Plasticity in Deep Neural Networks

Uddeshya Upadhyay[1] and Suyash Bagad[2]

***Abstract*— Plasticity is essential component in many biological neural networks, however many state of the art deep learning methods do not incorporate plasticity. We incorporate one form of plasticity in conventional deep learning network which can be trained using back-propagation, propose methodology on how to train such networks and study the effect of plasticity in different settings.**

I. INTRODUCTION

Spiking Neural Networks are proposed as an attempt to mimic the biological neural networks in a manner which is much more closer (with respect to hardware requirements and computations) than prominent deep neural networks based on the back-propagation which depends upon the differentiable nature of the neural networks.

In this project, we have introduced plasticity in a simple fully-connected network of neurons. Essentially, incorporating plasticity has two implications for a network, first, we are trying to get as close as possible to the biological neural network, and second, how this improves performance with low memory footprint as compared to the conventional Artificial Neural networks. The fundamental difference of the Spiking neural network with plasticity and the conventional neural networks is that the synaptic weights have two components, fixed and plastic, the latter being the product of learning rate and the Hebbian trace. The plastic component is a purely lifetime/episodic quantity, essential in learning of patterns over a period of time.

II. METHODS

A. Episodic Training

We introduce plasticity in our networks following the work by Miconi *et al* [1] which calculates the Hebbian between the pre-neuron and ost neuron activity and uses it to change the strength of the connection between two neurons as described by the formula 1.

$$x_j(t) = \sigma\Big(\sum_{i \in inputs} [w_{i,j}x_i(t-1) + \alpha_{i,j}Hebb_{i,j}(t)x_i(t-1)]\Big) \quad (1)$$

$$Hebb_{i,j}(t+1) = \eta x_i(t-1)x_j(t) + (1-\eta)Hebb_{i,j}(t) \quad (2)$$

The above formulation assumes a time dependence in the learning, while this may be obvious in some sequential tasks, such a dependency is not obvious in many tasks such as classification of objects etc. One way to introduce the temporal aspect in the training is by batching together the input data in appropriate way. For reconstruction task we presented our network with random 961 bit vector, each vector was shown for t_{ns} followed by a zero 961 bit vector for t_{zs}, at the end a randomly chosen vector from the batch is corrupted by setting some bits to -1 and passed through the network. A global loss (mean square error) is calculated between the output for the corrupted vector and the ground truth which is then back-propagated to learn the parameters.

We also study the effect of the network with and without plasticity in the presence of noisy input. We used [MNIST] and [FMNIST] dataset for the same. We designed an autoencoder[cite] where we introduced plastic layers at the end of encoder and decoder in one case and no plastic layers in other case. Each of the image in batch is shown t_{ns} times followed by zero image t_{zs} times. The last image is shown with some added noise and the global loss is computed between the final output for each image and the ground truth image.

We summarize our experiments and results in next section.

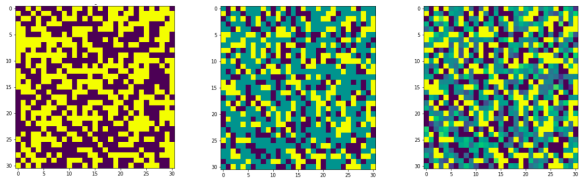
III. EXPERIMENTS AND RESULTS

Incorporating plasticity in the network improves memory of the network, this is verified by performing the following experiments.

A. Reconstruction

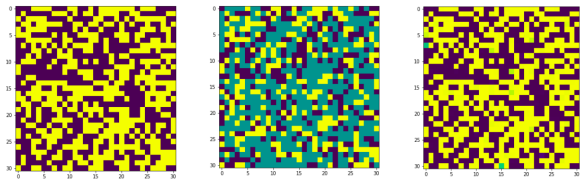
We have used the primitive Hebbian trace update rule to start with. We first consider the problem of reconstructing the given input to a desired output with minimum loss. Herein, each input is a vector of values in $\{-1, +1\}$, of size, say in_dim. This input vector is degraded by randomly changing some $-1s$ and $+1s$ to 0. Now, this degraded vector is passed as an input to a fully-connected layer of neurons, each of which following the plasticity rule as defined above. The method used here is: multiple copies of same input vector, with some intermediate zero vectors between any two sets of copies of input vectors are fed as input to the layer. These copies of input vectors are fed to the layer at different episodes (time instances). Thus, the plastic component of synaptic weights gets updated as the Hebbian rule and with time, we start getting output with lesser error. The results are shown in Figures 1 and 2.

[1]Dept. of Computer Science and Engineering, IIT-Bombay
[2]Dept. of Electrical Engineering, IIT-Bombay



(a) Target (b) Input (c) Output

Fig. 1: Reconstruction of a given input pattern after 10 epochs, Mean loss = 0.02393



(a) Target (b) Input (c) Output

Fig. 2: Reconstruction of a given input pattern after 930 epochs, Mean loss = 0.00010

B. Autoencoding in presence of noise

We present our network an episodic batch of images, where the task of the network is to get back the input (i.e encode and decode), at the test time we present a noisy input to the network from the test set of images (test set images were never presented to the network, noise introduced by adding Gaussian fluctuations). Results for MNIST and FMNIST dataset are presented in Figures **??**, **??** and **??**, **??** respectively.

We used convolutional networks as our encoders and decoders. Encoder consists of stack of convolutional layers batch-norm layers in an alternating fashion. Decoder has a structure which is a mirror image of encoder where convolution layers are replace with convolutional transposed layers in order to perform up-sampling. Figure [**?**] shows the schematic of our network and how we have accommodated plasticity in the network.

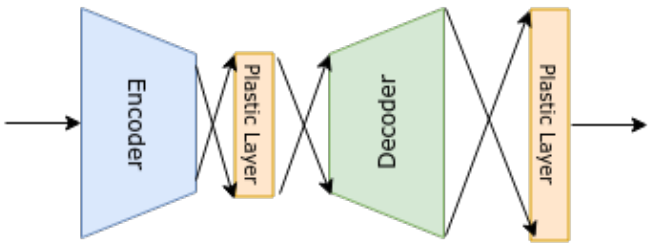
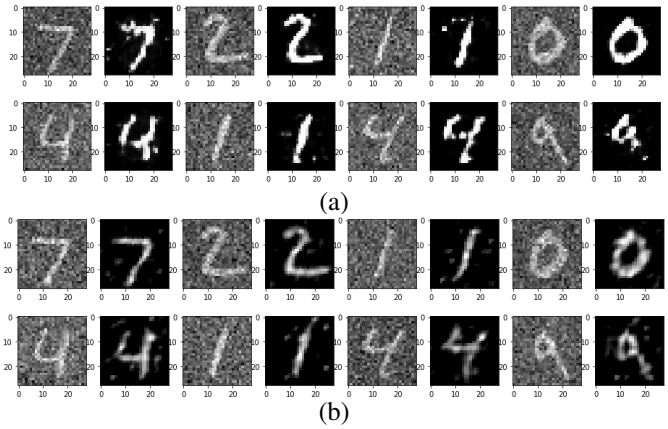


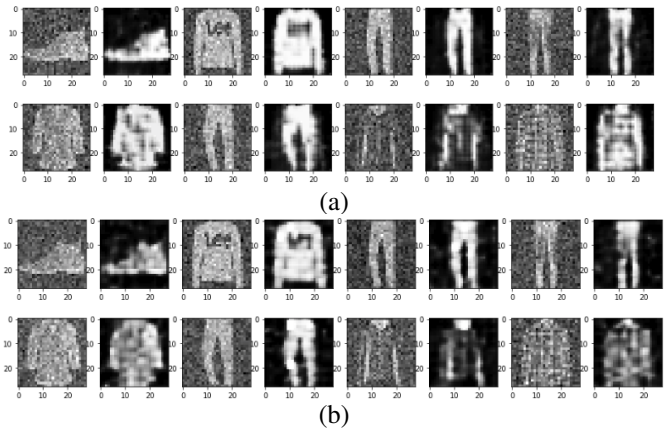
Fig. 3: Convolutional autoencoder network. Plastic layers included after encoder and decoder



(a)

(b)

Fig. 4: (a) Denoising using an autoencoder where plastic layers are included after encoder as well as decoder. (b) Denoising using similar DNN based Autoencoder-decoder network on MNIST dataset



(a)

(b)

Fig. 5: (a) Denoising using an autoencoder where plastic layers are included after encoder as well as decoder. (b) Denoising using similar DNN based Autoencoder-decoder network on FMNIST dataset

C. Classification performance plastic vs non-plastic

Hebbian plasticity improves the memory of the network, this was shown to be true in some of the experiments proposed in the network. Author makes claim that this effect is very generalized and plasticity should improve the performance in the scenario where memory of the network is crucial. To see if this type of plasticity can help in the classification task we design and study the performance of the networks as follows.

The task is to classify MNIST dataset. The test set contains 10,000 images of handwritten digits and training set consists of 60,000 images with labels. Since proposed plasticity does not perform well in batches (hence we need to feed the network one example at a time) this makes running time very slow, so instead of using all the 60,000 images we use less than 25,000 images and study the performance. This also

help us to observe the performance with less amount of data (and if plasticity can perform well in such cases).

We design a convolution network which gives result comparable to state of the art for mnist classification (when trained using all the training images). We make two version of the network one with the standard softmax layer at the end (we call this model non-plastic model) and one with the last layer replaced with a plastic linear layer (with softmax activation function) with recurrent connection (we call this model plastic model).

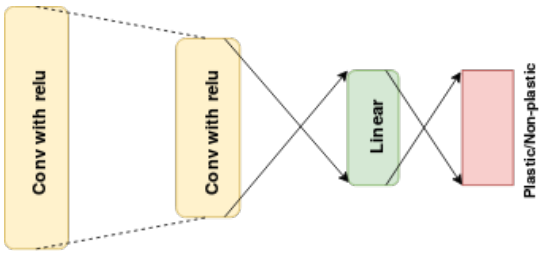
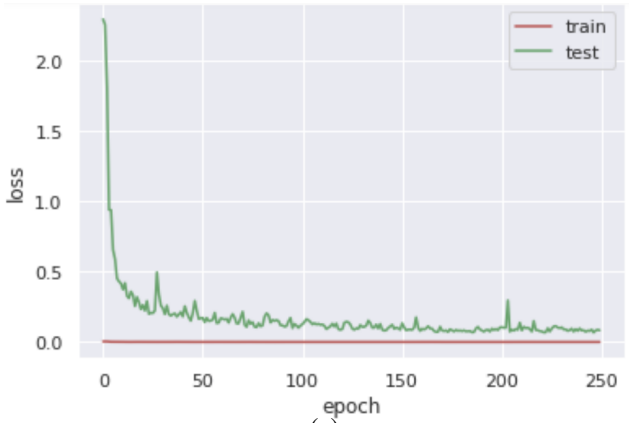


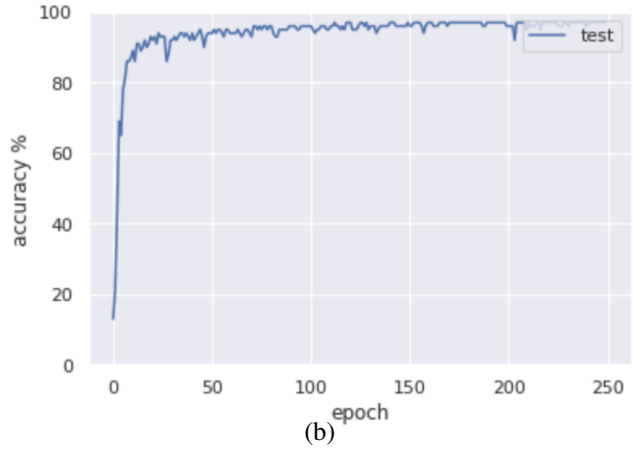
Fig. 6: Convolutional autoencoder network. Plastic layers included after encoder and decoder

We feed the network examples over 250 epochs, in each epoch we sample 100 images randomly from the training set, we feed one such sample at a time to the network and use the output to calculate loss and back-propagate the gradient of the error. After every epoch we test the performance of the network on entire test set (10,000 images). We plot the test set accuracy of the network after every epoch for both plastic and non-plastic version.

Our results show that introducing this kind of plasticity in the network usually lead to more unstable training. Also we did not see any improvements in the performance by introducing plasticity. Test set accuracy for both the network start saturating at higher values and best test set accuracy for non-plastic version is higher than corresponding plastic version (97% vs 95%).

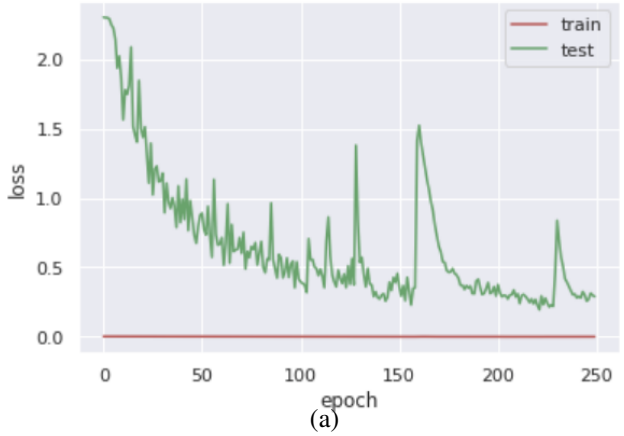


(a)

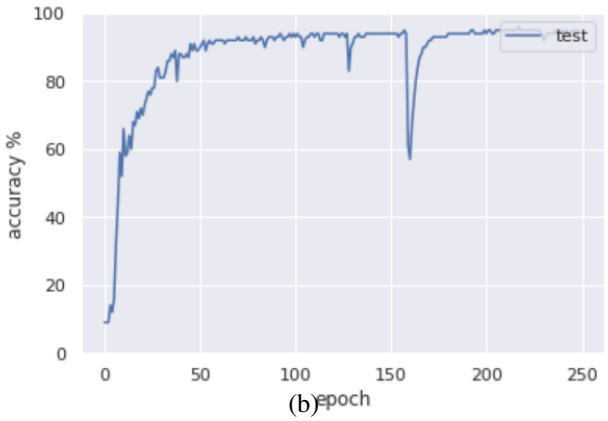


(b)

Fig. 7: (a) train and test loss over epochs using non plastic model (b) Accuracy of non plastic model over epochs



(a)



(b)

Fig. 8: (a) train and test loss over epochs using plastic model (b) Accuracy of plastic model over epochs

IV. CONCLUSIONS

The inclusion of Plasticity in conventional neural network architecture promises good results in certain experiments as discussed in work. However, we tried including plasticity in more common set of vision related task (denoising, classification) and did not find much improvement over the baseline model not using plasticity. One of the downside of using this framework of plasticity is that it fails to perform well when inputs are passed in batches and hence requires the input to be provided as one sample at a time, this inherently makes the training much slower even on GPUs since we can't process multiple samples at a time (which is very easily doable and common practice in deep learning).

REFERENCES

[1] Miconi, Thomas & Clune, Jeff & Stanley, Kenneth. (2018). Differentiable plasticity: training plastic neural networks with backpropagation, Proceedings of the 35th International Conference on Machine Learning (ICML2018), Stockholm, Sweden, PMLR 80, 2018

[2] Uber Engineering: https://eng.uber.com/differentiable-plasticity/

[3] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[4] Baldi, P., 2012, June. Autoencoders, unsupervised learning, and deep architectures. In Proceedings of ICML workshop on unsupervised and transfer learning (pp. 37-49).

[5] Hopfield, J.J., 1988. Artificial neural networks. IEEE Circuits and Devices Magazine, 4(5), pp.3-10.

[6] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[7] Radford, A., Metz, L. and Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[8] Deng, L., 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine, 29(6), pp.141-142.