Neuron Clustering for Mitigating Catastrophic Forgetting in Feedforward 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 fundamental problem with artificial neural networks (ANNs) in which learned representations are lost as new representations are acquired. This significantly limits the usefulness of ANNs in dynamic or non-stationary settings, as well as when applied to very large datasets. In this paper, we examine a novel neural network architecture which utilizes online clustering for the selection of a subset of hidden neurons to be activated in the feedforward and back propagation passes. It is shown that such networks are able to effectively mitigate catastrophic forgetting. Simulation results illustrate the advantages of the proposed network with respect to other schemes for addressing the memory loss phenomenon.**

I. Introduction

Catastrophic forgetting is a well-studied problem that affects artificial neural networks as well as many other supervised learning systems [1]. When being presented with new information neural networks tend to quickly unlearn prior representations that are not being presented. While there is evidence that this phenomenon exists in biological systems, loss of information tends to occur in a much more pronounced manner in artificial neural networks.

Traditionally, neural network training is structured in a way that helps mitigate this effect. When training in a fixed environment, training data is generally shuffled and presented in a random order [2]. The data must be presented in a way that makes it appear stationary (each sample is independent and identically distributed). If the data is presented in a manner that is non-stationary, the network may not be able to capture and retain representations that are presented at irregular intervals.

Traditionally, dynamic environments have been recognized as tasks that are challenging for neural networks. If the task or environment changes, a neural network tends to catastrophically forget the previous learned task or environment setting, as it learns the new. However, large enough stationary environments may appear non-stationary if the sampling scheme is such that the inputs are not identically distributed relative to the parameter adaptation rate.

Another area where catastrophic forgetting may be having a detrimental effect is with our ability to train large networks on large datasets. In order for a neural network to capture all of the characteristics and features within a very large dataset, it must also be made large (large number of weights). There appears to be a law of diminishing returns such that adding capacity the network captures fewer features of the large dataset [3]. Catastrophic forgetting may be at play here. If a dataset is large then key characteristics will be under-presented, and as a result the network will not be able to adequately capture the details of such features before they are lost.

Catastrophic forgetting is a problem that needs to be addressed if we want to apply neural networks to a wider range of tasks outside of the domain in which they are presently being applied. Neural networks simply fail in many online and reinforcement learning tasks, and are unable to cope with changing environments due to their inability to retain old information.

Catastrophic forgetting in multi-layer perceptron networks has two main underlying causes. The first is the overlapping of global representations within the hidden layer(s). Activation functions that are non-zero over most of the input space naturally provide a global representation. As a side effect, with these activation functions, a gradient based update to minimize error for a sample point in one region of the input space will affect the output for a sample point that is completely unrelated [4].

The other cause is the lack of redundancy in a network's weights. That is, most models do not set aside unallocated resources to address data diversity. If the model is such that every neuron is involved in computing a network's output, and every weight is potentially updated to minimize error, then it is no surprise that the network can only minimize error for the samples that have recently been presented.

The rest of the paper is structured as follows. Section II outlines a brief history of other related techniques with a short discussion on existing techniques for mitigating forgetting affects in neural networks. Section III describes our approach and its implementation details. Section IV contrasts our technique to other schemes while Section V provides results and a discussion for several test cases. Finally, Section VI draws some conclusions.

II. Background

There have been numerous techniques proposed in the literature to address the issue of catastrophic forgetting. Prior to the recent renaissance of neural networks, one can find numerous older ideas and techniques dating back to the 1990s [1].

Some techniques that came out of that era simply attempted to promote non-overlapping (or sparse) representations [5]. A technique known as activation sharpening [6] directly accomplished this very goal. Activation sharpening works by adding an update that explicitly tries to make the activations sparse. It is interesting that overlapping representation was identified as a cause of forgetting many years ago.

One technique that works in simpler (i.e. lower dimensional) problem domains is simply to use radial basis function (RBF) networks. RBF networks have local activation functions (zero over most of the input space), and do not suffer as much from catastrophic forgetting. Unfortunately, they have a few disadvantages. The main issue with RBF networks is that one must know where to place the centroids before training can begin, this requires at least some of the sample data to be present at the beginning of training. The other issue is that these are not deep networks; RBF networks have one layer and as a result they have trouble generalizing, especially in high dimensional spaces. They suffer from the curse of dimensionality in that many basis functions are needed to cover higher dimensional spaces [2].

More recently, other techniques have been found to help mitigate catastrophic forgetting, including neural network architectures known as maxout and local winner-take-all networks. A technique known as Dropout has also been found to help. Although the authors in [7] found that the technique that works best often depends on the task, they demonstrated that maxout networks were effective in at least 3 test cases. These techniques add redundant weights in the network such that only a subset of the neurons are active for every feedforward pass. These techniques warrant a bit more discussion for which the following section provides a brief description.

A. Maxout Networks And Local winner-take-all Networks

In standard feedforward neural networks, the output of all k neurons within a layer is computed as $\vec{y} = f(W\vec{x})$ where $\vec{x}$ is the input vector, W is the weight matrix and f denotes a non-linear activation function. Local winner-take-all networks group neurons within a layer where there are n neurons per group. In local winner-take-all architectures, only the neuron within the group that yielded the largest output is allowed to have a non-zero output. All other neurons within the group are forced to produce an output of zero.

Maxout networks are similar in concept - they also group neurons within a layer such that there are n neurons per group. The key difference is, maxout networks only have a single output per group, where the group output is set to the winning neuron's output. A maxout layer with k neurons and n neurons per group will produce k/n outputs, where local winner-take-all would have n outputs with non-winning outputs set to zero.

In both maxout and local winner-take-all settings, during back-propagation the neuron that was selected for output will have its gradient updated, while other neurons in the group will be discarded. Thus, only a subset of the neurons are active for any given feedforward pass.

An alternative way of looking at the differences between maxout and local winner-take-all is in terms of sparsity. Local winner-take-all provides a sparse representation in the feedforward pass, since only a single neuron within each group has a non-zero output. In contrast, maxout does not provide a sparse representation because a layer hosts only a single output per group, and all groups may provide non-zero outputs. Maxout does, however, offer sparse gradient update since only the neuron that contributed to the group's output will be updated. For a more in depth description of maxout and local winner-take-all, the reader is referred to [8] and [9]. Related discussion can also be found in [7].

B. Dropout

Dropout [10] is a regularization technique aimed at reducing overfitting by randomly (usually with a probability 0.5 [11]) setting neuron outputs to zero. This effectively randomly selects neurons to form smaller (sub)network models out of the larger network being trained. During the inference phase, the full network is used, which can be seen as averaging all of the smaller network models. Dropout was commonly used with rectified linear activations, although [7] reported that it helped with catastrophic forgetting when applied to maxout networks.

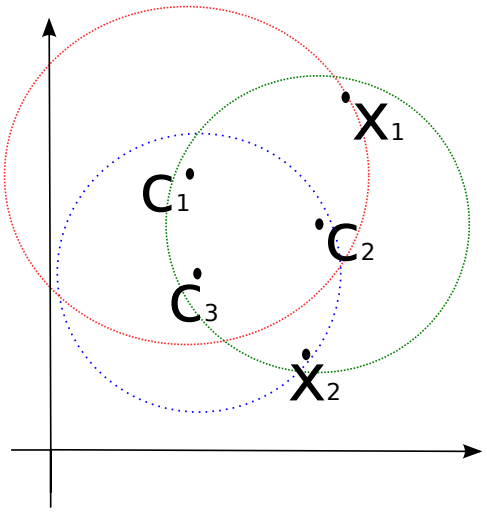
III. Cluster-Select

The technique proposed here, coined "cluster-select", attempts to reduce hidden layer overlap as well as inject redundancy to the hidden layer activations. In regular feedforward neural networks, each neuron can be seen as having a weight vector. During the feedforward process, each neuron's output is computed by performing a dot product of its weight vector with the neuron's input, in addition to adding the bias term. In the case of cluster-select, each neuron is allocated a centroid vector c in addition to the weight vector. For any given input vector, denoted by $\vec{x}$, the distance from that input to the neuron i's centroid is computed as:

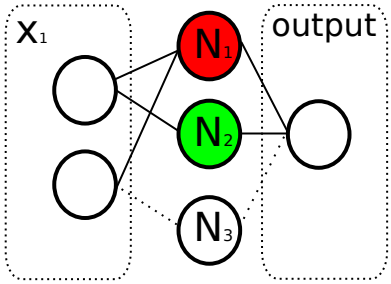
$$d_i = \|\vec{x} - c_i\|_2^2 \tag{1}$$

In the default case, this is simply the squared Euclidean distance. Upon computing distances for every neuron to a given input, the k neurons whose centroids are nearest to the input vector are selected. To achieve this, the distances are ranked by choosing the k^{th} entry in the list where $d^{(k)}$ gives the distance of the k'th entry. Any hidden units beyond this distance $d_i > d^{(k)}$ have their outputs forced to zero. Fundamentally, during each feedforward pass only the neurons that were selected are allowed to have a non-zero activation value. Every neuron that was not selected has its output forced to zero. Consequently, weight updates are applied only to the subset of selected neurons.

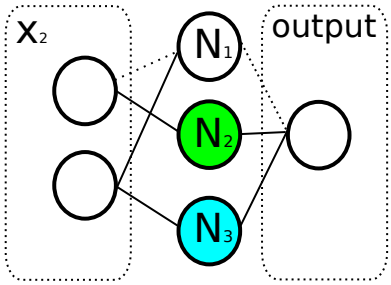
Figure 1 depicts the proposed technique as applied to a simple 2-dimensional case. Shown in figure 1(a) are 3 centroids labelled c_1, c_2, and c_3. Figure 1(b) and 1(c) illustrate a network with one hidden layer, where centroids c_1, c_2, and c_3 belong to neurons N_1, N_2, and N_3 respectively. In this example 2 out of 3 neurons are selected, such that $k = 2$. Note that sample X_1 is nearest to c_1 and c_2 in figure 1(a). As a result, if X_1 provided as input to the network (as shown in figure 1(b)) it will only activate neurons N_1 and N_2. On the other hand X_2 is nearest to c_2 and c_3, hence it will activate neurons N_2 and N_3, as shown in figure 1(c).



(a) Illustration of 3 centroids (c_1, c_2, and c_3) and 2 sample points (X_1 and X_2)



(b) Illustration of the case whereby X_1 is input to the feedforward pass. Only neurons N_1 and N_2 are selected.



(c) Illustration of the case when X_2 is input to the feedforward pass. Only neurons N_2 and N_3 are selected.

Fig. 1. Illustration of the proposed cluster-select process

We next address the issue of when and how centroids are to be placed and updated. Ideally, we want a way to detect when the data has changed regimes before placing new centroids. The network output error was chosen as an indicator for when the statistical properties of the input data have changed. Centroids are placed whenever the network detects that the output error for a given class label has suddenly increased beyond some threshold. Training is performed on a mini-batch basis in which a mini-batch consists of a matrix of samples that are fed to the network sequentially, and an aggregate gradient is calculated for the purpose of weight updating.

A moving average error is maintained for each class label. If ϵ_l represents the moving average error for class label l, and e_l denotes the mean squared error in the current mini-batch for all samples belonging to class label l, then ϵ_l is updated according to:

$$\epsilon_l \leftarrow (1-\alpha)e_l + \alpha\epsilon_l \tag{2}$$

Where alpha is some constant close to 1. We used 0.95 and found that this value worked well in most settings.

If the error suddenly increases beyond the moving average by a certain threshold t (that is if $\epsilon_l > te_l$), then new centroids are placed, indicating that the network is being presented with data of a different statistical nature, one which it has not been trained on. The reason for maintaining a moving average error, and attempting to detect when such error increases, is to provide a data-driven method for the network to detect when change has occurred. The neural network should be viewed as a black box that can process non-stationary data and internally allocate resources as needed.

When the placement of new centroids is triggered, the neurons that have the lowest eligibility will have their centroids overwritten by the new ones. Eligibility is a concept borrowed from reinforcement learning [12]. During each feedforward pass, should a neuron be selected, its eligibility is additively increased by 1. All neurons have their eligibility decayed by a constant factor regardless of whether they are selected. Eligibility's main purpose is to keep track of which neurons are not being used.

Centroids are placed simply by taking samples belonging to the current mini-batch and overwriting the centroid location of the neurons that have the lowest eligibility. Once new centroids are in place, the moving average error ϵ_l is reset to e_l to prevent the triggering of the $\epsilon_l > ke_l$ criteria from placing additional centroids.

During training, centroids are shifted closer to samples that correspond to them. If a neuron was selected by its input, then its centroid is moved by some small constant toward the location of the sample that selected it.

$$\vec{c}_i^{new} \leftarrow \vec{c} + \beta(\vec{x} - \vec{c}) \tag{3}$$

This strives to place centroids in locations that represent dense regions of samples. β is a small constant close to 0 and is considered a hyperparameter. To prevent neuron weights and outputs from growing to very large values, a hyperbolic tangent activation function was applied during feedforward to the neurons that were selected. Note that the hyperbolic tangent activation function used is $1.7159tanh(2/3x)$, where the constants came from [2]. When training commences, centroids are initialized to a value that is far away from any sample point (all elements are set to -10). The moving average error estimate is also initialized such that it will trigger placement of new centroids on the very first epoch.

A. *Ensemble*

Under certain conditions, an ensemble of learners can contribute another layer of robustness to mitigating catastrophic forgetting. Networks with different hyperparameters tend to learn and forget at different rates. Some networks will be very fast to learn new representations but will also be fast to forget samples that are not being frequently presented.

Algorithm 1: Cluster-Select Training

```
Initialize all centroids to some far away value.
Initialize ε_l ∀ l to a very small value.
for each mini-batch of training do  training loop
    feedforward mini-batch using centroid selection.
    Compute error.
    for each class label do
        Compute class label error as e_l.
        if ε_l > te_l then
            Choose centroids from samples that caused
            error increase. Place the new centroids on
            least eligible neurons.
            ε_l ← e_l (reset moving average error to
            prevent re-triggering)
        end
        ε_l ← (1 − α)e_l + αε_l
    end
    c_i^new ← c + β(x − c)
    Back-propagate error and update weights.
end
```

Other networks will require new information to be presented many times before it can be learned, but they will also be slow to forget old information. By employing an ensemble of learners of both types of networks, it is possible to obtain the best of both types of networks. Moreover, networks tend to forget different regions of the sample space, depending on the hyperparameters chosen. An ensemble can exploit this property and allow the networks to boost each other.

IV. Comparison to other Techniques

Our technique is similar to dropout in that it essentially invokes a sub-network out of the selected neurons. Unlike dropout, which selects neurons stochastically, we deterministically set neuron outputs to zero based on a data-driven criteria. The clustering process aims to add redundancy in the network by localizing each sub-network to regions of the input space that have centroids.

Cluster-select combines the locality of RBF networks with the generalization capability of regular neural networks. By selectively activating neurons depending on how close they lie to the sample point, only neurons that are in close proximity will be activated. The neurons that are selected are used exactly the same as a regular feedforward network, allowing them to generalize in the same manner.

Both maxout and local winner-take-all do something similar, they selectively activate neurons within a group allowing the selected neurons to focus on the regions in which they are trained. Unlike local winner-take-all and maxout, the proposed technique explicitly attempts to cluster the regions of the input space in which the neurons are activated.

V. Results

A. Test Setup

In order to explore different aspects of the proposed approach, experiments were performed on three different datasets. We first considered the MNIST handwritten digits dataset [13]. The second test case was on the 20 newsgroups dataset [14], and the third case was on an artificial dataset consisting of random binary patterns. In all cases, we ran multiple simulations using the hyperopt library to achieve hyperparameter optimization [15]. This allowed us to be more objective in comparing techniques, since it is possible to inject bias if a human is tweaking the hyperparameters. Hyperparameters were chosen randomly for each experiment.

In order to simulate a dynamic environment, all tests partitioned the datasets into two parts, $P1$ and $P2$. Once training on $P1$ reached an acceptable low error level, training was switched to dataset $P2$ and performance was measured on both $P1$ and $P2$ while the network continued to learn $P2$. The objective function which hyperopt optimized over was $min(P1_{error} + P2_{error}) \quad \forall \quad epochs$ or the minimum error reached for $P1 + P2$ at some point in training. If this error is 0, it would mean that at some point in training the network was able to learn both $P1$ and $P2$ with no error.

Initially, we had included learning rate as a hyperparameter, however we noticed that results were often difficult to compare since it affects the speed of convergence of the network. We ended up with some networks that had very high learning rates and were very unpredictable. Although they often did well to minimize the hyperopt objective function, when plotted results were very noisy and converged overall to much worse solutions, rendering the results difficult to interpret. It was decided that using a constant learning rate across all simulations made the results much easier to compare. All networks had 2 hidden layers and an output layer. To keep things simple, both hidden layers were always given the same set of hyperparameters for a given run (with the exception of the run with rectified linear activations).

For the maxout and local winner-take-all techniques, we had hyperparameters for the nodes per group (varied from 2 to 128), and the number of hidden neurons (varied from 1024 to 2048). The output layer was set to softmax as softmax seems to be the most common output type used in the literature that performs classification with these types of networks. Dropout results are also included with both rectified linear activations and maxout activations. We used a dropout rate of 0.5 since it is known to work best. For rectified linear activations, the only hyperparameter we had was the number of neurons per layer. We allowed the hidden layers to host a separate number of neurons to avoid having only one hyperparameter.

The cluster-select technique required a several other hyperparameters. There were hyperparameters for the number of centroids selected on each feed-forward pass, the number of centroids to replace when the $P1$ to $P2$ switch was detected, and the speed at which to shift centroids (β in the previous section). Since it is not clear which layers yielded the highest contribution to clustering, a discrete parameter was included to indicate whether the first hidden layer should be clustered, the second, or both. If clustering was turned off for a given layer, all neurons would feedforward on that layer with hyperbolic tangent activation functions. The output layer was set to hyperbolic tangent.

Another important hyperparameter to consider for cluster-select is the threshold t at which error must increase before new centroids are allocated. If it is too low, centroids will be

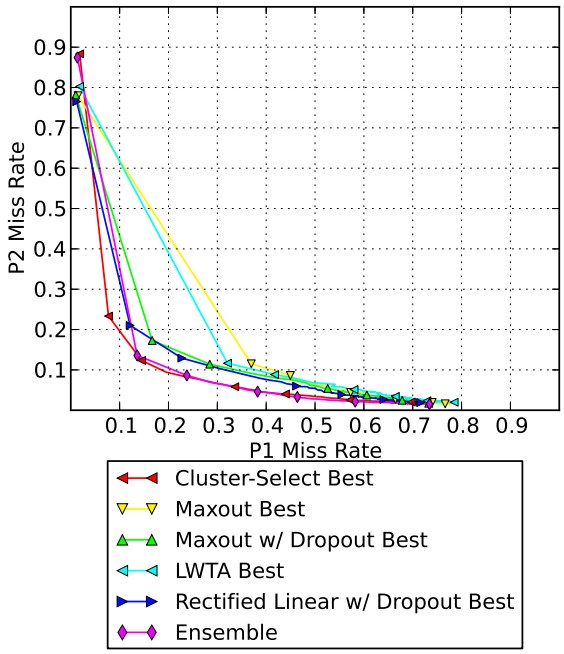


Fig. 2. P1 miss rate vs. P2 miss rate possibilities frontiers for MNIST forgetting task

inappropriately allocated at a time other than when the $P1$ to $P2$ switch occurs. If it is too high, then no centroids will be allocated when the $P1$ to $P2$ switch occurs. Fortunately, we found that this parameter has a very wide range that works. The criteria considered was that centroids were allocated only at the beginning of training, and when the $P1$ to $P2$ change occurs. This hyperparameter was set to a known value that worked for these tests and is also dependant on the moving average threshold rate α, which was set to 0.95.

Results are also included for an ensemble of learners, since it was noted that each run with different hyperparameters would tend to learn and forget at different rates. The hope was that an ensemble of networks could boost each other, allowing for it to degrade much more gracefully in the context of catastrophic forgetting. For the ensemble case, we did not search for any hyperparameter; instead we took the 5 best hyperparameters from cluster-select and used them to created an ensemble of 5 networks. To measure error for the $P1$ and $P2$ test sets during training, we simply used a linear combination of the ensemble outputs.

B. MNIST Test

For the MNIST test, we divided MNIST into two subsets, half of the digits were placed into dataset $P1$, and the other half were placed into dataset $P2$. This gave 5 total class outputs for the network. The neural network was trained until no improvement was observed for 100 epochs on $P1$, then training was switched to $P2$. This made it so that a new set of 5 digits would map to the old labels. The learning rate for this test was set to 0.005.

During training on $P2$, we observed the error rate on the test set which was also divided into $P1$ and $P2$ to generate a possibilities frontiers plot, as shown in figure 2. This technique of plotting the impact of catastrophic forgetting was first introduced in [7] and illustrates the error on $P2$ relative to $P1$. The closer the curve gets to the bottom left hand corner, the better the network was able to capture both $P1$ and $P2$ at some point during training. How close the curve gets to the bottom left corner corresponds directly to the loss function used for hyperopt. Figure 2 depicts the best hyperparameter sets out for all of the experiments. Cluster-select performs the best here with the ensemble of the 5 best cluster-select networks giving nearly identical performance.

C. 20 Newsgroups Test

The 20 newsgroups dataset is a text classification dataset consisting of 18,837 posts to 20 usenet newsgroups. The task is to determine which of the 20 newsgroups the post was submitted to, based on the contents of the post. Before this could be fed to a neural network, some technique of feature extraction from the text had to be done. We chose to use the TF-IDF method to extract 2000 features [16]. This dataset has a problem of fitting only to 2 or 3 features if they are included; so we chose to remove the headers, footers, and quotations block as recommended by the scikit-learn documentation when dealing with this dataset [17]. Removing these features makes this a much more challenging problem. We decided to reshuffle the training and testing data such that we chose randomly chose 2048 feature vectors for testing, and 16,789 for training.

This set had 20 total classes, and was divided into two segments ($P1$ and $P2$ each consisting of 10 classes labels. When no improvement was observed for 300 epochs, training was switched from $P1$ to $P2$ such that a new set of classes would map to the old labels. Error rates were observed on the $P1$ and $P2$ test sets during the switch from P1 to P2 and plotted in figure 3.

It is possible to achieve a better training accuracy by including many more than 2000 features, however this would greatly slow down training, and we felt that this test was to measure forgetting, not to achieve the highest accuracy on this data set. For this test we fixed the learning rate to .025.

Figure 3 shows the results. A total of 75 experiments were ran for each type of network with random hyperparameter sampling. All techniques were able to achieve an accuracy of around 30% on $P1$. In the context of learning both P1 and P2, the cluster-select was able to perform the best. The ensemble of 5 cluster-select networks performed only slightly better than a single network. The difference isn't very significant.

One surprising result here is that dropout seemed to harm forgetting performance. This is contrary to what others have reported in other tests and differs from our MNIST result where dropout appears to greatly aid performance.

D. Autoassociative Encoder

In order to evaluate this technique on a regression task, an artificial dataset was set up containing 200 random binary patterns. Each binary pattern contains 100 random binary inputs and 100 random binary outputs. The task is to train a neural network to associate a given 100 bit input to an unrelated 100 bit output. All networks in this test had linear outputs to deal with regression, and the binary targets were either -1 or $+1$.

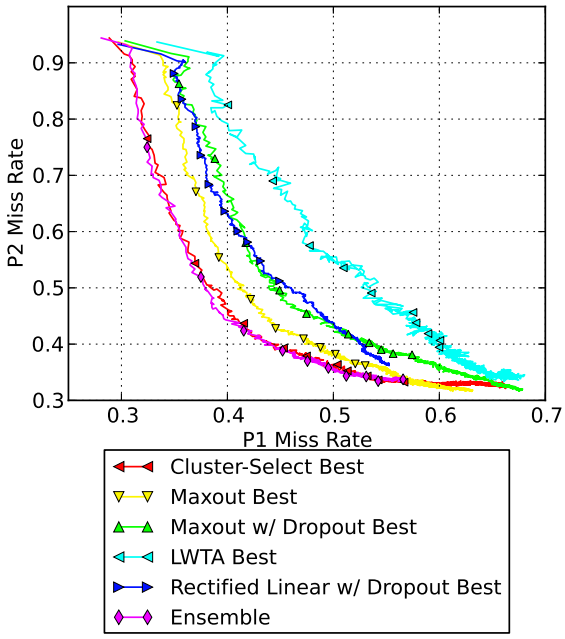


Fig. 3. P1 miss rate vs. p2 miss rate possibilities frontiers for 20 newsgroups dataset forgetting task

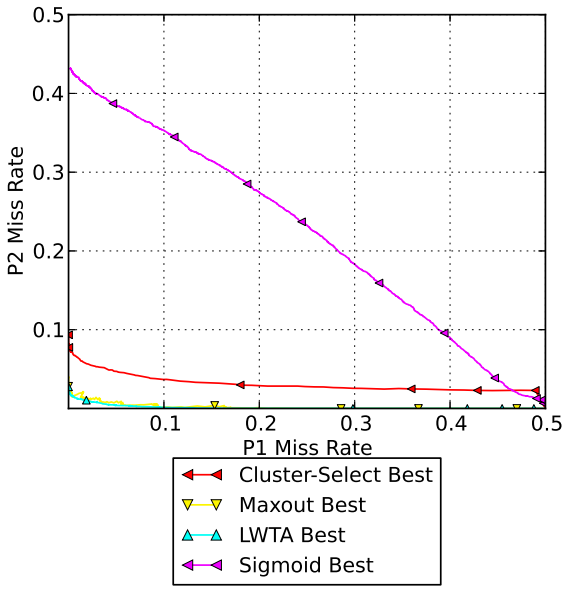


Fig. 4. $P1$ Miss Rate vs. $P2$ Miss Rate Possibilities Frontiers for Autoassociative Encoder Forgetting Task

The dataset was divided into two parts each containing 100 patterns. As before, training was done on the first dataset $P1$, then when the error had not decreased for 300 epochs, training was switched to the second dataset $P2$. Error rate is measured in the percentage of total bits the network got incorrect. A bit was considered correct if the sign of the network output matched the sign of the bit (i.e. if the network output is less than 0 the bit must be -1 to be considered correct, else if the network output is greater than 0 the bit must be 1 to be correct). Randomly guessing would produce an error rate of about 50%, hence the graph scale in figure 4 goes to 50%. Because this dataset was so small, we did not divide it up into mini-batches, instead we trained on the entire batch on each epoch.

For this test 100 experiments were performed per network type. As can be seen, this test actually shows local winner-take-all and maxout ahead of cluster-select. A network with sigmoid hidden layers was also included for comparison. This test shows that all three techniques (maxout, local winner-take-all, and cluster-select) do well on this regression test. We believe cluster-select did not do as well as it could have because there was still some overlap in the cluster selection. Some centroids for $P1$ were likely selected when training on $P2$. Considering that each of the 100 vectors in P2 had to have k centroids selected (k being the number of centroids selected, a hyperparameter), at least some of those inputs likely selected centroids that were allocated for $P1$.

VI. Conclusion

This paper introduced a novel technique aimed at mitigating catastrophic forgetting in supervised learning based neural networks. The technique works to minimize overlap in the hidden layer as well as add redundant neurons that are selectively activated, based on a data-driven criterion. The proposed approach performs better than current state-of-the-art schemes for classification and offers comparable results on regression tasks. Given that many machine learning problems involve non-stationary dynamic environments, the framework shown can be broadly applied.

Acknowledgement

The authors would like to thank James Bergstra, Daniel Yamins, and David D Cox for the hyperopt library for doing automated hyperparameter optimization [15].

References

[1] O.-M. Moe-Helgesen and H. Stranden, "Catastophic forgetting in neural networks," *Dept. Comput. & Information Sci., Norwegian Univ. Science & Technology (NTNU), Trondheim, Norway, Tech. Rep*, 2005.

[2] 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.

[3] Y. Dauphin and Y. Bengio, "Big neural networks waste capacity," *CoRR*, vol. abs/1301.3583, 2013.

[4] 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.

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

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] N. Srivastava, "Improving neural networks with dropout," Ph.D. dissertation, University of Toronto, 2013.

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

[13] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[14] "20 newsgroups dataset." [Online]. Available: http://people.csail.mit.edu/jrennie/20Newsgroups/

[15] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 115–123.

[16] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *In Proceedings of the First Instructional Conference on Machine Learning.*, 2013.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.