Unsupervised Neuron Selection for Mitigating Catastrophic Forgetting in Neural Networks

Ben Goodrich
Department of Electrical Engineering and
Computer Science
University of Tennessee
Knoxville, TN 37996-2250
Email: bgoodric@utk.edu

Itamar Arel
Department of Electrical Engineering and
Computer Science
University of Tennessee
Knoxville, TN 37996-2250
Email: itamar@ieee.org

Abstract—**Catastrophic forgetting is a well studied problem in artificial neural networks in which past representations are rapidly lost as new representations are constructed. We hypothesize that such forgetting occurs due to overlap in the hidden layers, as well as the global nature in which neurons encode information. We introduce a novel technique to mitigate forgetting which effectively minimizes activation overlapping by using online clustering to effectively select neurons in the feedforward and back-propagation phases. We demonstrate the memory retention properties of the proposed scheme using the MNIST digit recognition data set.**

I. Introduction

Catastrophic forgetting (also known as catastrophic interference) is the tendency of an artificial neural network to rapidly lose old representations upon being presented with new inputs/samples. Multi-Layer Perceptrons (MLPs) in particular are highly susceptible to this problem due to the inherent shared neuron activations through which the network encodes information.

Catastrophic forgetting can severely hinder learning in nonstationary environments. Traditionally, when training a MLP one shuffles the training data to ensure that the observations are presented in an order that makes it appear stationary (in the sense that each sample is independently and identically distributed). Should the data be presented in a nonstationary manner, the network may not adequately capture the representations pertaining to all samples due to temporal bias toward a particular subset of samples. This presents serious limitations to the usefulness of MLP neural networks in online learning tasks where one may not have the convenience of deciding what order to present the training samples to the network.

A classical example whereby MLPs may fail in an online setting is when applied in the context of reinforcement learning, where a value function must be estimated in an online manner and the input samples are correlated [1] [2]. There is a close connection between catastrophic forgetting and what is known in neuroscience as the stability vs plasticity dilemma. Considering biological systems also suffer from the same effect, it is unlikely that an optimal solution to this dilemma exists, even for narrow application domains. Neural networks seem to suffer in a more pronounced way, however, motivating the need to better understand the underlying causes for the phenomenon as well as attempt to find solutions that would mitigate the effect.

It appears that catastrophic forgetting is caused primarily by two main factors. The first is overlapping representations in (dense) hidden layer activations. When a network is trained on a given sample point, due to hidden-layer activations overlap, adjustments of weights for this sample would impact the manifold learned in regions that pertain to a different mapping. This effect can be especially disastrous for networks that use activation functions that are non-zero over most of the input space - such as sigmoidal or hyperbolic tangent activations. Updating weights to a sigmoidal neuron can have consequences for that neuron's output if it is shown a completely unrelated input [3].

The other factor is simply the lack of any redundancy within the weights of most network models. In the feedforward phase, all weights are generally involved in computing the network output. During the back-propagation and weight updating phases, almost every weight in to the network is being updated. Network models that partition the weights so they are not always used, or that selectively update certain weights, tend to better cope with catastrophic forgetting.

With these two factors in mind, we propose a technique that uses online clustering, as an unsupervised learning process, to select (or mask) neurons during the feed-forward pass. This reduces overlap when presented with samples from different regions, as well as introduces redundancy to the network model. Each neuron is thus assigned a centroid, in addition to its weights set. Finally, only the neurons that have centroids that are nearest to the sample point are selected. This effectively imposes overlapping sub-networks out of a large network.

The rest of the paper is structured as follows. In Section II we briefly review prior attempts at mitigating catastrophic forgetting in supervised learning settings. Section III describes the proposed scheme and its implementation details. Section IV discusses results of applying the scheme to challenging non-stationary settings, while in Section V the conclusions are drawn.

II. Existing Techniques

Over the years, many techniques have been proposed for mitigating catastrophic forgetting in supervised learning systems. One family of solutions proposed relates to the radial basis function (RBF) network, a type of neural network which uses a localized activation function that eliminates overlap in the hidden representation. RBFs have been applied with some success [2], however they present several key challenges. First, they do not generalize well to higher dimensional spaces [4]. Second, the center vectors of the basis function must be predetermined in a way that covers the entire possible input space requiring at least some prior knowledge of the input space before training commences.

Another technique that has been explored is to simply promote sparse representations in the hidden layer. This helps eliminate overlapping representations in the hidden layer. An older approach, known as activation sharpening, attempted to achieve this [5]. Other sparsity techniques have been used with mixed success [6].

Recently, schemes such as Dropout [7], Local Winner Take All [8] networks, and Maxout [9] networks have been considered with varying degrees of success [10]. These techniques all add redundancies within the network weights. Dropout is a technique that randomly selects neurons to use during the feed forward and back propagation pass. This facilitates redundancy in the network by ensuring that only some of the weights are used on each pass. Dropout is also generally combined with Rectified Linear activation units which promote sparsity by their nature, and also help to eliminate overlapping representations. Local Winner Take All and Maxout networks are very recent techniques that employ rules to selectively choose neurons to use during the feed forward pass. This can greatly promote redundancies within the network weights.

III. Unsuperivsed Neuron Selection for Retaining Previous Representations

A. Online Clustering

The framework proposed here seeks to introduce redundancy to the network as well as reduce overlapping representations by selectively choosing which neurons are to be activated in the feed-forward pass. This section provides a brief review of how neural network training is typically done to establish some notation, then describe how we include the centroid based selection scheme.

With regular feedforward networks, each neuron can be seen as having a weight vector. Generally all of the weights from neurons in a single layer are combined to obtain a weight matrix W. The weight matrix consists of all of the weight vectors (as row vectors). More specifically, the elements of W are w_{ji} where i is an index into the previous layer and j is an index into the current layer. $W^{(1)}$ refers to the weights of the first hidden layer, $W^{(2)}$ refers to the weights of the next layer. (This discussion will only deal with networks having one hidden layer)

On each feed forward one typically passes a group of samples through each layer as a minibatch. By defining the matrix of samples as X where each sample is a column vector in this matrix, one can compute the output of the hidden layer using simple matrix algebra. $Y \leftarrow f(W^{(1)}X)$ where Y is a matrix of column vectors of outputs for the hidden layer and $f(\bullet)$ is the hidden activation function. The final network output can be computed as $Z \leftarrow g(W^{(2)}X)$ where g is the output activation function (if any).

To apply dropout in a feed forward pass, one could define a R matrix of random binary values that take on 0 or 1 with probability 0.5. All one would need to do is define $Y \leftarrow f(R \odot (W^{(1)}X))$ where $\odot$ denotes element-wise multiplication.

Every neuron can be given a centroid by defining a centroid matrix that has the same dimensions of W (call it $\mathcal{C}$). That is, every neuron's centroid is a row vector in this matrix. This matrix will be used to select neurons whose centroids are nearest to the sample point. A matrix Y^d needs to be defined that tells the distance of each neuron's centroid to each sample point.

The elements of Y^d can be computed by finding centroid distances. Using euclidean distance this becomes the following.

$$y^d_{jl} \leftarrow (c_{j1} - x_{1l})^2 + (c_{j2} - x_{2l})^2 + \cdots + (c_{jn} - x_{nl})^2 \quad (1)$$

More generically, Y^d is a matrix of distances from a row of C to a column of X (neuron centroid to sample point).

$$y^d_{jl} = dist(\mathcal{C}_{j,*}, X^t_{*,l}) \quad (2)$$

Once Y^d has been computed, it can be used to deactivate neurons that are too far away. That is, if a neuron's distance is beyond some threshold for a particular sample, its activation is set to 0. That neuron is also not updated in the back propagation phase since it was unused. Selective feed forward and updating is done by generating a mask matrix from Y^d defined as M consisting of binary 0,1 values. M is the same dimensions as both W and $\mathcal{C}$ and is used to deactivate neurons. If y^d_{jl} exceeds the threshold, then that neuron's centroid is too far away and the corresponding m^d_{jl} element in the mask matrix is set to 0.

$$m_{jl} \leftarrow \begin{cases} 1 & \text{if } y^d_{jl} < y^{d,(k)}_l \\ 0 & \text{else} \end{cases} \quad (3)$$

Selecting k nearest neurons for each sample requires doing a column-wise sort operation on Y^d and taking the kth row and using that as a threshold. In the notation above $y^{d,(k)}_l$ represents the k'th order statistic (k'th smallest value) on the l'th column of Y^d.

On feed forward the M matrix is used to mask out neurons whose centroids are too far away from the samples by doing $Y \leftarrow f(M \odot (W^{(1)}X))$. This matrix is also used in the back propagation phase to ensure that weights are not updated for neurons that were not selected.

B. Centroid Placement

Ideally, one would want to have centroids placed in a way that minimizes overlap between old and new classes. Clustering should be done in a way that preserves the topology of the input space where nearby sample points cause the same neurons to be selected while distant and unrelated sample points would cause different paths in the network to be activated.

One way to place centroids would be to attempt to detect when the network error suddenly increases beyond its normal value. When this occurs, it is assumed that the network is being presented with something new. The neurons that were selected are inadequate to handle this new information, previously unselected neurons must be selected to handle this new sample data.

A concept from reinforcement learning known as an eligibility trace is kept for each neuron [11]. If a neuron k is selected then its eligibility gets increased by 1.0.

$$e_k \leftarrow e_k + 1 \quad \forall\, k \text{ selected} \tag{4}$$

All neurons have their eligibility decayed by a small factor $\beta = 0.99$ (regardless if they are selected).

$$e_j \leftarrow \beta * e_j \quad \forall j \tag{5}$$

The purpose of eligibility is to keep track of neurons that have remained unused over the longest period of time. When placing new centroids, neurons are selected that have the lowest eligibility to overwrite their centroid location.

A moving average is kept for the average class label error rate for each label the network recognizes. If the error increases beyond this moving average by a certain threshold then new centroids are placed. The k neurons that have the lowest eligibility are chosen to have their centroids replaced. New centroids are placed at the location of the input samples that had a larger than normal error (The moving average error is also reset so it won't be immediately triggered again next time).

Before training begins, centroids are initialized to some very far away location (These can be considered unused centroids). The moving average error detection scheme is initialized to a very small value, this will cause it to trigger on the first epoch which will place new centroids then. After the first epoch, the network trains using the centroids that were placed on the first epoch to select neurons. The error detection scheme waits until a sudden increase in error is detected, at which time it allocates new centroids to the locations of the sample data that caused the increase. This will cause those sample points to use the new centroids while preserving the older centroids.

In addition to manually placing centroids, centroids are moved closer to the sample points that triggered their selection. That is, the average location of all sample points that selected a centroid is used as a new location to move the centroid to by some velocity.

Algorithm 1: Anti-forgetting centroid based training

```
Initialize all centroids to some far away value
Initialize $error_l\ \forall\, l$ to a very small value
for each minibatch of training do  training loop
    Feed forward minibatch using centroid selection.
    Compute error
    for each class label do
        Compute class label error as $error_l$
        if $error_l > moving\_average\_threshold$ then
            Choose centroids from samples that caused
            error increase. Place the new centroids on
            least eligible neurons.
            $moving\_average_l \leftarrow error_l$ (reset moving
            average error to prevent re-triggering)
        end
        $moving\_average_l \leftarrow$
        $0.95 * moving\_average_f + 0.05 * error_l$
    end
    $\mathcal{C} \leftarrow C_{old} + \alpha * (\mathcal{C}' - \mathcal{C}_{old})$
    Back Propagate Error & Update Weights
end
```

IV. Results

The training task that was conducted was similar to the task in [8]. However, unlike [8] weight restoration was not done as we feel that this invalidates the task. All 10 MNIST digits were divided into two sample sets. 5 of the digits were placed into dataset P1, and the other 5 were placed into dataset P2. Training was performed on a neural network with a single hidden layer and 5 outputs (one for each label) to recognize P1. At epoch 1000 training was switched to dataset P2 and continued. While the network learned to associate class labels for P2, performance was evaluated on the test set for P1.

All networks had 2048 hidden neurons. They used tanh activations and tanh outputs as described in [4]. Results are included from one network that had 2048 Rectified Linear hidden activation units with a softmax output and dropout for comparison to an existing technique that is known to help with forgetting. All runs have a learning rate of .0025 with a minibatch size of 1024.

Centroid movement speed was set to $\alpha = 0.001$ per minibatch. That is, the average position of the k samples that each centroid selected are used to build a matrix (denoted by $\mathcal{C}'$). The centroid positions are then updated to be:

$$C \leftarrow C_{old} + \alpha * (C' - C_{old}) \tag{6}$$

The threshold for the error detection scheme was set to 5.0, and 128 new centroids were placed if the threshold was exceeded. That is, if the class label error was more than 5 times greater than the moving average error for that label, 128 centroids were placed.

The threshold for the error detection scheme is an important parameter. This threshold could be done away with and new centroids could simply be placed when the P1 to P2 switch occurs, but that defeats the purpose.The reason for having this threshold is so that the network could innately detect when something new is seen, in essence the network becomes a black box that is capable of handling non stationary problems. Ideally it should be set to a value that triggers only when the P1 to P2 switch occurs. If it is set too low, it may be triggered at inappropriate epochs, and if it is set too high it won't trigger at the P1 to P2 switch. From the measurements on this MNIST task, any value from 4.0 to 11.0 would have worked.

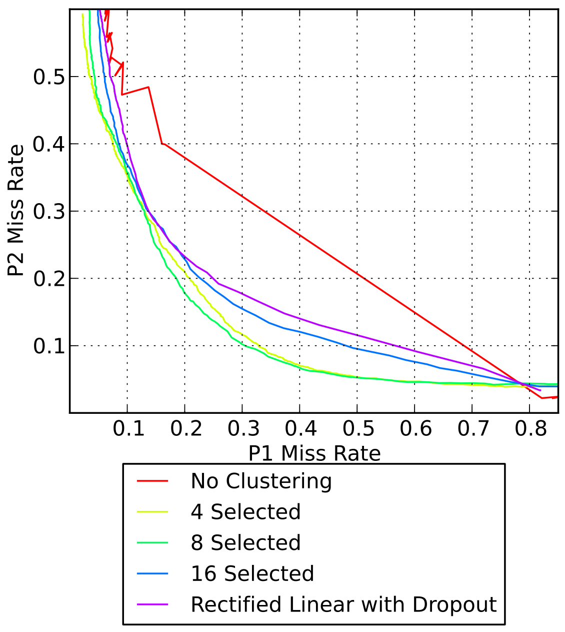

Fig. 1. P1 Miss Rate vs. P2 Miss Rate Possibilities Frontiers for MNIST Forgetting Task

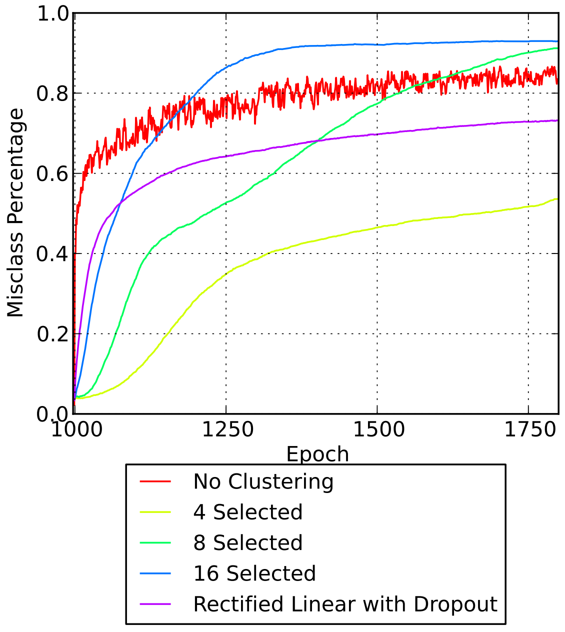

Fig. 2. P1 Miss Rate while training on P2

Figure 1 shows a possibilities frontiers curve for the P1 error rate vs. The P2 error rate. This plot is similar to the plots provided in [10]. It traces the error for P1 relative to P2, in this case lower error for both tasks is better so the curve that is nearest to the bottom left shows a network that is able to learn both P1 and P2.

A possibilities frontiers plot is useful because it shows that there were at least one or more epochs where the network was able to retain memory for both tasks, however this may not tell the whole story. It does not show how relatively quickly each network forgot. The network could have learned both P1 and P2 well for only one epoch, then very quickly forgot. It also does not show how the network forgot over time, that is if the forgetting follows a linear trajectory or if it loses accuracy at an exponential rate.

Figure 2 shows the forgetting rates for each network starting at the epoch where the P1 to P2 switch occurred. As can be seen, the regular network with no clustering immediately loses P1, while the other networks forget at a more gradual rate.

V. Conclusion

This paper has argued that building a path through a network using online clustering can help mitigate catastrophic forgetting in a data-driven manner. Building such paths through the network based on deterministic rules addresses both issues of injecting redundancy within the network as well as reducing activation overlap. The technique proposed can be efficiently implemented over parallel processing platforms. Moreover, the approach can be broadened and applied to other connectionist architectures, such as recurrent neural networks.

References

[1] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems 7*. MIT Press, 1995, pp. 369–376.

[2] V. U. Cetina, "Multilayer perceptrons with radial basis functions as value functions in reinforcement learning."

[3] S. Weaver, L. Baird, and M. Polycarpou, "Preventing unlearning during online training of feedforward networks," in *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, 1998, pp. 359–364.

[4] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop," in *Neural Networks: Tricks of the trade*, G. Orr and M. K., Eds. Springer, 1998.

[5] R. M. French, "Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks," in *In Proceedings of the 13th Annual Cognitive Science Society Conference*. Erlbaum, 1991, pp. 173–178.

[6] R. French, "Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference," 1994.

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[8] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber, "Compete to compute," in *Advances in Neural Information Processing Systems*, 2013, pp. 2310–2318.

[9] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[10] I. J. Goodfellow, M. Mirza, X. Da, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgeting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[11] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.