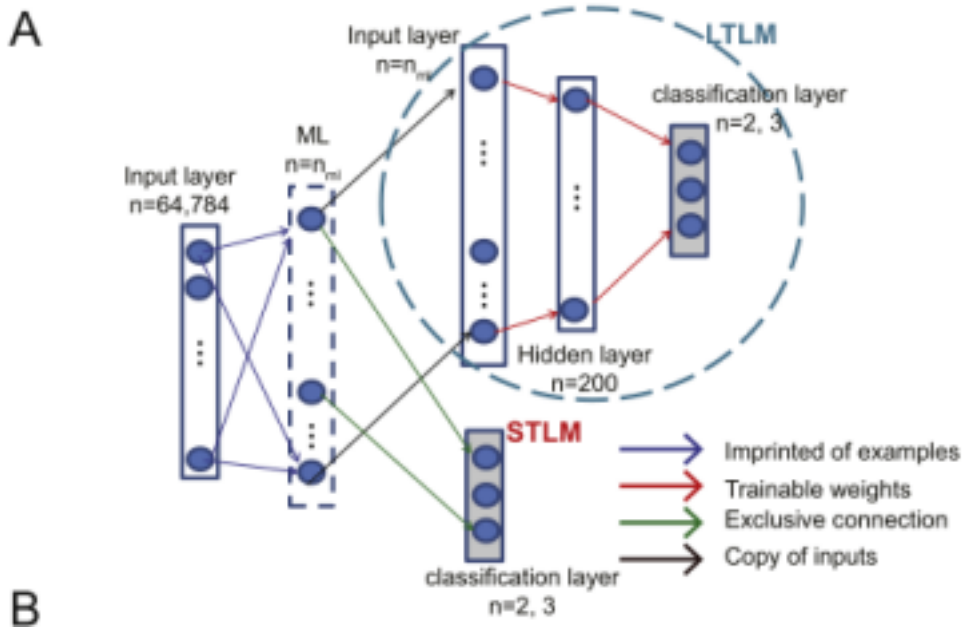
DynMat – A network that can learn after learning

Introduction

Most traditional neural network suffer from an inherent problem called Catastrophic Interference. This is the problem of forgetting the old learning when the model is trained on a different dataset. For example, if a machine is trained with digit ‘0’ of MNIST dataset and then if the same trained model is trained using digit ‘1’ of MNIST Dataset, then the traditional model would forget the learning of digit ‘0’. DynMat solves this problem of Catastrophic Interference. The problem with the traditional model is that they keep overlapping representation of digit ‘0’ and ‘1’ i.e. the network uses some weights to represent both digit ‘0’ and ‘1’. DynMat keeps sparse representation of the digit ‘0’ and ‘1’.

Architecture



Input Layer:

The MNIST database consists of images of numerical digits from

‘0’ to ‘9’. Each image has 784 pixels. Hence, there are 784 functional units in the input layer.
Images are normalised before they are fed into input layer.

The DynMat network consists of three parts:

a. Matching Layer(ML) : It is a dynamic layer i.e. the number of functional units in the matching layer will change during training. The number of neurons in this layer is the number of stored examples in the network. One neuron of the matching layer is connected to all the functional units of the input layer. The weight of the connection between one ML neuron and one input layer functional unit will contain the corresponding pixel value of the normalised image which is stored in that ML neuron. For e.g., the connection between 22^{nd}functional unit of input layer and 5^{th} functional unit of ML layer will contain the 22^{nd} pixel value of the normalised image that is stored in 5^{th} ML neuron. In this way each ML neuron stores an image in the network.

b. Short Term Learning Module(STLM): The number of neurons is the number of classes that are present in the classification. Each neuron will represent one class that is present in the dataset. The input of this layer comes from functional units of ML layer. The STLM neuron corresponding to class ‘a’ will be connected to all the functional units of the ML layer that stores the examples of class ‘a’. The weight of the connections between the neurons of STLM and ML layer is 1.

c. Long Term Learning Module(LTLM) : The input of the LTLM layer is the output of the ML layer. This module can contain any traditional network that can be used to train for classification problem.

Training:

A hyperparameter (θ) is decided as a threshold value to decide whether there is a need to add the input image in the network or not. To do this, cosine similarity is calculated between all the images that are stored in the network(in ML layer) and the input image. The following formula is used:

$$h_{i,k}^{ML} = \sum_j w_{ij}^{ML} \frac{x_j^k}{||\overrightarrow{x_k}||}, where \ \ w_{ij}^{ML} = \frac{x_j^i}{\overrightarrow{x_i}},$$

where,

h_{ik}^{ML} represents input to ML neuron m_i induced by k^{th} example
w_{ij}^{ML} represents the connection from node l_i to ML neuron m_i
X_j^k is the j^{th} component of k^{th} example
$|| x_k||$ represents the norm of k^{th} image

h_{ik}^{ML} is the cosine similarity of the k^{th} input image with the stored example in the i^{th} ML neuron.

If the value of cosine similarity is less than θ for all the images present in the network, then the normalised input image is stored in the network. To store the image in the network, a new functional

unit is added to the mathcing layer. This neuron is connected to all the neurons of the input layer. The weight of each connection is the corresponding pixel value in the normalised image. For e.g., the connection link between a new neuron and the 22ndinput layer functional unit will contain the 22nd pixel value of the input image. Also the new neuron in ML is connected to the corresponding STLM neuron, according to the class of the input image. Also there will be a new neuron added in the LTLM layer and this new LTLM neuron will be connected to the ML neuron.

LTLM:

After STLM and ML modules are trained, the ML layer stores a concise version of the dataset.The LTLM layer requires the whole dataset that was used to train STLM and ML modules for training. For a given input image, cosine similarity is calculated with every stored example in ML layer. The vector that contains the cosine similarity between the input image and all the stored example is fed as an input to the LTLM layer. The LTLM layer is trained on this input data.

Prediction:

For a given input image, the cosine similarity between the input image and all the images stored in the ML layer is calculated. The output of the ML layer is fed into STLM layer. Each STLM functional unit combines the cosine similarity of the input image
with all the stored examples of a given class. It does it using the

following formula:

$$h_{i,k}^{STLM} = \sum_{j} w_{ij}^{STLM} g(h_{j,k}^{ML}), \ where$$

$$w_{ij}^{STLM} = \begin{cases} 0, & i \neq c \\ 1, & i = c \end{cases}, \ g(x) = e^{-\left(\frac{x-1}{0.1}\right)^2}$$

h_{ik}^{STLM} represents input to STLM neuron s_i connected by k^{th} ML functional unit

w_{ij}^{STLM} represents the connection from ML neuron m_j to STLM neuron s_i
c represents class id
g(x) is the non-linear function used to combine the cosine similarities of the input image with all the images of a given class.

The predicted class of the input image is the class for which the output of the STLM neuron is maximum.

The output of the Matching layer is also fed into the LTLM layer and hence the class of the input image is also predicted using LTLM layer.

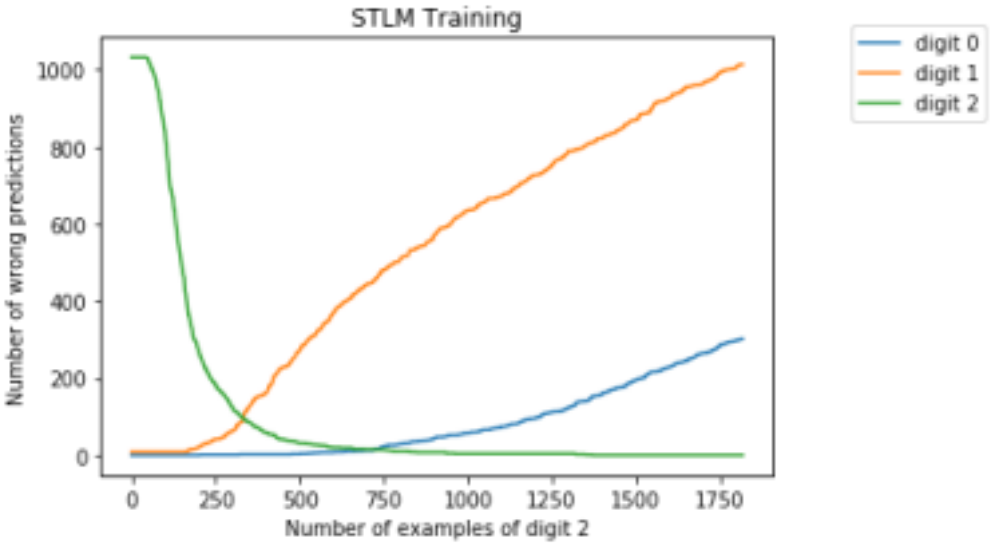
Novelty

The LTLM module takes the whole dataset and computes the class label as described above. The STLM module also predicts the output of the given input image. The LTLM has the power of generalisation since the model is trained over the whole dataset. STLM has the power of sparse representation due to which it

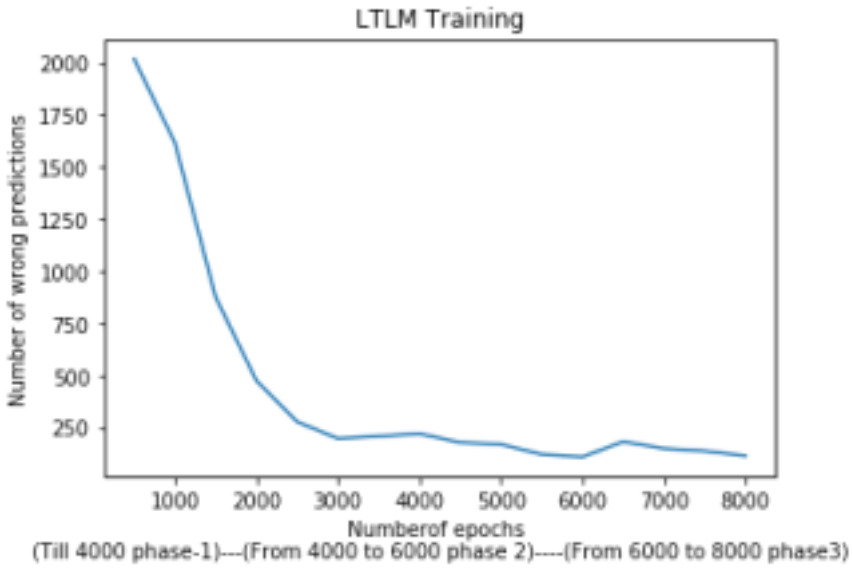
can
decrease the effect of catastrophic interference. The author of this paper has created such architecture to compare the model of STLM and LTLM in his paper. We have tried to combine the power of both this models and investigate whether it will give a better result. LTLM takes huge time to train and STLM takes much less time to train. So, we want to investigate whether we can combine the power of STLM and suboptimal LTLM so that the performance of the combination is better than either STLM or suboptimal LTLM. This is important because the time required to train an optimal LTLM model is huge.

Current Investigation:

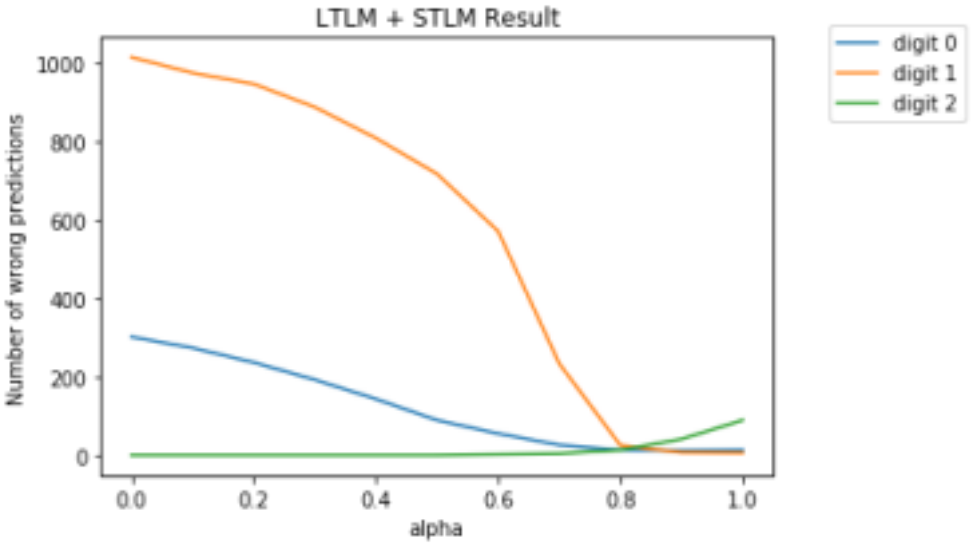
MNIST Dataset is used to train this model. First the STLM model was trained with ‘0’ digit dataset and then with ‘1’ digit dataset. Then the STLM model was trainedwith ‘2’ digit dataset. Hence, we obtained the ML layer for digit ‘0’, ‘1’ and ‘2’ digit dataset. The number of functional units in ML layer was 2580. After every addition of new example, the STLM module was tested. Following is the graph obtained after the testing of STLM module:



After this the combined the dataset of digits ‘0’, ‘1’ and ‘2’ was taken from MNIST dataset. The dataset was shuffled randomly and then given as input to the model for the training of LTLM layer. The LTLM layer had one hidden layer containing 300 functional units. The activation function for the hidden layer was sigmoid function. The network was trained using Adam optimizer and for the model was trained in three phases: the learning rate for first, second and third phase was 10^{-4}, 10^{-5} and 10^{-5} respectively. The first phase completed in 4000 epochs and both second and third phase were completed in 2000 epochs. After the completion of training for LTLM module, the following graph was obtained



Then we investigated the combination of STLM and LTLM on the datasets of digits ‘0’, ‘1’ and ‘2’ obtained from test dataset of MNIST Dataset.



The combination of STLM and LTLM was done by: α*LTLM + (1-α)*STLM. It was observed that the model gave best performance at α =0.8 .

Next investigation was to increase the number of digits to {0,1,2,3} but the problem was that it needed higher number of

neurons to train. It led to even more time to train LTLM. Since there was no availability of GPU on my laptop, so I was not able to train the model with this inputs.

Next experiment was to check the result on a different set of numbers – {0,2,3}. The the same architecture of LTLM was not able to train the dataset of {0,2,3}, so currently I am building the architecture of the network to train {0,2,3}.

Challenges

One challenge for the model is that the input to the LTLM is the cosine similarities of the stored examples with the input image. Since the input of the LTLM is from [0,1], hence it takes a long time to train the LTLM network.