Training a Lifelong Learning Neural Network to apply on the Edge

1st Hitesh Vaidya
Dept. Of Data Science
Rochester Institute of Technology
Rochester NY, USA
hv8322@rit.edu

2nd Manali Dangarikar
Dept. Of Computer Engineering
Rochester Institute of Technology
Rochester NY, USA
md2591@rit.edu

3rd Prakhar Soni
Dept. Of Computer Engineering
Rochester Institute of Technology
Rochester NY, USA
ps3263@rit.edu

4th Shaffatul Islam
Dept. Of Microelectronics Engineering
Rochester Institute of Technology
Rochester NY, USA
si9836@rit.edu

***Abstract*—A lifelong learning neural network, also known as an incremental learning or continual learning neural network, is designed to learn and adapt over time by continuously incorporating new information and tasks without forgetting previously learned knowledge. Traditional neural networks are typically trained on fixed data sets, where the network is trained on a specific task and then its weights are frozen. If the network is to be trained on a new task, it usually requires retraining from scratch, and the retraining may result in catastrophic forgetting, where the network loses the ability to perform previously learned tasks. A lifelong learning neural network aims to overcome these limitations by allowing the network to learn and accumulate knowledge incrementally. This paper presents an approach of using Self-Organising Maps (SOMs), which is an unsupervised learning algorithm, for implementing lifelong learning on the MNIST data set, by incrementally training the neural network, one class at a time, and deploying the model on the Akida hardware for inference.**

***Index Terms*—lifelong learning, self-organising maps, incremental training, akida**

I. Introduction

An intelligent system that can continually learn and adapt to new information is a crucial requirement for many real-world applications. One of the major challenges in developing such systems is catastrophic forgetting, which refers to the phenomenon where a system, such as one based on artificial neural networks (ANNs), struggles to remember previously acquired knowledge when learning from new samples. This issue becomes even more challenging when there are no explicit task boundaries provided with the input samples. Catastrophic forgetting has been extensively studied in the context of ANNs. However, there is relatively less research investigating the issue of forgetting in architectures such as the self-organizing map (SOM). SOMs have been widely used in tasks such as clustering and dimensionality reduction. Exploring the challenges of catastrophic forgetting within SOM architectures is an interesting and under explored research area. Self-Organizing Maps (SOMs) are a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and therefore are a method to do dimensionality reduction. They are also called Kohonen maps, after their inventor, Teuvo Kohonen [1]. The rest of the paper is structured as follows: section II provides a detailed overview of self-organizing maps, its advantages and the architecture of the network; section III describes the proposed approach in detail, training and deployment methodologies; section IV presents the experimental results and evaluation of the proposed approach; finally, section V concludes the paper and highlights future research directions.

II. Self-Organising Maps (SOMs)

SOMs are inspired by the way our brains organize sensory input through neurons. Each area of the cerebral cortex in our brain is responsible for specific activities, and a sensory input such as vision, hearing, smell, and taste is mapped to neurons in a corresponding cortex area via synapses in a self-organizing manner. SOM is designed to mimic these brain mechanisms by training through a competitive neural network, which is a single-layer feed-forward network where neurons with similar output are placed in proximity to each other [2]. Although the SOM algorithm is relatively simple, it can be confusing at first and may pose difficulties when trying to apply it in practice due to its multiple perspectives. It can be seen as analogous to Principal Component Analysis (PCA) for dimensionality reduction and visualization, as well as a form of manifold learning for non-linear dimensionality reduction. The algorithm maps high-dimensional input data onto a lower-dimensional latent space, usually a 2D grid, while preserving the original input space's topology. This mapping can also be used for projecting new data points and identifying their corresponding cluster on the map. Furthermore, SOMs are known for their robustness to noise, which makes them suitable for real-world applications.

A. Architecture and Learning Algorithm

The SOM consists of a grid of units or neurons. Each neuron has a pre-determined position. The SOM works by iteratively adjusting the weights of these neurons in the grid to resemble data points from the input space.

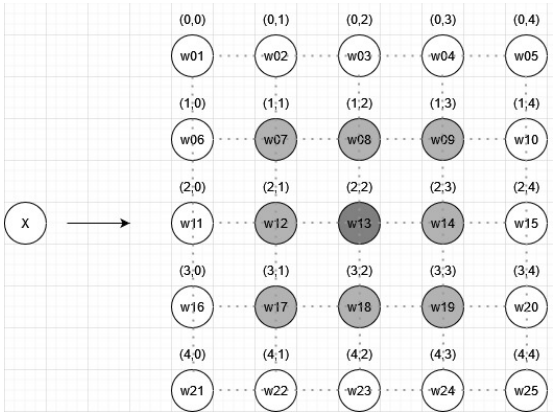

Fig. 1. Training of the SOM. Here, the neuron at position (2,2) is selected as the best matching unit for input image X.

The training process of a Self-Organizing Map (SOM) involves several steps that enable the neural network to learn from input data in a competitive and cooperative manner. Firstly, the weights of neurons are initialized randomly. Then, for each input sample, the neuron with weights closest to the input sample is identified as the Best Matching Unit (BMU). A neighborhood function is defined around the BMU. The extent of this function is typically large at the beginning and shrinks over time. Further, the weights of neurons within the neighborhood are adjusted to make them more similar to the input sample. The adjustment is proportional to the distance of the neuron from the BMU. The weights of the neurons which are closer to the BMU are adjusted more. This process of identifying BMU, defining the neighborhood, and adjusting weights is repeated for a set number of iterations until the map's topology starts to stabilize. This method allows SOMs to efficiently handle high-dimensional data and identify underlying patterns in the input space.

III. PROPOSED METHOD

A. General Workflow

The workflow of our implementation involves several steps to train and deploy a Self-Organizing Map (SOM) model for accurate predictions. The first step is offline training, where the model is trained on TensorFlow using training data. During this step, the model learns and adjusts its parameters to minimize the error in its predictions. The next step is predicting the label for every unit in the SOM, which is also done offline. To do this, each unit is passed through the trained model, and the predicted label is recorded. Once the model is trained, and each SOM unit has a predicted label, the model can be deployed on the Akida Neural Processor for inference. This is the process of making predictions with the trained model. Finally, the accuracy of the model is calculated by checking which SOM unit a given test sample matches with. If the predicted label of that matched unit is the same as the label of the test sample, the accuracy count is incremented by 1. Otherwise, the count remains the same. By repeating this process for all test samples, the overall accuracy of the model can be calculated.

B. Training Algorithm

We developed an algorithm inspired by the working of a Convolutional Neural Network (CNN) to train the SOM, bypassing the traditional training algorithm. To avoid catastrophic forgetting in incremental lifelong learning, we used a custom loss function that utilizes the running variance of individual units. We performed loss calculation like a convolution layer. To calculate the Best Matching Unit for an input image, we used a min pooling layer. The weight update was performed using a Gaussian update mask. This training process achieved a significant speedup, reducing the execution time from 40 minutes to 15 minutes.

Algorithm 1 SOM Training Process

Require: Dataset X, SOM dimensions $(grid_x, grid_y)$, number of iterations T, initial learning rate lr, decay function $decay$

Ensure: Trained SOM with updated weights W

Initialize a SOM grid with dimensions $(grid_x, grid_y)$ with random weights W
Initialize the initial radius σ as $\max(grid_x, grid_y)/2$
for $t = 0$ to T **do**
  for each sample x in X **do**
    Calculate the Euclidean distance between x and each neuron's weight vector in W
    Determine the Best Matching Unit (BMU), the neuron with the smallest distance to x
    Calculate the radius of the neighborhood around the BMU, σ_t, using the decay function
    Calculate the learning rate for this iteration, lr_{nrt}, using the decay function
    for each neuron j in the grid **do**
      Calculate the influence of the BMU on neuron j, $h(j, BMU, t)$, based on its distance from the BMU and σ_t
      Update the weights of neuron j as follows: $w_j = w_j + lr_{nrt} * h(j, BMU, t) * (x - w_j)$
    end for
  end for
end for=0

C. Label Prediction for SOM

In order to test the accuracy of the SOM it is needed to predict which class does every trained unit in it look like. In other words, we must obtain the predicted label of every unit in the SOM. To obtain this predicted label, we use a statistic called *class count*. It stores the count of how many number of

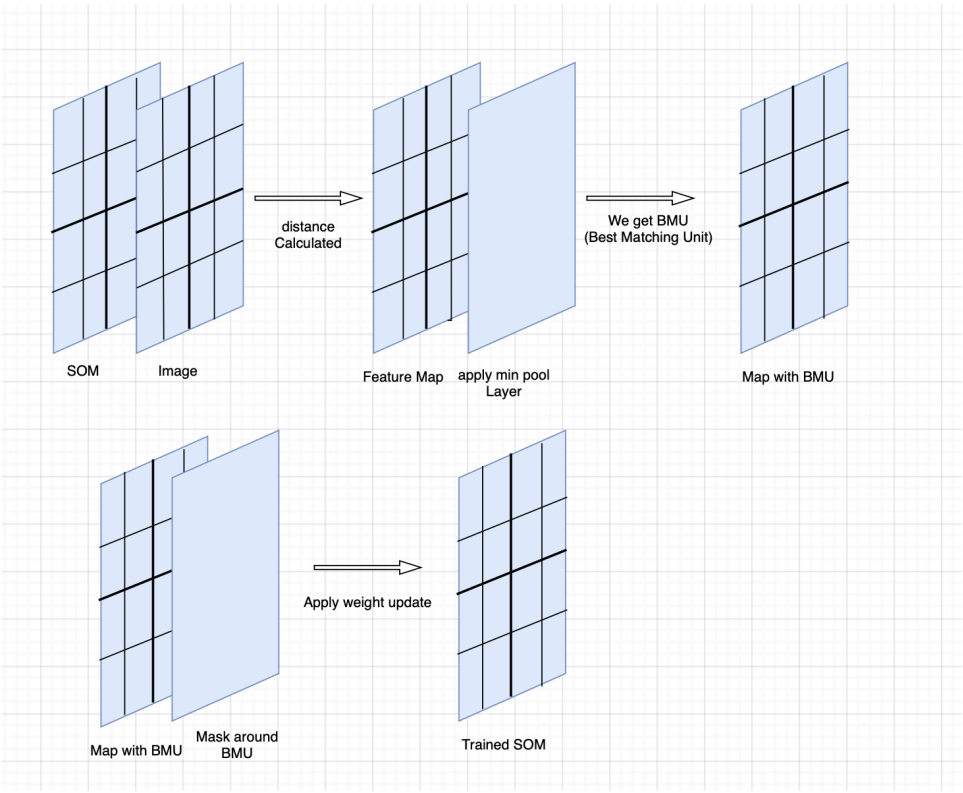

Fig. 2. SOM Training Process

times a unit was selected as the Best Matching Unit (BMU) for a particular task or class. Since our dataset has 10 classes, the *class count* for every unit is a vector of length 10. At every index, we store BMU count for the corresponding class. At train time, the BMU counter at the index of the current input class is incremented if a unit gets selected as the BMU. At test time, the class for which the BMU count is maximum is assigned as the predicted label for a unit.

D. Inference on Akida

After the model has been trained and each BMU has a predicted class, we deploy our SOM on the Akida processor for inference. For the inference, we need to identify which unit best matches a test input sample. To determine the BMU for a test input sample we use cosine similarity measure. The unit that has the maximum cosine similarity with the input is selected as the BMU for it. Then, obtain the predicted class value of the BMU and if it matches with the expected class label of the test input sample then the accuracy measure is incremented by 1 and not otherwise.

This entire process of calculating cosine similarity of every unit followed by $\arg max$ operation to get the BMU and then comparison of predicted vs. expected label is performed by creating several layers in Tensorflow. Akida requires a computational graph for this layer pipeline which is generated using the functional API of Tensorflow. We implemented several methods to generate a working computational graph that could perform inference on Akida. Regardless of the method of execution, we need the SOM matrix and the *class count* vector of every unit along with the test input sample to calculate the accuracy at test time. We can term th

1) Nesting Layer objects with multiple inputs: We created Layer objects of Tensorflow for every operation mentioned above. The data flows through all the layers to get the predicted label as the final output and a model is created out of all the layer objects. However, we also had to pass the SOM matrix and the *class count* as two additional inputs along with the test sample. However, the Akida does not work with a model that accepts multiple inputs. Therefore, we were unable to build a model of such kind to measure the accuracy of the SOM.

2) Using model statistics as hyperparameters: Since we could not use the approach in the above section, we created another model that would use the SOM matrix and the *class count* as model hyperparameters. We even tried the method of concatenating these values along with the input but it is computationally inefficient to manage the dimensions of such an input which led to the failure of this approach.

3) Creating custom Layer: Next, we created a new custom layer class that inherits the Layer class of the Tensorflow. The class accepts the SOM matrix and the *class count* as the class fields and `call()` function accepts the input sample to perform all the operations just in one layer. Such a layer would accept the input sample and output the predicted label of the corresponding matched unit in the SOM. A model compiled with such a layer was incompatible for Akida because it does not work on custom layer classes.

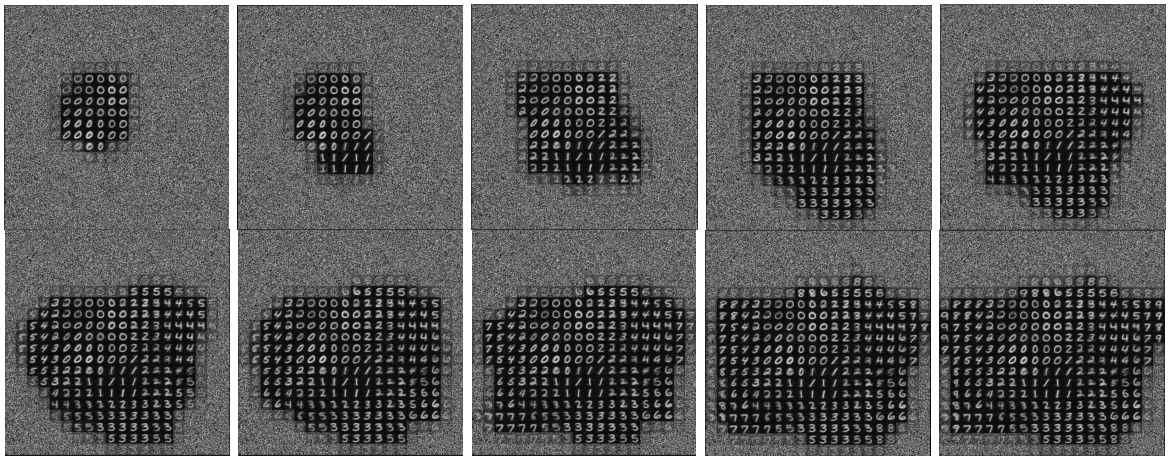

Fig. 3. SOM visualization after every class

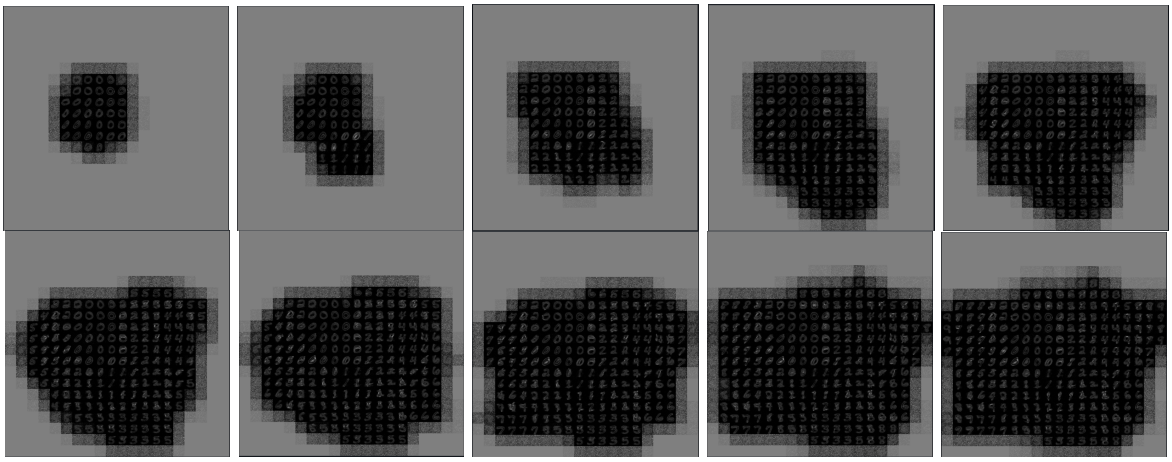

Fig. 4. Running variances after every class

4) Convolution operation for Cosine Similarity: Finally, we tried a calculate the cosine similarity between SOM units and the input sample using a convolution layer (*CONV2D()*) of the Tensorflow. We split the units of the SOM of dimesions (28×28) and initialize the kernels of the convolution layer with them. Then, we calculated the cosine similarity between every kernel and the input sample followed by getting the index of the kernel with maximum cosine similarity value and then obtaining the predicted label of the respective kernel. The *cnn2snn* library raised runtime exception in such an operation for multiplication operation.

IV. Dataset

We experiment our model on MNIST dataset which is split in a class incremental fashion. The MNIST images are of dimensions $[28 \times 28]$ and have labels in the range $[0 - 9]$. An incremental dataset is where data is split into different tasks and each task may have multiple classes. In our case, the task size was 1 which means, we had 10 different sets of images. Each set had all the images belonging to the corresponding class number. Further, we normalized the images to bring their pixel values in the range of $[0 - 1]$

V. Results

We created an SOM with $[20 \times 20]$ units where the dimensions of each unit were $[28 \times 28]$. Every unit was initialized using a random normal distribution and had initial radius and learning rate values of 1.5 and 0.5 respectively. We trained this model class incrementally on the classes of MNIST. During training, no task or class boundaries were passed as heuristic to the model. As seen in figure 3, when we move gradually through classes 0 till 9 there is a good line of separability amongst the trained clusters. The gaussian noise seen in figure 3 indicates an empty or untrained image. The very last figure shows that there is still room for more incoming sample from new classes. This shows that we can create a static or non-expandable SOM to encapsulate a compressed representation of a class incremental input space and still have space for more training. By assigning independent radius and learning rate values to every unit, we promoted more competition amongst them to speed up the training process and the results support our hypothesis. Figure 4 show the corresponding running variance values of every unit in the SOM that are used for calculating distance values between the input samples and the

SOM units at train time.

VI. Conclusion and Future work

We developed a continually learning version of Self-Organizing Maps that has applications in edge computing. The Akida platform provides a real life application to test our model and prove our hypothesis that SOMs could be used as effective lifelong learning model to preserve compressed representation of incremental input space. We can use the values shown in figure 3 as mean and the running variance values as shown in figure 4 to generate new samples from each trained unit. Thus, SOMs can also be used as effective generative models that are not as computationally expensive as others like the variational autoencoder. They can contribute greatly to generative replay in lifelong learning.

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps", Biological Cybernetics, vol. 43, no. 1, pp. 59–69, 1982, doi: 10.1007/bf00337288.

[2] https://towardsdatascience.com/understanding-self-organising-map-neural-network-with-python-code-7a77f501e985

[3] T. Kohonen, "Essentials of the self-organizing map," Neural Netw., vol. 37, pp. 52–65, Jan. 2013.

[4] Hitesh Vaidya and Travis Desell and Alexander Ororbia, "Reducing Catastrophic Forgetting in Self Organizing Maps with Internally-Induced Generative Replay",

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.