Neural Field: Supervised Apportioned Incremental Learning (SAIL)

Justin Lovinger, Iren Valova
Computer and Information Science
University of Massachusetts Dartmouth
N. Dartmouth, MA, USA
jlovinger, ivalova@umassd.edu

***Abstract*— When a person learns, they observe and interact with their surroundings, and monitor the outcome of these interactions. During this process, the brain only examines single snapshots of information. It does not need to continuously revisit past instances of time to retain learned information. Supervised neural networks, as much as they resemble the human brain, do not learn well incrementally. The standard multilayer perceptron, radial basis function network and others, have a common problem of losing learned information when additional data points are presented. In this paper, we develop a supervised learning neural network that converges on new information without overriding previous knowledge. The proposed Neural Field can operate in real time environments where a data point is seen once and never revisited, or chase a function that changes over time without continuously relearning on a full dataset. A dataset can be presented in portions and the network will attain knowledge incrementally. With a static ordered grid of radial basis function neurons, the Neural Field both learns incrementally and generalizes effectively.**

Keywords— supervised learning; classification; regression

I. Introduction

In the early 1900s, Ivan Petrovich Pavlov discovered the famous Pavlovian reflex, and showed that biological brains can learn from positive and negative reinforcement. His theory on classical conditioning has remained an integral component of our understanding of learning. Reinforcement learning (RL) aims to emulate classical conditioning in artificial agents. Here, we examine RL w.r.t. Q learning [1, 2, 3]. Supervised learning algorithms commonly learn $Q: S \times A \to R$ [3, 4, 5]. However, an RL agent only sees a small section of the problem space at any time. As such, a RL agent must learn Q piece by piece, without a known dataset.

However, most supervised learning algorithms, such as the popular multilayer perceptron (MLP), rely on the repeated presentation of a series of data points from a well-sampled problem space, i.e. a known dataset. Every data point is presented once during a learning iteration, and then repeated until convergence. If, instead, we imagined a sliding window, an analogy for RL, moving over the dataset, and only a small number of data points could be seen through the window, the algorithm would retain only the data points that the window had recently shown [5]. This problem has been solved by storing every data point that the window passes over in a growing dataset [5]. However, this solution is not feasible for long learning sessions with continuous and infinite potential data points, as the dataset quickly expands beyond finite computational memory.

The proposed **n**eural **f**ield: **s**upervised **a**pportioned **i**ncremental **l**earning (NF:SAIL) algorithm is designed to overcome this limitation. By creating a static, ordered grid of radial basis function-like nodes, the NF:SAIL can retain learned data in one portion of the input space, even after repeated presentation of data points in another portion of the input space. This spatial storage mechanism allows the NF:SAIL to learn functions incrementally, without repeated presentation of a series of data points from a well-sampled problem space. The cascade correlation network (CC) [6] is similarly capable of retaining previously learned information due to its use of static weights. However, CC is strictly a batch algorithm, which requires a known dataset to learn. Using a cache mechanism similar to [5], CC has found only limited success with online RL [7].

NF:SAIL is designed to meet four requirements. First, NF:SAIL must retain learned information from a portion of the input space with relative accuracy, even after repeated presentation of data points from different portion of the input space. Second, NF:SAIL must be capable of approximating continuous functions, as opposed to an algorithm used only for classification that outputs discrete classes. This can be formalized as: the output of the network, defined by $f: \Re^n \to \Re^m$, where $\varphi(\vec{x} + \vec{\varepsilon}) \neq \varphi(\vec{x})$ for a small vector $\vec{\varepsilon}$, and an input $\vec{x}$, i.e. the network output changes, even with a small change to the input vector. Third, NF:SAIL must generalize to unknown outputs, assuming the input vector for the unknown data point is in a portion of the input space that the algorithm has learned. This last requirement is simply the embodiment of supervised learning. Fourth, NF:SAIL must learn incrementally without knowledge of a dataset beyond the data point it is presented. This strict online learning requirement allows NF:SAIL to operate directly in real time environments.

II. The Neural Field Algorithm

In order to learn incrementally while retaining previous information, function information must be stored in the local

input space around individual neurons, as opposed to globally on all neurons. With this local storage, we can learn part of a function in one portion of the input space, and then learn another part of the function in another portion of the input space, without overriding the obtained knowledge. The key to this local storage is the neural contribution, which is mathematically formalized as a radial basis function. Given an input vector $\vec{x}$, a neuron center $\vec{N}_n$, and a Gaussian variance v, our radial basis function computes the neuron contribution c_n as:

$$c_n = e^{-\left(\frac{\|N_n - \vec{x}\|^2}{v}\right)} \quad (1)$$

With this Gaussian representation, the neuron contribution c_n approaches 1 as the distance between the neuron center $\vec{N}_n$ and the input vector $\vec{x}$ approaches 0. Conversely, the neuron's contribution approaches 0 as distance approaches ∞. The variance determines how quickly c_n approaches 0. This property makes the neuron contribution essential for storing function information in local neuron space.

A. Learning the Output

The NF:SAIL output is calculated through the standard radial basis function (RBF) rule [8, 9]. Normalization by total contribution is added for a smooth transition between learned data points to improve generalization. Given the neuron output $\vec{O}_n$, and m network neurons, the network output $\vec{\varphi}$ is:

$$\vec{\varphi} = \frac{\sum_{i=0}^{m} \vec{o}_i c_i}{\sum_{i=0}^{m} c_i} \quad (2)$$

Instead of directly calculating the matrix inverse to obtain the neuron outputs [8], we use an iterative learning rule to allow for incremental learning. We adjust $\vec{\varphi}$ towards a given target vector $\vec{t}$, without overriding parts of the function in another portion of this input space, by scaling the neurons rate of change by its percent of contribution.

$$\Delta\vec{O}_n = \mu \frac{c_n}{\sum_{i=0}^{m} c_i}\left(\vec{t} - \vec{\varphi}\right) \quad (3)$$

where μ is a user provided learning rate, $0 < \mu \leq 1.0$. With this rule, the closer a neuron is to the input vector $\vec{x}$, the faster it is adjusted, which prevents information stored in distant neurons from being overridden by new data points.

B. Positioning Neurons

Since the goal is to store function information spatially, neurons should cover the input space. Clustering is often employed for this purpose. However, most clustering methods require a known dataset, contradicting our requirements. Incremental clustering can be performed by a self-organizing map (SOM), but the movement of neurons in a SOM rapidly overrides previously learned data points. Instead, an ordered grid of static neurons allows complete coverage while effectively retaining learned information. Fig. 1 shows 9 neurons covering a 2-dimensional input space spanning from -1 to 1. Points represent neuron centers. Circles visualize a low neuron contribution of 0.01. Positioning neurons with overlap allows for generalization, as in the tile coding and RBF methods popular in reinforcement learning [3].

With the 9 neurons in Fig. 1, very limited information can be stored. Given two data points with similar input vectors and dissimilar output vectors, only one of these points could be learned on a small number of neurons. Since only a small amount of information can be stored in this portion of the input space, we say it has a coarse resolution. To improve the resolution, and, therefore, the amount of information that can be stored, we decrease the distance between neurons, thereby increasing the number of neurons in a given portion of the input space (Fig. 2). This greater information density comes at the cost of greater computation and the risk of over fitting, a common tradeoff in supervised learning.

C. Accessing Neurons

The naive approach of pre-generating neurons to cover an input space presents a number of problems. First, the number of neurons required increases exponentially with the dimensionality of the input space. Iterating over every neuron to acquire the output vector becomes unfeasible with high dimensional input spaces. Second, the many neurons required to cover a high dimensional input space necessitate a large amount of memory. Third, it requires a known input space. However, many problems dynamically generate data points through real time experience or nondeterministic simulations. Such data points can unexpectedly appear beyond the bounds of a set input space. If neurons do not cover these unexpected data points, learning and generalization is compromised by the same principle of too coarse resolution covered in section 2.2.

The solution to these problems is conceptually simple and elegant. With static neuron positions and variance, we know the sphere of influence of every neuron before allocation in memory. The key is an algorithm to determine the positions of neurons around an input with contribution greater than some threshold c_t. First we must know the distance d at which a neuron with variance v has a contribution of c_t:

$$d_t = \sqrt{v \log\left(\frac{1}{c_t}\right)} \quad (4)$$

Given a hypersphere of radius d_t, centered on an input i, and a grid of points spaced m apart, we can determine all points on the grid within the given hypersphere.

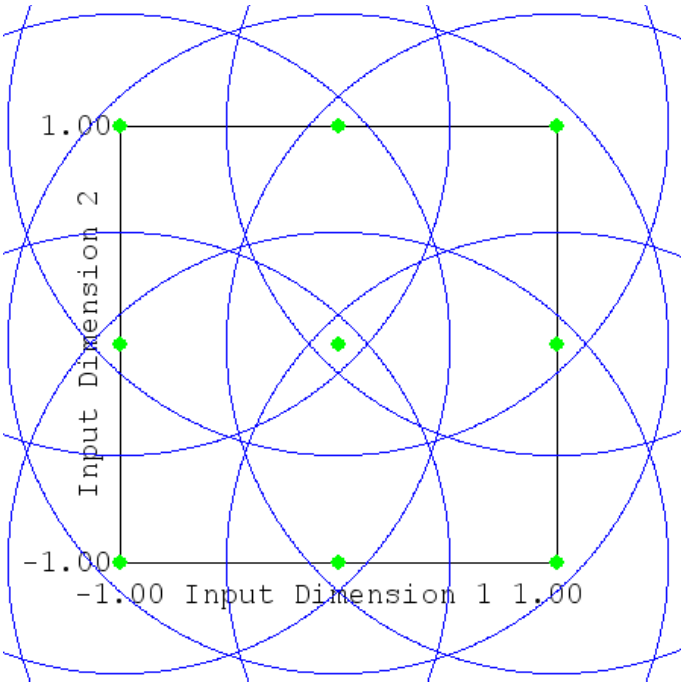

Fig. 1. Neuron positions in 2 dimensions

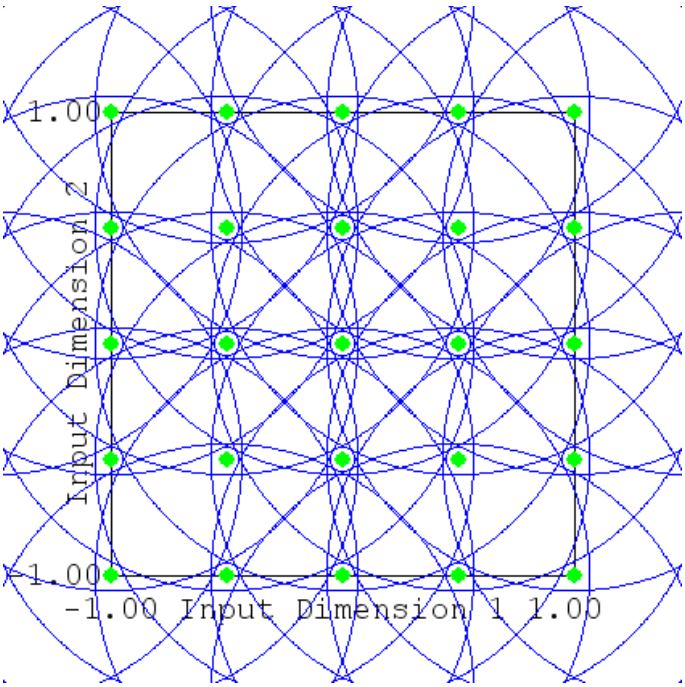

Fig. 2. Finer resolution neuron positions

Following the pseudocode in Table 1, each recursive iteration determines one coordinate for a set of points on the grid, then constrains the next function call according to this coordinate. Once the coordinates of the last dimension are determined, all coordinates are concatenated to generate a set of vectors for points on the grid within the hypersphere. This technique efficiently computes all positions without checking any points beyond the hypersphere. By calling this function with an input vector and distance threshold, we obtain the grid positions of all neurons that provide a significant contribution to the input.

With positions known, a hash table mapping position to neuron allows access to each neuron in constant time. This hash-field solves the time complexity problem. With a small extension, we solve the memory complexity and out of bounds problems. Instead of pre-generating every neuron, we wait until the neuron is required. If the neuron exists, we return it immediately, otherwise, we create the neuron and assign it to the hash-field before returning it. This procedure, commonly known as lazy evaluation, allocates only neurons in the space around inputs.

D. Adaptive Variance

As dimensionality of data increases, the maximum distance between data points and neurons also increases. In the worst case, for a given variance v, the furthest data point may have no neurons within the distance given by (4). The result is an inability to learn. Both increasing v and a finer resolution

TABLE I. ALGORITHM

GridPoints (center c, radius r)
If c contains 1 coordinate
  Let $s =$ closest first coordinate on $grid > (c_0 - r)$
  Let $e =$ closest first coordinate on $grid < (c_0 + r)$
  Return all coordinates from s to e spaced m apart

Set $P =$ an empty list
Set $x =$ closest first coordinate on $grid > (c_0 - r)$
While $x \leq c_0 + 1$
  Let $r2 = \sqrt{r^2 - (x - c_0)^2}$
  For partial point p in GridPoints $(c_{1..n}, r2)$ **do**
    Let $j = x$ concatenated with p
    Add j to list P
  End for
  Set $x = x + m$
End while
Return P

Call GridPoints (i, d_t)

---

solve this problem, but increase computation and memory usage. Instead, variance can adapt to the needs of a particular input, allowing efficient computation and learning on all data points. Given the distance to the closest neuron, d, and a user provided variance scaling factor s:

$$v = \frac{\left(d + s\log\left(1+d\right)\right)^2}{\log\left(1/c_t\right)} \tag{5}$$

where c_t is the contribution cutoff value.

With (5), every input is covered by at least 1 neuron. Fig. 3 and 4 depict adaptive variance. Squares represent inputs. Points represent neuron centers. Circles visualize a low neuron contribution of 0.01. Inputs very close to a neuron rely entirely on that output of that neuron (Fig. 3), while inputs far from any neuron rely on the combination of several nearby neurons (Fig. 4). The learning and generalization capabilities of the network remain the same, but unnecessary computation is avoided. As such, the advantage of guaranteed coverage of the input space is maintained for all resolutions, while computational and memory complexity is optimized.

III. Results and Comparative Analysis

These tests prove the four requirements for NF:SAIL, i.e. the ability to retain learned data points after repeated presentation of other data points, the ability to approximate functions, generalization to unknown inputs, and learning incrementally. In each experiment formulation, we conduct comparative tests with two incremental learning architectures: MLP and RBF. MLP is presented in its online learning mode, and is commonly used in reinforcement learning [4, 5]. RBF is chosen because it bears commonalities with NF:SAIL through its local output calculation. While there are many other kernel-based and supervised algorithms, we avoid batch learning systems due to their requirement for a previously known dataset, which contradicts the requirements of reinforcement learning. Each requirement is presented in its own subsection with appropriate datasets. The function approximation, classification, and regression tests feature a 3-fold cross validation method for the analysis of results.

A. Retention of Previously Learned Data

While effective classification or regression are essential for any supervised learning system, NF:SAIL is primarily designed for the ability to retain learned information after many iterations of learning new data points, without revising previous data points. We refer to the initially learned data points as stagnant data.

In the absence of standard testing methods to demonstrate this ability, we devise a test wherein a supervised learner is trained on one half of a dataset until convergence (error < 0.02), then the second half of the dataset until convergence. Finally, the learner is tested on the first half of the dataset, the stagnant data. If the learner has a low error on the first set, we say that it can retain stagnant data, otherwise, the learner forgets stagnant data. In order to isolate the learner's ability to retain stagnant data from its ability to generalize, we use datasets of randomly generated data points.

Each dataset consists of 10 data points. Each data point has a single input and a single output value. Each value is randomly chosen from a range of -1.0 to 1.0. This test is run 100 times with 100 different random datasets. The NF:SAIL is configured with a resolution of 0.005, a learning rate of 1.0, and variance scaling of 1. The MLP is configured with a hyperbolic tangent transfer function for the hidden layer and a linear transfer function for the output layer, a learning rate of 0.05, a momentum of 0.015, and 1 to 20 hidden neurons, increased until the MLP can converge. The RBF has 40 neurons, and uses the same output learning mechanism as the NF:SAIL, with a learning rate of 0.0125. However, the kernel

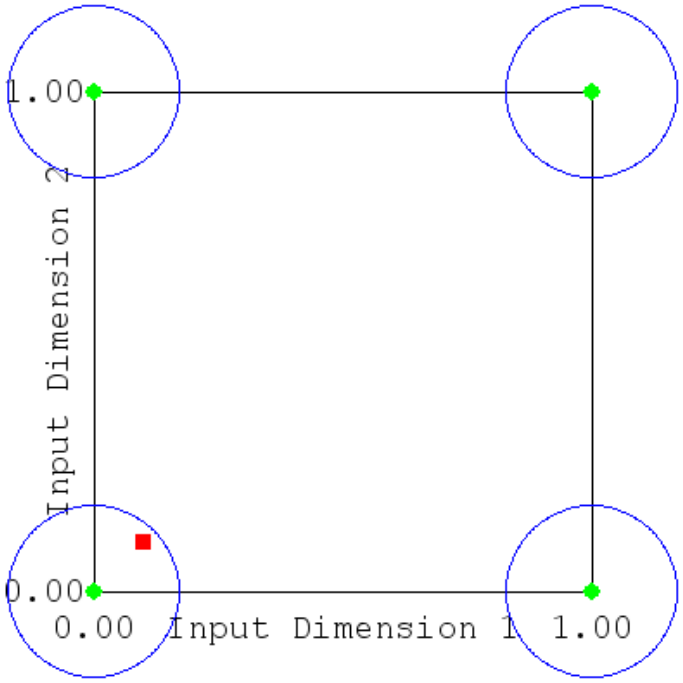

Fig. 3. Adaptive variance with input near neuron

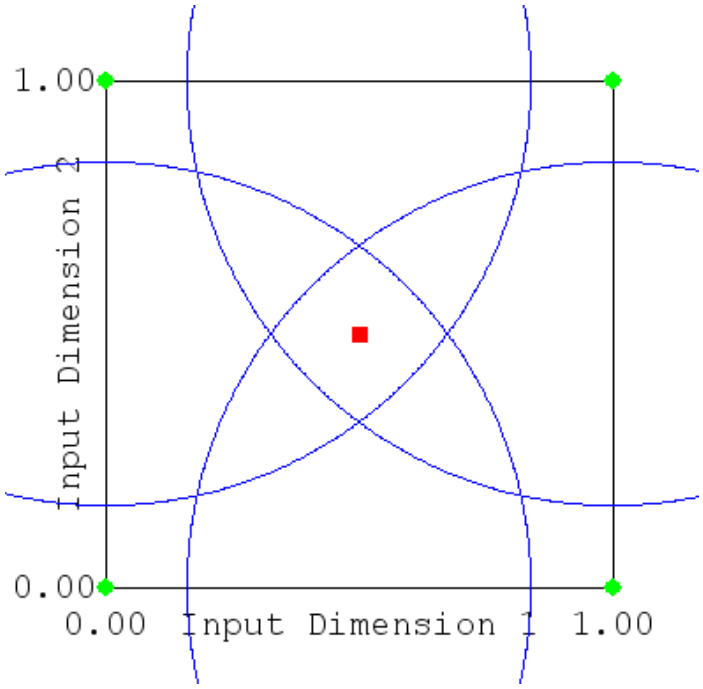

Fig. 4. Adaptive variance with input far from neurons

centers for the RBF are positioned with a self-organizing map (SOM) with a neighborhood of 2, a movement rate of 0.1 for the closest neuron, 0.0368 for the nearest neighbor, and 0.0018 for the second nearest neighbor. The mean retention error and standard deviation for each learner is presented in Table 2.

The proposed NF:SAIL algorithm shows exceptional ability to retain stagnant data, with a retention error near the convergence threshold. The MLP and RBF rapidly override stagnant data to converge on the new set of data. Despite the RBFs similar output and learning mechanism, the movement of neuron positions proves a major disadvantage over NF:SAILs static neuron centers. The detriments of MLP presented in [4] are only reinforced by these results. With the ability to consistently retain previously learned information, NF:SAIL can effectively learn as information is presented, without revisiting previous data points. An ability absent in the current state of the art for supervised learning.

Since the resolution of NF:SAIL determines how much information can be stored in a given portion of the input space, retention ability is also affected. Fig. 5 demonstrates this dependence by plotting the mean retention error against the resolution of a NF:SAIL. Mean retention error is determined with the same test as Table 1. Only resolution is adjusted. Despite noise caused by the stochastic nature of this test, the correlation makes it clear that finer resolution allows for greater retention of stagnant data. Furthermore, NF:SAIL, even with coarse resolution, shows an ability to retain data beyond both the compared algorithms.

B. Classification and Regression

To test the classification and regression abilities of the NF:SAIL, we perform 3 fold cross validation on the popular iris [10] and cancer [10, 11] classification datasets, as well as the California housing [12, 13, 14] and a sine regression dataset, and compare the performance of NF:SAIL to MLP and RBF. These use the same architecture as our previous tests, and hyperparameters are manually adjusted for each dataset. The sine dataset consists of 300 random inputs from 0 to $3p$ with corresponding outputs generated from the trigonometric sine function. The nonlinearity of the sine function makes it useful for benchmarking function approximation [14, 15]. The number of iterations until convergence, or a reached limit, is recorded for each fold, and the mean of these values is presented. Mean squared error for both training and testing sets is also recorded, and the mean of folds presented. Standard deviation (SD) is also presented for each metric. The results are shown in Table 3.

For the iris dataset, the NF:SAIL has a resolution of 0.25, and a set variance of 0.125. The MLP has 4 hidden neurons, a learning rate of 0.25, and a momentum of 0.1. The RBF has 30 neurons, a learning rate of 0.15, and a maximum of 5000 iterations. For the cancer dataset, the NF:SAIL has a resolution

TABLE II. RETENTION TEST RESULTS

	NF	MLP	RBF
Avg Retention MSE	0.0891	0.6687	1.1391
Standard Deviation	0.1357	0.4853	2.0981

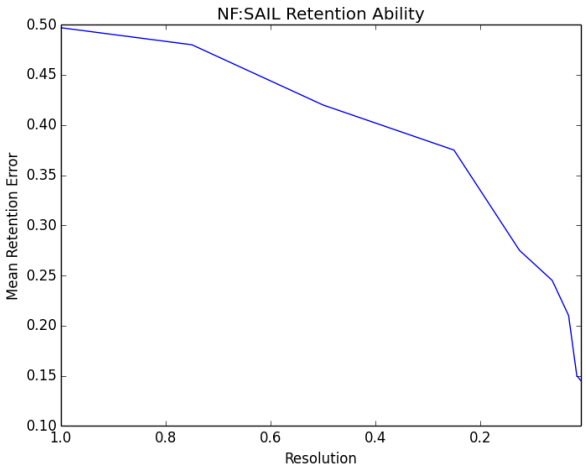

Fig. 5. Effect of resolution on retention ability

of 0.75, and variance scaling of 2. The MLP has 10 hidden neurons, a learning rate of 0.05, and a momentum of 0.01. The RBF has 3 neurons, a learning rate of 1.0, and a max of 100 iterations, which is chosen because no improvement in training error is seen after 100 iterations. For the California housing dataset, the NF:SAIL has a resolution of 0.5, variance scaling of 1, and a max of 10 iterations. The MLP has 10 hidden neurons, a learning rate of 0.1, a momentum of 0.025, and a max of 1000 iterations. The RBF has 24 neurons, a learning rate of 0.125, and a max of 100 iterations. For the sine dataset, the NF:SAIL has a resolution of 0.25, and a set variance of 0.125. The MLP has 12 hidden neurons, a learning rate of 0.04, and a momentum of 0.02. The RBF has 12 neurons, and a learning rate of 0.25. For all datasets, the NF:SAIL has a learning rate of 1.0.

NF:SAIL proves its ability to effectively generalize on all datasets, surpassing the similar RBF network in all cases. Despite using the same output learning and calculation method, the NF:SAILs ability to effectively process large numbers of neurons, with guaranteed coverage of the input space, proves an advantage over the RBFs moving neurons. By surpassing all other algorithms on the housing and sine dataset regression problems, NF:SAIL proves its ability to effectively generalize with continuous inputs and outputs with remarkable effectiveness.

We emphasize that NF:SAIL is not intended to outperform the state of the art in classification and regression. Instead, NF:SAIL provides the unique ability to effectively retain learned information without revising previous data points. Regardless, NF:SAIL displays classification and regression abilities that closely match the state of the art.

On all datasets, NF:SAIL converges remarkably quickly. With this consistency, one can trust NF:SAIL to perform well once configured for a given problem. Most supervised algorithms are sensitive to hyperparameter values, requiring constant adjustment to maintain performance while data points in a real time dataset change. This concept is illustrated in Fig. 6, where various hyperparameters for MLP are tested on the iris dataset. Hidden neurons are always 4. Even small adjustments result in instability, lack of convergence, or even divergence. This sensitivity wastes human resources on hyperparameter adjustment. Fig. 7 presents the same test for NF:SAIL. Variance is always half of resolution. NF:SAIL proves robust to hyperparameter adjustment with remarkable

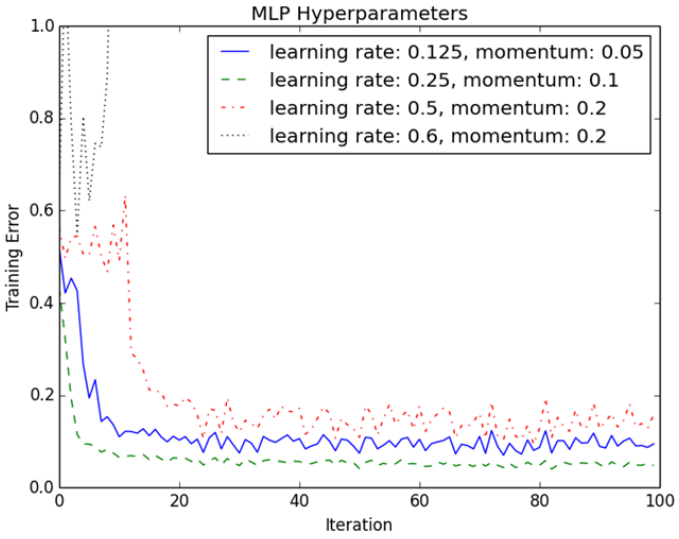

Fig. 6. MLP hyperparameter comparison

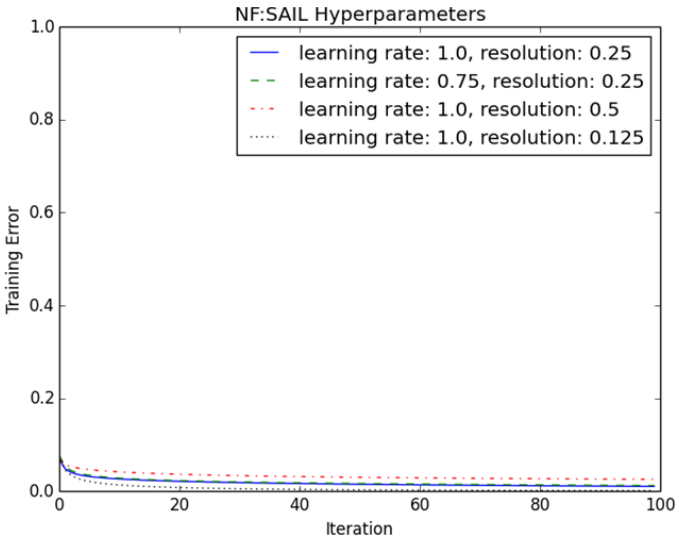

Fig. 7. NF:SAIL hyperparameter comparison

performance despite non-optimal adjustment, saving time and human resources, allowing for high autonomy and low cost.

IV. Conclusions

In this paper, we introduce a new supervised learning algorithm that uses a static ordered grid of radial basis function neurons featuring neuron contribution (1), which is at the heart of the algorithms abilities. Through the tests in section 3.1 and 3.2, we have shown that NF:SAIL has the distinct advantage of effectively retaining previously learned data points even after converging on new data points. We have also shown that the proposed NF:SAIL can effectively generalize on classification and regression tasks. Finally, we presented a mechanism for efficient lazy evaluation of static neurons in an infinite grid.

NF:SAILs ability to train on new data points without overriding previously learned information, and its ability to effectively generalize, make it an ideal algorithm for learning in continuous real time environments. When a massive number of data points can be seen, and each data point is unique, it is infeasible to store every data point for training. Instead, the learner must learn from each data point as it is presented, without overriding information from previous data points, even if a portion of the input space remains unvisited for many iterations.

Examining problems beyond real time learning, NF:SAILs exceptional ability to retain learned information while training allows one to efficiently update the network when new data points are obtained. Simply feeding new data points to NF:SAIL allows for efficient convergence without retraining on the set of all previous and new data points. An ability absent in state of the art supervised learning methods.

V. References

[1] Watkins, Christopher John Cornish Hellaby. "Learning from delayed rewards." PhD diss., University of Cambridge, 1989.

[2] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8, no. 3-4 (1992): 279-292.

[3] Sutton, Richard S., and Andrew G. Barto.Reinforcement learning: An introduction. Vol. 1, no. 1. Cambridge: MIT press, 1998.

[4] Lange, Sascha, and Martin Riedmiller. "Deep auto-encoder neural networks in reinforcement learning." In *IJCNN*, pp. 1-8. 2010.

[5] Riedmiller, Martin. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method." In *Machine Learning: ECML 2005*, pp. 317-328. Springer Berlin Heidelberg, 2005.

[6] Scott Fahlman, and Christian Lebiere. "The Cascade-Correlation Learning Architecture." In *Advances in Neural Information Processing Systems 2*. 1990.

[7] Rivest, François, and Doina Precup. "Combining TD-learning with cascade-correlation networks." In *ICML*, vol. 3, pp. 632-639. 2003.

[8] Broomhead, David S., and David Lowe. "Multi-variable functional interpolation and adaptive networks." *Complex Systems* 2 (1988): 321-355.

[9] Broomhead, David S., and David Lowe. *Radial basis functions, multi-variable functional interpolation and adaptive networks*. No. RSRE-MEMO-4148. ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM), 1988.

[10] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[11] Mangasarian, Olvi L., R. Setiono, and W. H. Wolberg. "Pattern recognition via linear programming: Theory and application to medical diagnosis." *Large-scale numerical optimization* (1990): 22-31.

[12] Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions, Statistics and Probability Letters, 33 (1997) 291-297.

[13] Huang, Guang-Bin, Paramasivan Saratchandran, and Narasimhan Sundararajan. "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation." *Neural Networks, IEEE Transactions on* 16, no. 1 (2005): 57-67.

[14] Huang, Guang-Bin, Lei Chen, and Chee-Kheong Siew. "Universal approximation using incremental constructive feedforward networks with random hidden nodes." *Neural Networks, IEEE Transactions on* 17, no. 4 (2006): 879-892.

[15] Dhar, V. K., A. K. Tickoo, R. Koul, and B. P. Dubey. "Comparative performance of some popular artificial neural network algorithms on benchmark and function approximation problems." *Pramana* 74, no. 2 (2010): 307-324.

TABLE III. DATASET COMPARISONS

	Iris Dataset			**Cancer Dataset**			**California Housing Dataset**			**Sine Dataset**		
	NF:SAIL	***MLP***	***RBF***	***NF:SAIL***	***MLP***	***RBF***	***NF:SAIL***	***MLP***	***RBF***	***NF:SAIL***	***MLP***	***RBF***
Mean Iterations	17.33	1979	5000	13.66	198.7	100	10	1000	100	1	484	20.66
Mean Training Error	0.019	0.038	0.089	0.019	0.018	0.227	0.037	0.068	0.165	0.0022	0.0346	0.0174
Mean Testing Error	0.079	0.063	0.155	0.105	0.076	0.228	0.054	0.068	0.165	0.0022	0.0329	0.0197
Iterations SD	4.03	2167	0	3.09	108.7	0	0	0	0	0	378.87	4.921
Training Error SD	0.000	0.023	0.012	0.000	0.004	0.006	0.002	0.001	0.005	0.0002	0.0301	0.0021
Testing Error SD	0.021	0.023	0.015	0.033	0.007	0.022	0.007	0.003	0.004	0.0001	0.0263	0.0040