeneous systems. Many models and architectures have been proposed earlier which train a neural network using a distributed computing system Dean et al. (2012). However, all of the proposed methods involve data transfer between the systems in the distributed network. To ensure smooth data transfer between systems, it is highly essential that all the systems in the network are synchronised and are online. This requirement reduces the robustness of those methods. However, our proposed architecture does not require any data transfer between systems. This allows all the systems in the network to be asynchronous. The training will not be affected even if one system goes offline.

2 Related Work

Many architectures have been proposed in the past which are capable of incremental learning. Some of the previously proposed architectures utilize Ensembling methods while other architectures propose using specialized architectures. For example in Polikar et al. (2001) and He et al. (2011) ensemble methods were used to achieve incremental learning. They are capable of learning new data that is flowing into the system as well as learning new classes incrementally. Connolly et al. (2008), Polikar et al. (2001), Gabrys & Bargiela (2000), Fritzke (1995), Bouchachia (2006) were proposed which do not use any ensembling techniques. These algorithms have smaller memory footprint compared to ensemble methods but are incapable of incrementally learning new classes.

Our proposed architecture is similar to a stacking method which can learn new incoming data as well as incrementally learn new classes.Apart from incremental learning capabilities the proposed architecture can be trained in a distributed manner in an asynchronous fashion on heterogeneous computers in parallel. Some work has already been done in trying to achieve distributed training of machine learning models like the two algorithms proposed in Dean et al. (2012) where the neural network is split to smaller parts and then trained in parallel. Some of the existing distributed systems infrastructure like Map Reduce Dean & Ghemawat (2008), Graph Lab Low et al. (2012) has been utilized by the machine learning community to implement their algorithms. But none of the proposed distributed training methods are capable of incremental learning. The novelty of our architecture lies in the fact that our model is capable of incremental learning new classes with minimal rehearsal and also being capable of trained in a distributed manner.

3 Proposed architecture

The architecture consists of two levels of classifiers. In the first level, there are k classifiers where k is the number of classes in the data that the model has been trained on till now. Each classifier in the first layer is trained to recognize datapoints that belong to only one class. If they see a datapoint that they are not trained to recognize then they output 0. In the second level there is only one classifier and it is called as the final classifier. The final classifier takes input from all the classifiers in level 1 and predicts the class of the given input sample point. The proposed architecture is similar to StackingC architecture Seewald (2002). However, StackingC architecture doesnot append new classifier to the first layer of classifiers in-order to learn new classes incrementally. Another key difference between our architecture and StackingC is that unlike StackingC, in our architecture the final classifier takes in the confidence of the classifier and not the predicted output of the level 1 classifier.

The architecture proposed is built using an iterative algorithm where each new class is added in a one-vs-rest fashion. For adding a class 'K', the training data is split into positive and negative data, where positive points correspond to class K points. The negative data corresponds to points of all other classes than K. The negative data is only a subset of points from each of the other classes.

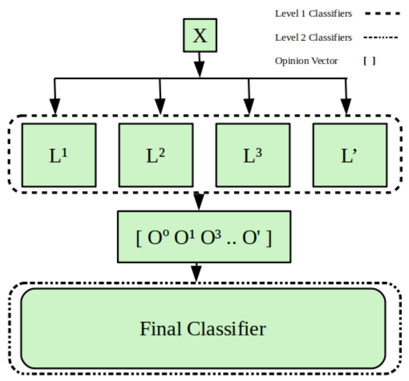


Figure 1: Architecture of EnsembleNet

The algorithm is presented in [reference to algorithm] table. The Algorithm 1 describes the training procedure of the proposed architecture. The training procedure takes a data point x_i where i is the class to which the datapoint belongs. If there is already a classifier l_i in the first level of the EnsembleNet that is trained to recognize datapoints belonging to class i, then the new datapoint is fed to l_i as a training point. If the label is not covered, then a new model is built and added to the net. The training data for this new classifier consists of X_i, where X_i is the set of all the new points belonging to class i and $X_{i-1,i-2,i-3..}$ is the subset of points belonging to all the previous classes. Here, X_i is considered as positive class and $X_{i-1,i-2,i-3..}$ considered as negative class. Finally, the entire training data is rehearsed to the final classifier by freezing the classifiers present in the first level and only retraining the final classifier.

Algorithm 1 EnsembleNet - Training algorithm

```
1: $X_N$ // New data set
2: $X$ // Entire data set
3: $x_i$ // Datapoint belonging to class $i$
4: $I$ // Set of all classes that EnsembleNet already knows
5: $l_i$ // Classifier trained to recognize $i$ class points
6: $l(x)$ // Opinion of classifier $l$ on datapoint $x$.
7: $L$ // Set of all classifiers in Level 1
8: $\Theta$ // Opinion vector
9: $F_C$ // Final classifier
10: for $x_i$ in $X_N$ do
11:     if $i \notin I$ then
12:         A New classifier $l$ is created
13:         $l$ is trained on with $i$ class as positive points and subset of rest classes as negative points
14:         $L = L \cup l_i$
15:     else
16:         $l_i$ is trained on $x_i$
17:     end if
18: end for
19: for $x$ in $X$ do
20:     $\Theta = \{\}$
21:     for $l$ in $L$ do
22:         $\Theta = \Theta \cup l(x)$
23:     end for
24:     $F_C$ is trained on $\Theta$
25: end for
```

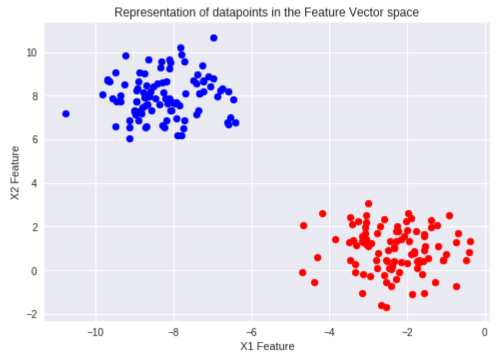
Algorithm 2 EnsembleNet - Prediction algorithm

```
1: $X_T$ // Test data
2: for $x_i$ in $X_T$ do
3:     $\Theta = \{\}$
4:     for $l$ in $L$ do
5:         $\Theta = \Theta \cup l(x)$
6:     end for
7:     $prediction = F_C(\Theta)$
8:     print(prediction)
9: end for
```

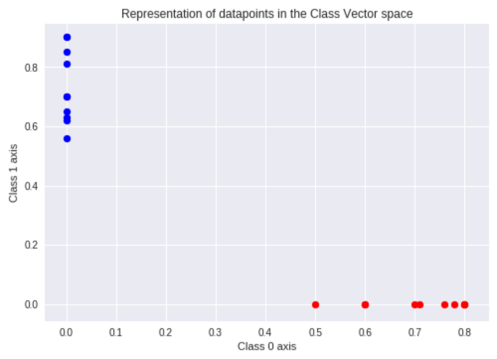
In the prediction phase, a given input point x is passed through each of the existing classifiers in layer 1.[Algorithm 2]. A classifier in the first level outputs it's confidence only when it recognizes the point. This is called as *opinion* of the classifier. For every point, *opinion* of all the classifiers in the first level is gathered and is passed to the final classifier for classification.

3.1 VECTOR SPACE TRANSFORMATION OF DATAPOINTS

Whenever datapoints are projected in a vector space, they are usually represented in the n-dimensional *feature vector space* (Figure 2.a), where x_1, x_2, x_3 ... are features. When these datapoints are passed to the level 1 of the EnsembleNet a feature transformation takes place and the points are now projected on a different vector space where the vectors are the confidence of a point belonging to a particular class. We can call this new vector space as *class vector space*. There are two possible cases that could occur. In the first case, only one classifier from layer 1 recognises the point and the point will lie along one axis in the *class vector* (Figure 2.b). If all the points in the dataset fall under case 1, then the final classifier can be a single perceptron. But this is not the case



(a) Datapoints projected in the feature vector space



(b) Datapoints projected in the class vector space (Ideal case)

Figure 2: Comparision of vector space representations

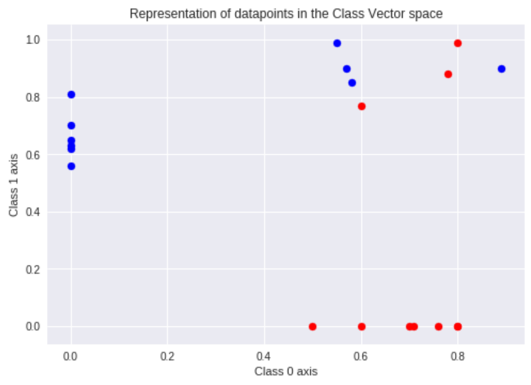


Figure 3: Datapoints projected in the class vector space (Non - Ideal case)

always. Many times, for a given datapoint, more than one classifier in layer 1 is triggered and the points donot lie along any axis as shown in Figure 3. Hence, a multi-layer perceptron was used to perform the final classification task.

3.2 Training time of EnsembleNet

The architecture of EnsembleNet allows it to be trained on an asynchronous distributed system with ease. All the classifiers in the first level are independent of each other and can be trained independently. Hence, parallelizing the training of the first level of classifiers by training each classifier on one system in a distributed network will greatly reduce the training time. If one classifier is trainined on one sub-system of a distributed system, then the training time (T) will be

$$T = max(T_{l_1}, T_{l_2}, T_{l_3}..T_{l_k}) + T_{F_C} \tag{1}$$

where T_{l_k} is the time consumed by a system to train the l_k classifier on one sub-system in the distributed system and T_{F_C} is the time taken to train the final classifier. If the distributed system contains only n nodes, then equation 1 can be re-written as

$$T = max(T_1, T_2, T_3..T_n) + T_{F_C} \tag{2}$$

where T_1 is the time spent by the first node in the system on training the first classifier and T_2 is the time spent by the second node in the system on training the second classifier and so on. n denotes the total number of nodes in the distributed system. The proposed architecture's training time was benchmarked against a standard CNN and standard CNN with rehearsal on Multi task learning experiment. The results have been tabulated in the Experimental results section of the paper in Table 4. The total time taken by the model to train on the base dataset is tabulated under the T_B column and the total time to train on the new dataset is tabulated under the T_N column. Total training time of the model is represented as T_F.The time required by the CNN on rehearsing the base dataset have been added to T_B. The experimental results demonstrate that EnsembleNet trains faster compared to a traditional CNN and CNN using rehearsal while demonstrating higher data retention capacity. A thorough mathematical analysis regarding training time and error analysis of the proposed architecture was performed and made available in the supplementary material section.

4 Experimental Setup

4.1 Dataset Description

4.1.1 MNIST-Handwritten Digits

MNIST Handwritten digits is a classic dataset in Machine learning and contains grayscale images of handwritten digits with 10 classes. The images are 70,000 in number and have a dimensions of 28x28 each. 60,000 training samples were selected as train data and 10,000 training samples were selected as test data.

4.1.2 CIFAR10

CIFAR10 consists of 50,000 training samples and 10,000 test samples where each point belongs to one of 10 mutually-exclusive object categories. Each image has a dimension of 32x32.

4.1.3 MNIST- Fashion

Apart from Handwritten and CIFAR10, the architecture was also tested on MNIST Fashion dataset. The dataset has 70,000 gray-scale images of fashion products belonging to 10 categories. The dimesions of each image in the dataset is 28x28. 60,000 images were chosen as training data and 10,000 images were chosen as test data.

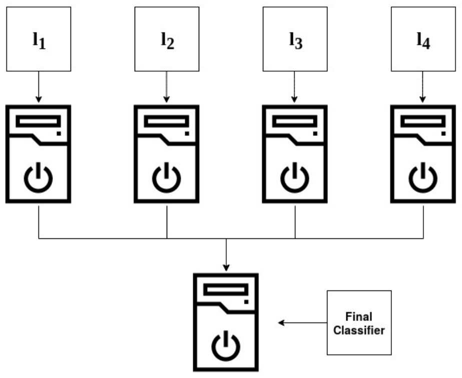


Figure 4: Parallel training of level 1 classifiers on a distributed system

	Handwritten digits	Fashion	CIFAR10
Classification Task	Grayscale Image	Grayscale Image	RGB Image
Classes	10	10	10
Image Shape	28x28	28x28	32x32
Train Samples	60,000	60,000	50,000
Test Samples	10,000	10,000	10,000

Table 1: Dataset Specifications

4.2 Models Evaluated

The proposed architecture was evaluated against the following architectures: 1) Standard Multi-layer Perceptron 2) MLP with Elastic Weight Consolidation 3)GeppNet 4)Fixed Expansion layer architecture and 5) GeppNet+STM on MNIST - Handwritten dataset. The benchmarking performance of each model on MNIST Handwritten dataset was taken from Kemker et al. (2018). The experiment that was proposed there was replicated using the same dataset without any changes. The evaluation of the $\Omega_{base}, \Omega_{new}$ and Ω_{all} were based on the α_{ideal} that were described in the paper.

4.2.1 Standard Multi-Layer Perceptron

A standard Multi Layer Perceptron was used as a baseline by Kemker et al. (2018). A hyperparameter search for the number of units per hidden layer (32-4,096) and number of hidden layers (2-3) was done to obtain the optimal model for the given task. This model was tested for incremental class learning by declaring the number of classes initially and then adding new sessions of data.

4.2.2 Elastic Weight Consolidation

In EWC, the weights that are important for remembering the previous data are disturbed comparatively less when new data is being added to the model. This is achieved by using a modified loss function

$$L(\theta) = L_t(\theta) + \sum_i \tfrac{\lambda}{2} F_i(\theta_i - \theta_{A,i})^2$$

F is the Fisher information matrix which is used to constrain the weights that are important for retaining previously learned data. A detailed description of the loss function can be found at Kirkpatrick et al. (2017)

4.2.3 GeppNet and GeppNet+STM

GeppNet and GeppNet+STM Gepperth & Karaoguz (2016) are biologically inspired networks that deploy rehearsal to mitigate forgetting of previous data. GeppNet consists of a SOM layer and a Linear regression classifier. The SOM layer is updated only when the new training sample has been determined as novel by the model. The novelty of the datapoint is determined using the confidence measure. When GeppNet+STM detects a novel point, it stores it in the STM (Short Term Memory). It replays the samples in the STM during a sleep phase. GeppNet and GeppNet STM store the previous data however, in GeppNet+STM a training example is only replayed if the model is uncertain about the prediction.

4.2.4 Fixed Expansion Layer

FEL Coop et al. (2013) is a two layer Multi layer perceptron with the second layer being bigger than the first layer. The weights in the second layer are sparse and remain fixed during the training. Only some of the units in the first layer are updated because only a subset of the FEL layer units are allowed to have non-zero output to the final classification layer. Moreover, only a subset of units in the FEL layer are connected to the units in the first layer.

4.3 Experimental Results

4.3.1 Incremental Class learning

The experiment that was performed was Incremental class learning, where new classes are added to the model iteratively and the ability of the model to retain the base knowledge, new knowledge and the overall knowledge is measured. For all the three datasets, classes from 0 to 4 were considered as base knowledge and classes 5 to 9 were added iteratively to the model.

For MNIST-Handwritten digits dataset, although MLP and FEL showed better capability to retain new data, they both performed poorly when their ability to retain Base data and Overall data were measured. Elastic weight consolidation technique demonstrated higher capability of retaining base knowledge, but the weight consolidation prevented the model from learning new data effectively. GeppNet and GeppNet+STM performed better compared to other models, but our model outperformed both the models in all the metrics. EnsembleNet showed significantly higher ability to retain new knowledge comapared to the GeppNet variants. The Geometric mean of Ω_{Base}, Ω_{All} and Ω_{new} has been calculated and is represented as Ω_G. EnsembleNet outperformed all other models in Ω_G as well.

The model was also tested on MNIST-Fashion dataset and CIFAR10. 0-4 classes were considered as the base knowledge and 5-9 classes were incrementally added in successive sessions.The proposed

Model	Ω_{new}	Ω_{Base}	Ω_{All}	Ω_G
MLP	1.000	0.60	0.181	0.477
EWC	0.001	1.000	0.133	0.0510
GeppNet	0.824	0.960	0.922	0.900
FEL	1.000	0.451	0.439	0.5828
GeppNet+STM	0.599	0.919	0.824	0.768
Ours	**0.973**	**0.979**	**0.980**	**0.977**

Table 2: Incremental class learning results on MNIST Handwritten digits dataset. The benchmarking results were taken from Kemker et al. (2018)

.

Dataset	Ω_{base}	Ω_{new}	Ω_{all}
MNIST-Fashion	0.943	0.941	0.963
CIFAR10	0.9231	0.999	0.967

Table 3: Incremental class learning results on MNIST-Fashion and CIFAR10 datasets

model showed similar performance on Fashion and CIFAR10 dataset. The α_{ideal} was set as 92% and 67.3% respectively. It has to be observed that EnsembleNet showed values greater than 0.92 for all Ω values on all the tested datasets.

4.3.2 Multi modal Incremental Learning

The experiment evaluates the ability of a model to learn multiple tasks sequentially. In this experiment, the model is first trained on one dataset and then it is expected to learn a new dataset. The first data on which the model has been trained is considered as the base dataset and the second dataset on which the model is trained is considered as the new dataset. MNIST Handwritten digits and MNIST Fashion datasets were used for the experiment. In one iteration, Handwritten digits dataset was trained as the base dataset and Fashion dataset was trained as the new dataset. In the second iteration, Fashion dataset was considered as the base dataset and Handwritten digits dataset was considerd as the new dataset. The model's behaviour has been benchmarked against standard Convolutional Neural Network Krizhevsky et al. (2012) and a standard Convolutional Neural Network that was allowed to rehearse the base dataset after being trained on the new dataset.

Model	Base Dataset	New Dataset	Ω_{new}	Ω_{base}	Ω_{all}	T_B	T_N	T_F
CNN	MNIST	Fashion	1.000	0.0013	0.5114	60s	60s	120s
CNN+rehearsal	MNIST	Fashion	0.758	0.9945	0.9314	120s	60s	180s
CNN	Fashion	MNIST	0.9937	0.2535	0.6787	60s	60s	120s
CNN+rehearsal	Fashion	MNIST	0.013	0.9262	0.521	120s	60s	180s
Ours	**MNIST**	**Fashion**	**0.907**	**0.9728**	**0.985**	**6s**	**6s**	**112s**
Ours	**Fashion**	**MNIST**	**0.9648**	**0.9272**	**0.9842**	**6s**	**6s**	**112s**

Table 4: Multi modal incremental learning results on MNIST-Fashion and MNIST-Handwritten digits datasets

The experimental results show that the proposed architecture has better capacity to retain the base knowledge and as well as learn new knowledge. Four variations of experiments were conducted on CNN and the CNN was allowed to rehearse the base knowledge in two experiments. Even after rehearsal, the standard CNN failed to show results that were comparable to our proposed architecture. The architecture showed an overall of higher performance on Ω_{all} and Ω_{new} metrics.

5 DISCUSSION

The working philosophy of EnsembleNet can be understood by understanding the Stability-Plasticity Dilemma Abraham & Robins (2005). The stability-plasticity dilemma basically states that, if a neural network's weights are disturbed easily when it is trained on new data, then the weights that are responsible for retention of old memories are distorted and the ability of the network to retain and recall the old data is lost. However, if the neural network's weights remain unchanged even when it is trained on new data, then it will fail to learn the new data. In the former case, the neural network can be considered as being too *plastic* and in the later case, the neural network can be considered as being too *stable*. The standard Multi-layer perceptron and Convolutional Neural networks are good examples of neural architectures that are too *plastic* in nature. The experimental results on Multi-modal experiments clearly show that CNNs are capable of learning new data very well, however their capacity for retaining base knowledge is low. Neural Networks that incorporate partial or total weight freezing perform poorly on learning new data as their architecture is too *stable* and crucial weight updations donot happen. Examples of architectures that utilize partial or total weight freezing are Fixed expansion layer network and Elastic weight consolidation networks.

The high performance of EnsembleNet can be explained by EnsembleNet's ability to keep a portion of it's network *stable* while leaving other portion of it's network to be *plastic*. The classifiers in the first level remain un-affected when the model is trained on new data. This protects the crucial weights in the network that are responsible for retaining old data. However, the weights in the newly added classifier and the final classifier are subjected to change which helps in the learning of new data. The lack of updation of weights in the level 1 makes EnsembleNet *stable* while rapid updation of weights in the final classifier and the new classifier makes it *plastic*. This inherent nature of remaining *stable* while having the flexibility to be *plastic* explains the high performance that has been observed in the experiments.

6 CONCLUSIONS

In this paper, we proposed a novel architecture which is capable of incrementally learning new object classes. We demonstrated that EnsembleNet is capable of retaining both recently learned data and newly learned data by only retraining a small portion of the entire network. In addition we showed that EnsembleNet can be trained much faster than traditional neural networks on a distributed system in an asynchronous manner. Future work will include exploration of ways to avoid storing of entire training data. Pseudorehearsal will be incorporated into the EnsembleNet architecture to make it more memory efficient. We are also planning to test the proposed architecture on datasets with higher number of classes like CIFAR100 and CUB200.

REFERENCES

Wickliffe C Abraham and Anthony Robins. Memory retention–the synaptic stability versus plasticity dilemma. *Trends in neurosciences*, 28(2):73–78, 2005.

Abdelhamid Bouchachia. Incremental learning by decomposition. In *2006 5th International Conference on Machine Learning and Applications (ICMLA'06)*, pp. 63–68. IEEE, 2006.

Jean-François Connolly, Eric Granger, and Robert Sabourin. Supervised incremental learning with the fuzzy artmap neural network. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 66–77. Springer, 2008.

Robert Coop, Aaron Mishtal, and Itamar Arel. Ensemble learning in fixed expansion layer networks for mitigating catastrophic forgetting. *IEEE transactions on neural networks and learning systems*, 24(10):1623–1634, 2013.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

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.

Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in neural information processing systems*, pp. 625–632, 1995.

Bogdan Gabrys and Andrzej Bargiela. General fuzzy min-max neural network for clustering and classification. *IEEE transactions on neural networks*, 11(3):769–783, 2000.

Alexander Gepperth and Cem Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934, 2016.

Haibo He, Sheng Chen, Kang Li, and Xin Xu. Incremental learning from stream data. *IEEE Transactions on Neural Networks*, 22(12):1901–1914, 2011.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

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*, 114(13):3521–3526, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

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.

Alexander K Seewald. How to make stacking better and faster while also taking care of an unknown weakness. In *Proceedings of the nineteenth international conference on machine learning*, pp. 554–561. Morgan Kaufmann Publishers Inc., 2002.

7 Supplementary material

7.1 Training time relationship between and ordinary network

In order to deduce the training time relationship between and *Ordinary network*, we need to make certain assumptions that are general and sensible in practice.

Assumptions or Control settings

- Training time of each of the sub-modules is proportional to the size of its corresponding input training data $T_i \propto |S_i|$
- Each of the sub-models have similar complexities and take same training time, i.e.
$$\frac{T_i}{T_j} = \frac{|S_i|}{|S_j|} \rightarrow T_i = \alpha * |S_i|$$
- Comparisons are drawn against a single neural network of multi class prediction type
- The training time of the final layer (say, T_f) of the is proportional to the number of input classes
- In a given scenario of comparison between Ordinary network and , we can assume $T_f = c$, some constant value.

In case of the parallel training scenario, the effective time required for training the non-final layer is,

$$T_b = max(T_1, \ldots, T_n)$$

.

Let $S_{max} \subseteq S$ be the partition corresponding to the sub-module that took the maximum time. Let $|S_{max} = \beta * |S|$.

The total time required to train the is therefore shown in (Equation 3).

$$T_{en} = T_b + T_f = T_{max} + c \tag{3}$$

The total time required to train the ordinary network is (Equation 4).

$$T_{ord} = \gamma * |S| \tag{4}$$

Now, the relationship between T_{ord} and T_{en} is shown in (Lemma 7.1).

Let us consider the ratio of training times of and Ordinary network,

$$\frac{T_{en}}{T_{ord}}$$

.

$$\begin{aligned} &\frac{T_{en}}{T_{ord}} \\ &= \frac{T_{max} + c}{T_{ord}} \\ &= \frac{\alpha * |S_{max}| + c}{\beta * |S|} \\ &= \frac{\alpha * \beta * |S_{max}| + c}{\gamma * |S|} \end{aligned}$$

When sub-module and the ordinary network are of similar architectures,

$$\gamma \approx \beta$$

. When all the sub-modules are similar,

$$\beta \approx \alpha$$

.

$$\begin{aligned}\frac{T_{en}}{T_{ord}} &= \frac{\alpha * \beta * |S_{max}| + c}{\gamma * |S|} \\ &= \frac{\alpha * \alpha * |S| + c}{\alpha * |S|}\end{aligned}$$

We require that,

$$\frac{T_{en}}{T_{ord}} < 1$$

, which is implied by (Equation 5).

$$\begin{aligned}\longrightarrow \frac{|S| * \alpha^2 + c}{\alpha * |S|} &< 1 \\ \longrightarrow = \alpha + \frac{c}{|S| * \alpha} &< 1 \\ \longrightarrow = \alpha^2 - \alpha + \frac{c}{|S|} &< 0\end{aligned}$$

$$\alpha = \frac{1 \pm \sqrt{1 - 4 * \frac{c}{|S|}}}{2} \tag{5}$$

If sub-module and ordinary are of similar architectures, then $T_{en} \leq T_{ord}$

Upon enforcing (Equation refeqn:enet-ord-axiom1) and due to (Lemmas 7.1.1,7.1.2,7.1.3), we can conclude that $T_{en} \leq T_{ord}$.

7.1.1 REAL VALUED α

We required that, α is a real value, which means the quantity under square root to be positive (Equation 6).

When the number of training points in the data set are at least 4 times larger than the number of classes, then each sub-module is non-trivially trainable.

Refer to (Equation 5) and consider the quantity inside the square root.

$$\begin{aligned}1 - 4 * \frac{c}{|S|} &\geq 0 \\ \longrightarrow 1 &\geq 4 * \frac{c}{|S|} \\ \longrightarrow |S| &\geq 4 * c\end{aligned}$$

$$|S| \geq 4 * c \tag{6}$$

7.1.2 POSITIVE VALUED α

We require,

$$\alpha \geq 0$$

.

The condition $\alpha \geq 0$ is satisfied all valid values of number of classes and sizes of training set of points.

Refer to (Equation 5) and consider the minimum value of α.

$$\begin{aligned} \alpha^2 - \alpha + \frac{c}{|S|} &< 0 \\ \longrightarrow \alpha \in [\frac{1 - \sqrt{1 - 4 * \frac{c}{|S|}}}{2}, \frac{1 \pm \sqrt{1 + 4 * \frac{c}{|S|}}}{2}] \\ \alpha = \frac{1 - \sqrt{1 - 4 * \frac{c}{|S|}}}{2} &\geq 0 \end{aligned}$$

Consider $\frac{1 - \sqrt{1 - 4 * \frac{c}{|S|}}}{2} \geq 0$,

$$\begin{aligned} \frac{1 - \sqrt{1 - 4 * \frac{c}{|S|}}}{2} &\geq 0 \\ \longrightarrow \sqrt{1 - 4 * \frac{c}{|S|}} &\leq 1 \\ \longrightarrow 1 - 4 * \frac{c}{|S|} &\leq 1 \\ \longrightarrow -4 * \frac{c}{|S|} &\leq 0 \end{aligned}$$

.

Therefore, $(\forall c, |S|) : \alpha \geq 0$.

7.1.3 HANDLING THE UPPER BOUND, $\alpha \leq 1$

The condition $\alpha \leq 1$ is satisfied all valid values of number of classes and sizes of training set of points.

Refer to (Equation 5) and consider the maximum value of α.

We required that,

$$\alpha \leq 1$$

.

$$\begin{aligned} \alpha &\leq 1 \\ \longrightarrow \frac{1 + \sqrt{1 - 4 * \frac{c}{|S|}}}{2} &\leq 1 \\ \longrightarrow 1 + \sqrt{1 - 4 * \frac{c}{|S|}} &\leq 2 \\ \longrightarrow \sqrt{1 - 4 * \frac{c}{|S|}} \leq 1 \longrightarrow -4 * \frac{c}{|S|} &\leq 0 \end{aligned}$$

This is again true, $(\forall c, |S|) : \alpha \leq 1$.

7.2 Estimating relationship between training and testing error rates

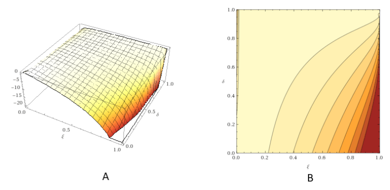


Figure 5: Relationship between sub-module error ξ and overall error δ for $n = 10$

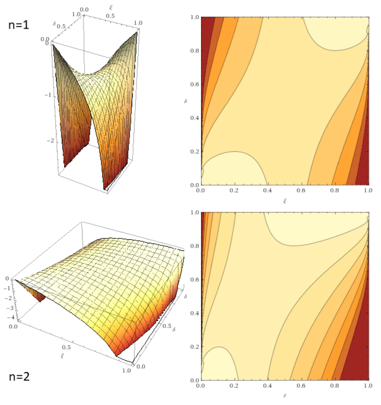


Figure 6: Relationship between sub-module error ξ and overall error δ for $n = 1, 2$