Procedia Computer Science
A Localist Paradigm for Big Data
Volume 53, 2015, Pages 356–364
Tsvi Achler
2015 INNS Conference on Big Data
*ITOP Corp, Palo Alto, CA, achler@gmail.com*

**Abstract**
Big data problems involve more than being able to create a network that can recognize based on a big data set. Big data problems also involve being able to incorporate new information as it arrives. Rehearsing big data sets may require an inordinate amount of time. We present a localist neural network recognition method that can perform equivalent recognition to popular distributed neural networks (shown mathematically) but does not require rehearsing for learning or update. It can also be placed in deep network configuration. However, the focus of this work is not the details of deep networks. The focus is on how easy it is to create and update individual layers. This is an important bottleneck because ultimately creating and updating layers are a problem whether networks are in a deep configuration or not. We use a small laptop running matlab as a microenvironment to reveal data limits determined by limited memory and processing speed. Within this microenvironment we show our approach accepts the largest datasets. Ultimately we encountered limits of the matlab routines to generate random numbers before the limit of our algorithm.

*Keywords:* Localist, Globalist, Distributed Weights; Expectation Weights; Recall, Big Data

# 1 Introduction

Our overarching goal in designing our biologically motivated framework is to endow networks with the most flexibility (ability to quickly learn including "one-shot-learning", modify, and use new patterns as they are encountered in the environment) and recall (describing or predicting the inputs associated with a recognizable pattern). Along the way we find this method is also efficient for big data which is the focus of this paper. Although we have some very promising results, it is important to keep in mind this approach is still in its infancy. It has not been given the same amount of research and development efforts as distributed methods have over the last 60 years. It can be shown mathematically to perform the same recognition [1], however certain developments such as methods to determine deep networks are still in development. It requires a completely new formulation because the form of information stored in weights is different.

## 1.1 Background

The form of information determines strategies and connection weights required for recognition, memory, and further processing. Certain forms may be more optimal for big data than others. Popular state of the art methods for big data are "globalist" using distributed feedforward weights.

Such methods have been integrated with deep learning hierarchy which helps with big data applications and improves performance. However, deep hierarchy can be implemented using other strategies and deep learning has not significantly changed difficulties with globalist networks' ability to recall, update, or ease of learning within a layer. Thus, it remains relevant to evaluate how different strategies are resilient in learning within single layers.

Globalist methods learn weights that include uniqueness information by using gradient descent to learn distributed weights [1]. Subsequently recognition involves a relatively simple equation (eq 1)


Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.312

during recognition. To better describe: uniqueness, globalist, and alternative localist methods, let's begin by reviewing the standard notation for pattern recognition. Let vector $\vec{Y}$ represent the activity of a set of output neurons, classes, or nodes individually written as $\vec{Y} = (y_1, y_2, y_3, \ldots y_h)^T$. Vector **X** represents sensory nodes that sample the environment, or input space to be recognized, and are composed of individual features $\vec{X}=(x_1, x_2, x_3, \ldots x_n)^T$ which form the basis of recognition. The input features can be sensors that detect edges, lines, frequencies, kernels, and so on, or may be hidden nodes determined by unsupervised learning methods. Based on distributed weights **W**, globalist-feedforward methods solve the recognition relationship:

$$\vec{Y} = W\vec{X} \text{ or } \vec{Y} = f(W, \vec{X}) \quad (1)$$

This approach represents a feedforward architecture because the direction of information flow *during recognition* is feedforward: one-way from inputs to the outputs. **W** represents a feedforward matrix of distributed weights or parameters that associates inputs and outputs. $W\vec{X}$ calculates the output using the feedforward weights and inputs. Globalist networks can be found in the literature embedded within different algorithm optimizations, for example: single-layer Perceptrons [2], multilayer Neural Networks with nonlinearities introduced into calculation of $\vec{Y}$ [3], and machine learning methods such as Support Vector Machines (SVM) with nonlinearities introduced into the inputs through the "kernel trick" [4]. Although these algorithms vary in specifics such as nonlinearities determining the function *f*, they share the commonality in that recognition involves a feedforward transformation using **W** during recognition. The learning of **W** for feedforward models is global because the weights include all of the information necessary to perform recognition with one multiplication. This includes how important each input feature is to the node compared to other nodes. In order to obtain this information, the learning algorithms associated with feedforward methods perform a gradient descent (an iterative minimization of error) using all the patterns in the training set. Moreover, the training set patterns need to be repeatedly rehearsed in an Independent and Identically Distributed (*iid*) Order with fixed frequencies and random order to correctly determine the overall relevance of each input and uniqueness information. This avoids catastrophic interference and forgetting [5,6]. However, the *iid* criteria make it difficult to quickly incorporate new information into the network as the organism encounters it. Moreover, even if a suitable mechanism of rehearsal is implemented, it faces capacity and time limits. We will show this with our big data demonstrations.

Some authors argue slow changes such as those using online learning or stochastic gradient descent, strategies give the neural networks greater capability to change weights e.g. [7]. To implement gradual learning, instead of explicit rehearsing it is assumed the environment is *iid*. But this is a strong assumption that is easily broken. For example if one spends winter at home, or is incarcerated for an extended period of time, the environment changes and environment presentation changes. Online methods would forget how to recognize outdoor scenes or missed loved ones. Moreover, online learning does not change the fact that weights may eventually change extensively – in a global fashion - even with a small change in a pattern [1].

Since a life-like environment cannot be assumed to be *iid*, patterns must be rehearsed. Subsequently updating methods based on feedforward connections require structures that store, recall, present in random order, and rehearse to *iid* specifications all previous patterns (the training set). This requires complex mechanisms and additional resources such as memory for all previously seen patterns that are costly to implement and may be especially unfeasible with neurons. Moreover the requirements become more extreme with big data.

The root of difficulties using globalist methods is that feedforward weights incorporate uniqueness information and must change weights globally as the network changes. In contrast, storing and computing information based on expectation allows localized modification of weights and avoids the learning difficulties associated with global modification. In other words, in a network directly based on local expectations, a modification of expectation will only require changes in the expectation. This

is observed in the difference between how the localist and globalist neural networks must learn new data.

Regardless of network type (globalist or localist) neurons have patterns which most-optimally activate them. Let's label neuron-nodes using their label and optimal pattern, for example node $y_A$. Its optimal input pattern is $\overrightarrow{X_A}$ where pattern "A" is represented by the inputs with values $\overrightarrow{X_A} = (x_{1a}, x_{2a}, x_{3a}, \ldots x_{na})^T$. Using globalist (feedforward) distributed weights an optimal pattern exists for neurons, however it is not simple to determine this pattern by looking at the weights because as we argue such information is mixed with uniqueness information and is subsequently not readily available or recallable [8-10]. The optimal pattern is the basis of our localist-expectation memory weights.

## 1.2 Uniqueness: globalist vs localist; distributed vs expectation weights

We suggest a solution to this problem by using gradient descent during recognition to find neuron activation instead of using gradient descent during learning. Our approach learns input output characteristics without the uniqueness information. In our method the uniqueness information is calculated separately during recognition, ultimately solving the same equations. Since uniqueness is not incorporated into the weights, weights are much easier to learn. We call the weights "expectation weights" and we use a localist Hebbian-like learning strategy. Both methods (globalist & localist) store different information and use different strategies, but ultimately solve the same equations.

Let's define the difference between the forms of information and weights: distributed and expectation weights; then the differences between globalist and localist learning and their relationship with uniqueness. Localist describes the relationship between an input and the output of a node which does not depend on any other nodes, inputs and outputs. For example, suppose there is an input that represents legs and an output node that represents zebra. A localist relationship for zebra and legs is that it has *4* legs. In a localist network, the value *4* is the weight that represents the connection strength between legs and zebra. This represents a prototype description of a zebra (node), which we call expectation, where it doesn't matter whether there exist other animals (other nodes) that also use the same input (and may also have *4* legs such as dogs, cats, rats, giraffes, etc., or may have none, *2, 6, 8* or any other number legs). The weight between zebra and legs remain the same regardless of other nodes. The same is true with zebra and stripes, and every other possible node and inputs within a localist network.

However zebras are distinctive (among *4* legged animals) in that they have stripes. Subsequently stripes may be more important for recognizing zebra than many other features such as legs, ears, eyes etc. Subsequently an elevated status of unique information (such as stripes) enables efficient recognition regardless of whether the network is localist or globalist. However if the uniqueness is incorporated into weights, the weights no longer maintain their expectation qualities. Moreover learning that incorporates uniqueness cannot be localist by definition because it depends on other patterns in the network. Many authors have observed this: whereby using globalist distributed weights localist expectation meaning is lost. It is not easy to discern meaning of individual neurons (recall) in globalist networks while localist networks can preserve meaning e.g. [8-11]. This is due to the encoding of uniqueness [1].

However in reviewing the literature, definitions of localist, distributed weights, and coding are not always straightforward. Some researchers describe the differences between globalist and localist models by how they are labeled. For example the interactive activation model of [12] is considered localist because all nodes are labeled and have meaning [11,13]. Thus researchers often suggest that the presence of hidden nodes or whether hidden nodes are labeled determine whether models are localist [14]. We argue that whether hidden nodes are endowed with meaning (commonly by the network creator) or basing the definition on the existence of hidden nodes does not get to the core of the distinction between globalist and localist. Our assertion is that in localist networks, meaning (the optimal pattern) can be discerned for each layer by evaluating output nodes based on their inputs and

expectation weights (only possible because they do not contain uniqueness information). From our perspective even if a node is given meaning by whatever external method, if it is not possible to discern meaning directly from the nodes' weights and inputs, it contains uniqueness information and is not localist. Alternately, if a node has no meaning attached to it (eg hidden node), but the meaning can be discerned directly from the weights and inputs, this is a localist node. Moreover we show by examples [1] the distinction between globalist and localist fundamentally exists even in a single layer, when there are no hidden nodes.

Following a localist learning strategy and definition, neurons only need to know about their immediate surroundings to learn. Hebbian Learning [15] is summarized and follows the strategy "what fires together wires together". Each neuron only needs to know which inputs (dendrites) are active when outputs (axons) are active and weigh the inputs relative to the outputs. Each neuron can then rely solely on mechanisms contained within itself (local mechanisms) in order to determine how to change its weights. Thus localist-Hebbian learning often describes a process to find weights without uniqueness information.

In summary, globalist-feedforward networks solve recognition with one multiplication per layer and in order to recognize correctly within one multiplication, they encode uniqueness information in the weights (more than just expectation information). To incorporate the uniqueness information into weights, globalist methods use a Gradient Descent (GD) error-driven mechanism during learning, which rehearses the patterns of the training set in a uniform identical and independently distributed (*iid*) fashion to learn distributed weights that precisely incorporate how relevant (unique) each input is. Thus in feedforward-globalist networks, distributed weights additionally depend on input and output nodes other than the immediate input and output node that they connect. This is why error-driven learning is referred to as global learning, the networks are called globalist, and the weights are described as distributed eg [11].

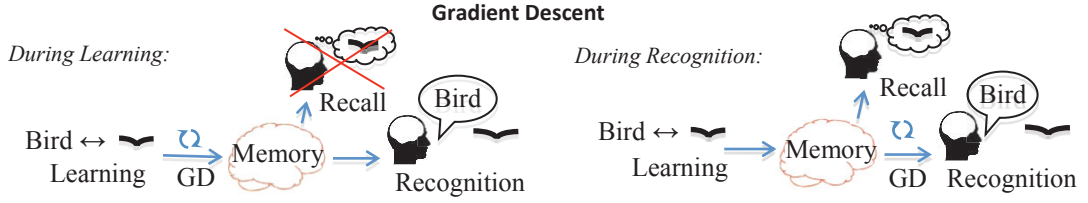


**Figure 1:** Globalist Gradient Descent (GD) During Learning (*left*) Vs Localist During Recognition *(right)* **Left:** memories are stored in a form that includes uniqueness information optimized for feedforward recognition. However this is not optimal for flexible updating and recall. **Right:** memories are stored in expectation form without uniqueness which is easier to learn, update, and recall. Uniqueness is determined during recognition.

## 1.3 Determining uniqueness information during recognition

The proposed model uses a gradient descent mechanism *during recognition* to determine how unique a piece of information is, based on the other nodes that are currently active (e.g. the other animals that are also being considered) and modulates the relevance of the input (e.g. stripes) accordingly. In effect the proposed model is doing what feedforward learning algorithms do during learning (modulate weights based on uniqueness) but during recognition (modulating inputs based on uniqueness). The gradient descent of the proposed model doesn't learn weights (weights do not change during recognition) but determines activation based on uniqueness.

It is possible derive the equations of our model from the feedforward model. Unfortunately due to space limitations this cannot be shown here but is available elsewhere [1]. The localist recognition method proposed here can be considered interchangeable with globalist feedforward neural networks because they ultimately solve the same equation 1. However it does not require explicitly finding **W**

and instead uses localist weights. In fact, it is shown that globalist weights **W** are the pseudo-inverse of localist weights **M** and visa versa [1]. Subsequently this equation can be written as:

$$\vec{X} - M\vec{Y} = 0 = \text{Error} \quad (2)$$

Our localist method operates *during recognition*. When output nodes in **Y** are active, a pattern is generated from stored weights in **M**. The generated pattern is $\mathbf{M}\vec{Y}$ and this pattern is subtracted from the incoming input pattern $\vec{X}$ (through inhibition). When a correct recognition is made, the stored pattern that is activated ($\mathbf{M}\vec{Y}$) will account for the input pattern ($\vec{X}$) and they will annihilate each other making the sum **0**, see equation 2. This shares some similarity to other generative models however the difference is this process occurs *during recognition* NOT *during learning*: to find activation not to learn weights.

Although equation 2 describes the basic relationship between inputs $\vec{X}$, output activation $\vec{Y}$, and feedback weights **M**, it does not provide a way to project input information to the outputs and determine activation. In other words, in this form it is still not possible to calculate $\vec{Y}$ given $\vec{X}$ and **M** without calculating the pseudo-inverse of **M** (**W**). To avoid calculating **W**, we use a gradient descent mechanism using "auto-inhibitive" connections to converge to equation 2. Two versions of the final equations of our model are found in table 1. See the "localist", "during recognition" section.

**Table 1:** Comparing pseudo-code of globalist and localist method

| Method | During Learning (given training set) | During Recognition (given $\vec{X}_{\text{test}}$) |
|---|---|---|
| **Globalist** | 1. Organize training set in *iid* order & frequency<br>***Gradient Descent:***<br>2. Determine (Initialize) number of nodes<br>3. Iterate until learning error < threshold<br>*i.* present an *iid* pattern from training set<br>*ii.* calculate delta and error<br>*iii* update weights **W** | 1. Calculate: $\vec{Y} = W\vec{X}_{\text{test}}$ |
| **Localist** | 1. Cumulative Hebbian Learning<br>$M(n)_{ij} = \frac{1}{n}\sum_n \frac{x_i}{y_j}$ *n:* presentation number<br>***or***<br>1. Average the training set using labels:<br>**M**=μ($\vec{X}$\|$\vec{Y}$) | ***Gradient Descent:***<br>1. Initialize $\vec{Y}$ activity (or use previous values)<br>2. Iterate until $d\vec{Y}$ < threshold<br>*i.* $d\vec{Y} = -M^T\left(M\vec{Y} - \vec{X}_{test}\right)$<br>*ii.* $\vec{Y}_{\text{new}} = \vec{Y} + d\vec{Y}$<br>***or***<br>*i.* $\vec{Y}_{\text{new}} = \vec{Y} + d\vec{Y} = \left(\frac{\vec{Y}}{\sum_{j=1}^{N} M_{ji}} M^T\left(\frac{\vec{X}_{test}}{M\vec{Y}}\right)\right)$ |

These equations may look similar to a predictive coding learning algorithm which performs a gradient descent to minimize energy, has bidirectional connections, iterative function, and has generative properties e.g. [16,17]. Unlike a learning algorithm however, the gradient descent here occurs *during recognition* and no weights are learned via this gradient descent. Instead activation $\vec{Y}$ is determined by gradient descent. To repeat, learning is defined as the phase when weights change. Minimization of error is not synonymous with learning. This model minimizes error by changing activation without changing weights. Minimizing energy during recognition finds activation $\vec{Y}$, the same $\vec{Y}$ that is found using feedforward weights **W** in equation 1 (but instead using **M)**. This allows simpler expectation weights **M** to be used.

In this model only the currently presented pattern needs to be iterated. In feedforward learning, all training patterns need to be repeatedly rehearsed in independent and identically distributed form (*iid*) for learning. Thus overall computational costs are less compared to feedforward learning [1] and are further studied within our big data paradigm in table 2 .

# 2 Demonstrations

We evaluated globalist and our localist methods given big data and asked: 1) What is the largest network that can be implemented with each respective method? 2) How easy is it to update the network with a new node or modify an existing pattern?

For simplicity we determined the training patterns by randomly generating characteristic patterns to evaluate the limits of algorithms. This allowed us to quickly generate datasets of any size. We used one pattern per class to most efficiently generate and test large data.

We ran the tests on a small Intel Celeron Quad Core 2 gigahertz laptop with 4 gigabytes memory running Windows 8.1 and Matlab 2009a. This is by no means the most optimal machine or software. Our system is clearly not optimized for big data. But our goal is to show when and how processing breaks down with big data and how our approach measures up. Since all of the algorithms are bound by the same limitations, we obtain a relative measure of difficulty with large data, but on a smaller scale.

We compared SVM, our pseudo-inverse method to convert localist to globalist weights for feedforward learning, and our gradient descent recognition methods. The SVM code used is widely used and freely available [18]. We trained and tested all algorithms on the same training data and only reported when all algorithms performed 100%. We recorded and reported processing times using the internal clock functions of matlab: tic-toc.

## 2.1 Comparing largest networks possible by method

**Table 2:** Comparing network performance based on data size and method

| Data size | | | Method Comparison | | |
|---|---|---|---|---|---|
| Nodes | Inputs | Matrix size | SVM | Globalist: during training | Localist: during testing* |
| 500 | 1000 | 500,000 | 5592.02 sec | 5.057 sec | 0.0095 sec per test |
| 1000 | 10,000 | 10,000,000 | Out of memory | 55.36 sec | 0.1348 sec per test |
| 2000 | 10,000 | 20,000,000 | Out of memory | 547.19 sec | 0.2449 sec per test |
| 6000 | 12,000 | 72,000,000 | Out of memory | Out of memory | 0.8305 sec per test |
| 8000 | 15,000 | 120,000,000 | Out of memory | Out of memory | 1.3613 sec per test |
| 9000 | 20,000 | 180,000,000 | Out of memory | Out of Memory | ? ** |

* time is measured during recognition ** Result not measurable: random number generator out of memory

The time it took for SVM to learn quickly increased with big data, and gave an "out of memory" error. To overcome this limitation we used an alternative method to find the distributed weights using the pseudo-inverse of the training data [1]. First we determined M and then calculated the pseudo inverse of the localist weights to obtain the distributed weights. This provided a faster way to determine distributed weights within our training paradigm and the time required is shown in the globalist section. Although faster than SVM learning, the amount of time also grew super-linearly with matrix size. Between matrix sizes of 10,000,000 and 20,000,000 entries (an increase of 2x), the training time required increased almost 10x.

Learning within the localist method was quick because localist weights are the expected patterns without uniqueness. The dataset generated are characteristic patterns which ultimately are the expectation weights. Thus the time to learn the data is the time it takes to store the generated data into memory. Moreover, each node can be learned independently.

Since gradient descent occurs during recognition using the localist method, the processing-intensive step is during recognition. Thus we reported "localist testing": the number of seconds required to recognize in table 2 and "computational costs during recognition" in figure 2. The time required steadily increased with matrix size but the increase was linear with the size of matrix. This is good news for big data processing. When considering both the cost of learning

and cost of recognizing, the localist method was by far the most efficient. Note that entries beyond the matrix size of 20,000 are missing for globalist methods in table 2 and figure 2. This is because the globalist methods became too slow, consumed all available memory, and gave an "out of memory" error. In contrast, we did not find a limit for our localist method using these tests. The matlab routine that generates random numbers gave an "out of memory" error beyond the matrix size of 120,000,000 - before our algorithm did!

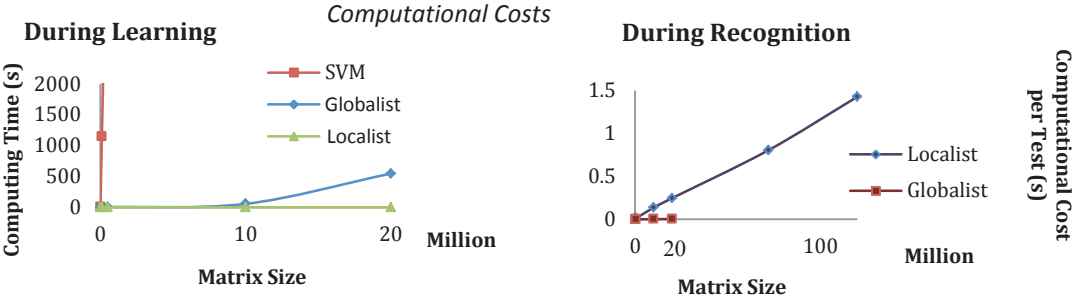


**Figure 2:** Costs During Learning (*left*) vs During Recognition (*right*). **Left**: Learning costs were super-linear for globalist methods which quickly became overwhelmed by large data, while localist learning costs for M were small. **Right:** Although costs during recognition (of feedforward activation) were small for globalist methods, learning costs only allowed them to sustain matrixes of up to 20 Million. Localist costs grew modestly in a linear manner and were able to sustain larger matrixes of up to 120 Million (and maybe even more).

## 2.2 Modifying and updating big data networks

There are two ways to update networks; the first is modifying the expectation of a node. In the localist model the nodes and their weights represent the expected pattern. To change the expectation it is sufficient to change individual weights directly to match the new expectations. Since uniqueness information does not have to be changed or recalculated (uniqueness information is calculated during recognition) these are all the changes that are needed. On average the recognition times remain the same. Thus there are virtually no costs (besides the cost of changing an entry in memory) and it does not depend on network size. In the globalist network the uniqueness information needs to be recalculated. Any entry, even a small change in expectation, can affect the whole network and the network needs to be retrained. Thus the costs of updating include relearning, depend on network size and show super-linear growth. In our random vector example, the costs for globalist modification was an additional 56.02 sec for size 10,000,00 matrix and 545.72 sec for size 20,000,000 matrix by repeating the original learning (variability due to random nature of data and gradient descent initial conditions).

The other way to update is to add a new node to the network. The learning cost to the localist network is infinitesimal since in learning each node is independent of others (localist). The cost is simply generating a new (random vector in this case) and adding it to the matrix. The cost for one node is also unlikely to be noticeable during recognition because of the slight (linearly) increased recognition times. For example a 10,000,000 vs a 10,010,000 matrix may require on average 0.1369s per test vs. 0.1374s per test (note that since our data sets are randomly generated there are variations). The costs to globalist methods are recalculating uniqueness information for the whole network which includes a super-linear increase in costs with matrix size. The cost of retraining on average is 56.86 s for matrix size of 10,010,000, 552.1s for 20,010,000, and grows super-linearly, limiting big data. Although for globalist networks additional *costs during testing* are minute, the *learning costs* eclipse them.

# 3 Conclusion

We have demonstrated that a neural network mechanism that determines uniqueness during recognition may be more optimal for big data. Complex learning is not required; learning is fast and more scalable with size. Big data costs are borne in recognition times but increases with big data are only linear. On the other hand traditional methods that determine uniqueness during learning and incorporate it into distributed weights, grow super-linearly with data size. Moreover even small modifications of the networks require the recalculation of uniqueness and re-incur these learning costs. Thus we have shown a very promising new method to address large data.

It is also important to note that neither the networks nor the computer were optimized in any way. There are several ways of optimizing, updating: hardware memory, CPU, adding GPU's, more efficient software, and finally optimizing the networks by incorporating hierarchy as with deep networks. However the optimizations are likely to benefit both globalist and localist methods.

Although we did not focus on deep structures in this paper, the same big data issues will occur within individual layers regardless of whether they are in a hierarchy. For example if there are 8,000 labeled patterns in the network, the top network may have 8,000 top-level nodes.

Future work will focus on deep learning since deep learning clearly helps big data processing, especially in scenarios with a large number of input features. Deep learning optimizes sparseness, orthogonality between nodes [16,19], but this is not specific to globalist methods. Our method solves the same equations and also benefits from deep hierarchy. Moreover localist networks can facilitate deep learning [19]. Since our localist weights represent expectations, they can be used to explicitly quantify overlap between nodes and recreate new ones without rehearsal. New nodes can serve as inputs in a hierarchy and minimize common expectations, reducing overlap, and computational costs during recognition. The reduction in computational costs is analogous to a reduction in learning costs of feedforward networks . The difference is that a localist (auto-associative) network in [19] is used to help learn globalist networks. In our work, solely a localist network (that performs gradient descent) is used. Thus a completely localist strategy can be used to optimize sparseness, increase orthogonality, and also perform recognition. The benefits of more-sparse and smaller layers for our localist method are realized during recognition, with reduced recognition times (analysis and simulations in work to be published). This work is the topic of our future paper.

Analogous to learning thresholds and parameters in distributed methods, we can also optimize our gradient descent by when to stop. We stopped our simulations when the sum of dy's of all the nodes in the network was below a predetermined threshold (0.01). This achieved 100% correct recognition for all of our test cases. The gradient descent can alternatively be stopped when one node quickly predominates; For example, when a node's dy is increasing while other nodes are decreasing and its activation is above others. Such optimizations may decrease processing times while preserving performance.

In summary, our method is able to add new data in a life-like manner, learning information as it becomes available, even briefly, without rehearsal. Adding new data to globalist methods requires relearning uniqueness information and the cost increases super-linearly with the size of the network. Adding new data (and new node) using our method does not incur such costs during learning because other existing neurons (and their weights) do not need to be modified since the weights are without uniqueness information. Subsequently our results suggest the localist method (using gradient descent during recognition to determine uniqueness) is a promising approach to address big data problems.

# Acknowledgment

Thank you to Asim Roy for discussions, suggestions, and encouragement to implement this method in the context of big data.

# References

[1] Achler T (2014) Symbolic Neural Networks for Cognitive Capacities, Special issue-Journal Biologically Inspired Cognitive Architectures.

[2] Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain, Psychological Review 65 6:386-408.

[3] Rumelhart DE, McClelland JL (1986) Parallel distributed processing: explorations in the microstructure of cognition, V1.

[4] Vapnik VN (1995) The Nature of Statistical Learning Theory, New York: Springer Verlag.

[5] McClelland JL, McNaughton BL, O'Reilly RC (1995) Why there are complementary learning systems in the hippocampus and neocortex, Psychological Review 102:419-57.

[6] McCloskey M, Cohen NJ (1989) Catastrophic interference in connectionist networks: The sequential learning problem, The Psychology of Learning and Motivation, 24, 109-165.

[7] Bottou L, LeCun Y (2004) Large Scale Online Learning, Adv in Neural Information Processing Systems (NIPS2003) 16.

[8] Fodor JA, Pylyshyn ZW (1988) Connectionism and Cognitive Architecture: A Critical Analysis, Cognition 28, 3–71.

[9] Marcus GF (1998) Rethinking eliminative connectionism, Cognit. Psychol. 37, 243—282.

[10] Sun R (2002) Duality of the Mind: A Bottom-up Approach Toward Cognition, Mahwah, NJ: Lawrence Erlbaum.

[11] Roy A (2012) A theory of the brain: localist representation is used widely in the brain, Front. Psychol. 3:551 10.3389.

[12] Rumelhart DE, McClelland JL (1981) Interactive processing through spreading activation, in C. Per-fetti & A. Lesgold (Eds.), Interactive processes in reading, Hillsdale NJ: Erlbaum.

[13] Plate T (2002) Distributed representations, in: Encyclopedia of Cognitive Science, Nadel L. ed London: Macmillan, 2.

[14] Page M (2000) Connectionist modeling in psychology: a localist manifesto, Behav. Brain Sci. 23, 443–512.

[15] Hebb DO (1949) The Organization of Behavior, New York: Wiley & Sons.

[16] Olshausen BA, Field DJ (1996) Emergence of simple-cell receptive field properties by learning a sparse code for natural images, Nature 381: 607-609.

[17] Rao RP, Ballard DH (1999) Predictive coding in the visual cortex, Nat Neurosci.

[18] Schmidt M (2006) SVM Software Web: http://www.cs.ubc.ca/~schmidtm

[19] Hinton GE, Osindero S, Teh Y (2006) A fast learning algorithm for deep belief nets, Neural Computation 18(7):1527–1554.