Infinite Lattice Learner: An Ensemble For Incremental Learning

Justin Lovinger*, Iren Valova†

Computer and Information Science Dept,
University of Massachusets Dartmouth, MA

Abstract

The state-of-the-art in supervised learning has developed effective models for learning, generalizing, recognizing faces and images, time series prediction, and more. However, most of these powerful models cannot effectively learn incrementally. Infinite Lattice Learner (ILL) is an ensemble model that extends state-of-the-art machine learning methods into incremental learning models. With ILL, even batch models can learn incrementally with exceptional data retention ability. Instead of continually revisiting past instances to retain learned information, ILL allows existing methods to converge on new information without overriding previous knowledge. With ILL, models can efficiently chase a drifting function without continually revisiting a changing dataset. Models wrapped in ILL can operate in continuous real-time environments where millions of unique samples are seen every day. Big datasets too large to fit in memory, or even a single machine, can be learned in portions. ILL utilizes an infinite cartesian grid of points with an underlying model tiled upon it. Efficient algorithms for discovering nearby points and lazy evaluation make this seemingly impossible task possible. Extensive empirical evaluation reveals impressive retention ability for all ILL models. ILL similarly proves its generalization ability on a variety of datasets from classification and regression to image recognition.

Keywords Supervised Learning, Incremental Learning, Ensemble Learning, Neural Networks

1 Introduction

With a growing need for real-time learning and the rise of big data, incremental learning is becoming ever more important. Some models are developed with a focus on incremental learning, but as the no free lunch theorem states: no one method is optimal for every problem. A large archive of existing models exist to solve particular problems. Deep learning is effective on vision and audio problems, random forests excel at classification, and support vector machines overcome noisy problems. When one of these problems are found to need incremental learning, it is unfortunate to abandon existing effective models. Infinite Lattice Learner (ILL) enables these existing models to perform incremental learning by simply wrapping them, without modification, in an ILL ensemble.

While incremental learning (Gepperth and Hammer, 2016) has garnered multiple meanings, in this paper we explore incremental learning w.r.t. learning from small subsets of an unknown data distribution while retaining information learned from previous data. An incremental learning model must meet the following criteria (Polikar et al., 2001):

1. It must be capable of learning from presented data, i.e., it must be a learning model.
2. It must retain previously learned information after learning from newly presented data, without requiring access to previous data, i.e., it must retain data.

We provide an additional criteria, not intrinsic to the definition of incremental learning, but essential for application in many problem spaces:

*jlovinger@umassd.edu
†iren.valova@umassd.edu

3. Its memory usage must not scale infinitely with number of samples, i.e., $M \not\to \infty$ as $n \to \infty$, where M is memory usage, and n is the number of samples. In other words, memory must be finite and bounded regardless of how many or how often samples are presented.

Many state-of-the-art incremental learning algorithms rely on building an increasingly complex set of models as data is presented, and thus violate this criteria. ILL performs incremental learning with exceptional accuracy without violating criteria 3. The majority of learning models attempt incremental learning via *catastrophic forgetting*, which destroys the current model and retrains a new model on the entire set of presented data. Catastrophic forgetting does not meet the criteria we give for incremental learning, but ILL allows these models to learn incrementally without catastrophic forgetting.

Reinforcement learning (RL) is a particular application with great need for incremental learning. RL, inspired by the Pavlovian reflex and the ability of biological brains to learn from positive and negative reinforcement, is often formulated as a function mapping a state and action space to a reward value. This Q learning (Watkins, 1989; Watkins and Dayan, 1992; Sutton and Barto, 1998) problem commonly uses supervised learning algorithms to learn $Q : S \times A \to R$ (Sutton and Barto, 1998; Riedmiller, 2005; Lange and Riedmiller, 2010). RL problems in continuous problem spaces can generate millions of samples in real time. Attempting to learn such a massive dataset, in its entirety and in real time, is impractical. Instead, Q is learned incrementally, as samples are discovered. A popular potential solution is a sliding window, where only the n most recent samples are presented to the learning model, where n is the size of the window. The model can fully train on this window of n samples and effectively learn it, regardless of incremental learning ability. However, the model will retain only recent data, as dictated by the size of n (Riedmiller, 2005). Clearly, this is not ideal. This problem has been solved by storing all samples in a growing dataset (Riedmiller, 2005), but this method is infeasible for long learning sessions in continuous problem spaces.

ILL, inspired by Neural Field (Lovinger and Valova, 2016), uses an infinite cartesian grid of points. Similar to the popular k-nearest neighbor (KNN) and radial basis function network (RBF) algorithms, ILL classifies and performs regression based on similarity between an input vector and nearby points. Unlike KNN and RBF, each nearby point corresponds to a user defined learning model. Output and learning are delegated, by ILL, to these nearby models. In this manner, ILL retains the characteristics of its underlying model, while providing exceptional retention ability through an ensemble of such models tiled upon a static grid of points.

The next section provides background on incremental learning. Section 3 details the ILL algorithm. Section 4 presents Neural Field as a special case of ILL. Section 5 provides extensive benchmarking and comparative analysis to empirically prove the effectiveness of ILL. Finally, Section 6 concludes.

2 Incremental Learning

The goal of incremental learning (Gepperth and Hammer, 2016) is to learn a function in pieces, as new information or samples are observed. By contrast, batch learning assumes a full static dataset, allowing effective hyperparameter optimization, model selection, and training. Agents operating in real time environments, such as autonomous robots and cars, must learn from a constant stream of data. Many online applications use a constant stream of user feedback to train incremental learning systems. With big datasets too large to fit in memory, data must be fed to learning models in incremental portions.

The simplest approach to incremental learning is the use of a model effective for batch learning with a training algorithm suitable for streaming data. Regression models, such as linear regression (Neter et al., 1996; Seber and Lee, 2012; Montgomery et al., 2015), can be incrementally trained with stochastic gradient descent, or any numeric optimization method that does not rely on information from previous iterations. Support vector machines (SVM) (Cortes and Vapnik, 1995; Pradhan, 2013; Rebentrost et al., 2014) are similarly compatible with incremental numeric optimization. Multilayer perceptrons (MLP) (Tagliaferri et al., 2003; Hagan et al., 1996; Gatys et al., 2015) have found popularity in incrementally learning a reward function for reinforcement learning (Watkins, 1989; Watkins and Dayan, 1992; Sutton and Barto, 1998). Like regression and SVM, MLP can be trained incrementally with various gradient descent methods. Radial basis function networks (RBF) (Lowe and Broomhead, 1988; Broomhead and Lowe, 1988; Tan et al., 2001) and self organizing maps (SOM) (Kohonen, 1982; Kohonen, 1990; Kalteh et al., 2008) also have natural training mechanisms for incremental learning.

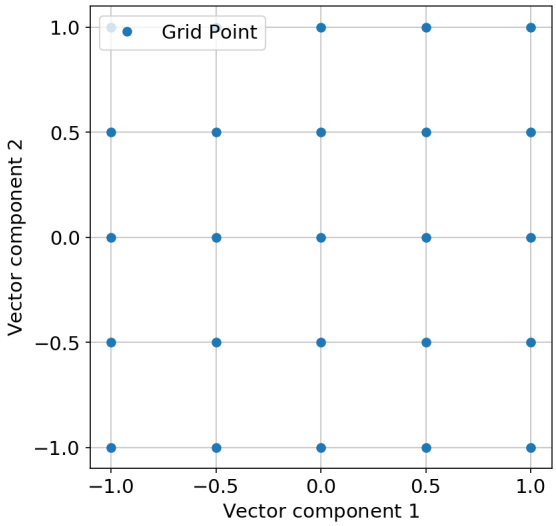

Figure 1: A cartesian grid with grid spacing $g = 0.5$

Explicit partitioning of the input space is a popular technique in modern incremental learning (Gepperth and Hammer, 2016; Polikar et al., 2001). Gaussian mixture distributions have found use in incremental learning by leveraging the local learning characteristics of a Gaussian distribution (Cederborg et al., 2010). Decision trees are also known to naturally partition an input space, and incremental variants have been explored (Utgoff, 1989; Jiang et al., 2013; Rutkowski et al., 2014). The cartesian grid of ILL performs an overlapping partition of the input space, borrowing the advantages of partitioning methods.

Another technique to enable incremental learning is an ensemble of models with individually poor incremental learning ability. Learn++ (Polikar et al., 2001) uses an ensemble of weak neural networks to learn incrementally. Random forest, a state-of-the-art ensemble for non-incremental learning, has been extended to incremental learning (Ma and Ben-Arie, 2014; Ristin et al., 2016). Unlike ILL, many such ensembles add new model instances to the ensemble as new samples are presented. This presents a challenge for big data problems with quickly streaming data, as the size of the ensemble rapidly grows beyond memory limits. ILL aims to remedy this issue by partitioning the ensemble into parts of the input space, providing an upper bound on size regardless of how often new samples are presented.

3 The Infinite Lattice Learner Algorithm

ILL enables incremental learning by generating an ensemble of learning models in an ordered cartesian grid, depicted in Figure 1. Each grid point $\vec{p}$ in this cartesian grid has a corresponding model $m_{\vec{p}}$. The output $\vec{o}_{m\vec{p}}$ of each model $m_{\vec{p}}$ is scaled by distance between the input $\vec{x}$ and its corresponding grid point $\vec{p}$ in the input space. In this way, information is stored in the local input space around model points, as opposed to globally in an individual model. With this local storage mechanism, 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. In short, ILL divides the problem space into an ordered grid and trains one model for each cell in this grid. A weighted sum of these models allows generalization between input vectors corresponding to separate cells.

3.1 Neighborhood

With an infinite grid of points, efficient algorithms are essential to obtain the points most essential to a given activation of ILL. For an input vector $\vec{x}$, the points near $\vec{x}$ are intuitively essential. Formally, the distance between $\vec{x}$ and any obtained point $\vec{p}$, $\|\vec{x} - \vec{p}\|$, should be a small value.

Each point $\vec{p}$ can correspond to a learning model $m_{\vec{p}}$. These models can individually learn and generalize. By, for each input $\vec{x}$, obtaining points in the neighborhood N around $\vec{x}$, we efficiently partition a problem by

its input space. Training a separate model for each $\vec{p} \in N$, for each N corresponding to an input $\vec{x}$, allows us to learn incrementally, one subset of the input space at a time. By overlapping sets, we retain the ability to effectively generalize.

With ∞ points and ∞ models corresponding to points, pre-initializing models is not feasible. Instead, lazy evaluation allows ILL to initialize models as they are needed. When a nearby point $\vec{p}$ is discovered by a neighborhood function, ILL attempts to retrieve the corresponding model $m_{\vec{p}}$ from a hash table. If $m_{\vec{p}}$ does not exist, it is initialized and stored in the hash table.

Standard methods for obtaining points in a neighborhood around a central point are infeasible on an infinite grid. However, with the cartesian ILL grid, the position of every point is known a-priori given only grid spacing g. Efficient analogs for popular neighborhood functions can be developed to work on an infinite cartesian grid. The particular neighborhood function used with ILL is an interchangeable hyperparameter. Some useful neighborhood functions are provided here.

3.1.1 Radius

A natural approach to finding points near a center $\vec{c}$ is to return all points within a given radius of $\vec{c}$. The grid points algorithm presented in Neural Field (Lovinger and Valova, 2016) can efficiently find all points around $\vec{c}$ within radius r on a cartesian grid with g spacing. This algorithm is re-stated in Algorithm 1 for convenience. See Appendix A for a note on implementing Algorithm 1 with floating point math. An example

Algorithm 1 Radius Points

function RADIUS_POINTS(center $\vec{c}$, radius r)
  $\vec{c}_1 \leftarrow$ the first component of $\vec{c}$
  if $\vec{c}$ contains only 1 component **then**
    $s \leftarrow g\lceil(\vec{c}_1 - r)/g\rceil$ ▷ smallest number on grid $\geq (\vec{c}_1 - r)$
    $e \leftarrow g\lfloor(\vec{c}_1 + r)/g\rfloor$ ▷ greatest number on grid $\leq (\vec{c}_1 + r)$
    return all numbers from s to e spaced g apart.
  end if
  $\vec{c}_{2\ldots} \leftarrow$ a vector slice containing all components after and including the second component of $\vec{c}$
  $P \leftarrow$ an empty list
  $x \leftarrow g\lceil(\vec{c}_1 - r)/g\rceil$ ▷ smallest number on grid $\geq (\vec{c}_1 - r)$
  while $x \leq \vec{c}_1 + r$ **do**
    for partial point p in RADIUS_POINTS($\vec{c}_{2\ldots}$, $\sqrt{r^2 - (x - \vec{c}_1)^2}$) **do**
      $j \leftarrow x$ concatenated with p
      Add j to list P
    end for
    $x \leftarrow x + g$
  end while
  return P
end function

of Algorithm 1 is given in Figure 2.

ILL can use Algorithm 1 to find neighborhood points near an input vector $\vec{x}$. When used with ILL, $\vec{c} = \vec{x}$. Choice of radius r can drastically affect performance. If r is too small, Algorithm 1 may return 0 points. If r is too large, ILL can waste computation on an unnecessarily large number of points. A simple heuristic can effectively select r based on $\vec{x}$. When $\vec{x}$ is near a grid point, r can be very small; when $\vec{x}$ is far from a grid point, r is increased:

$$r = d(\vec{x}, \vec{p}_c) + s \log(1 + d(\vec{x}, \vec{p}_c)) \tag{1}$$

where d is distance; $s \geq 0$ is a scaling factor; and $\vec{p}_c$ is the point closest to $\vec{x}$, which is easily calculated by rounding each component of $\vec{x}$ to the nearest multiple of g. Selecting $s = 1/N_d$, where N_d is the dimensionality of the neighborhood, is effective in practice.

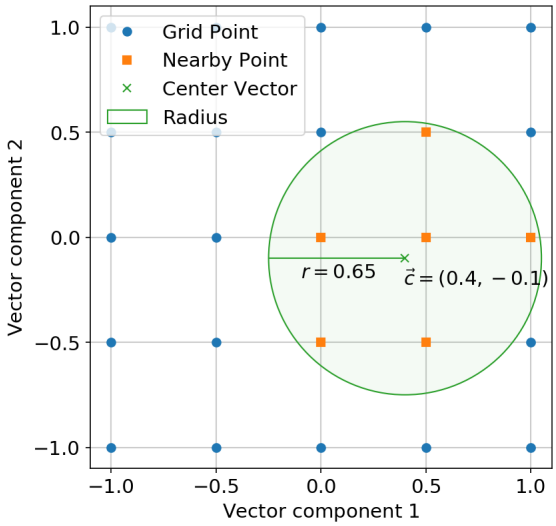

Figure 2: Algorithm 1 discovering all points within radius $r = 0.65$ of vector $\vec{c}$.

3.1.2 K-Nearest Neighbors

By leveraging Algorithm 1, a k-nearest neighbors (KNN) neighborhood function, interchangeable with Algorithm 1, is easily developed. Starting with a small radius $r = \epsilon$, r can incrementally increase by a user selected rate factor $\alpha > 1$ until k points are within a hyper-sphere of radius r centered at $\vec{c}$. Because the performance of Algorithm 1 scales with the number of points returned, this KNN variant has acceptable performance as long as k is not excessively large.

Pseudo-code is given in Algorithm 2. Note that Algorithm 2 can return more than k points if radius

Algorithm 2 KNN Points

$r \leftarrow \epsilon$
$P \leftarrow$ RADIUS_POINTS$(\vec{c}, r)$
while $|P| < k$ **do**
  $r \leftarrow \alpha r$
  $P \leftarrow$ RADIUS_POINTS$(\vec{c}, r)$
end while
return P

scaling factor α is not sufficiently small. Points P can be truncated by trimming all but the k nearest points, or left as is for a $|P| \approx k$ algorithm. Decreasing α so that Algorithm 2 never returns more than k points results in unsatisfactory performance and is a poor solution. Empirical tests show that a rate factor $\alpha \approx 1.2$ is effective when P is truncated, and $\alpha \approx 1.02$ is effective when P is *not* truncated. An example of Algorithm 2 is given in Figure 3.

3.2 Similarity

With an infinite grid of points, ILL requires a neighborhood function to select points near an input vector $\vec{x}$, as seen in Section 3.1. While a neighborhood function allows local storage of information, it is a coarse mechanism that doesn't differentiate between how close points are, as long as they are close enough to be in the local neighborhood. This mechanism can be refined with the use of a similarity function. Given two points $\vec{p}_1$ and $\vec{p}_2$, if $\vec{p}_1$ is closer to $\vec{x}$ than $\vec{p}_2$, formally $\|\vec{p}_1 - \vec{x}\| < \|\vec{p}_2 - \vec{x}\|$, corresponding model $m_{\vec{p}_1}$ should contribute more to the ensemble output than $m_{\vec{p}_2}$. The precise degree of contribution is determined by a similarity function.

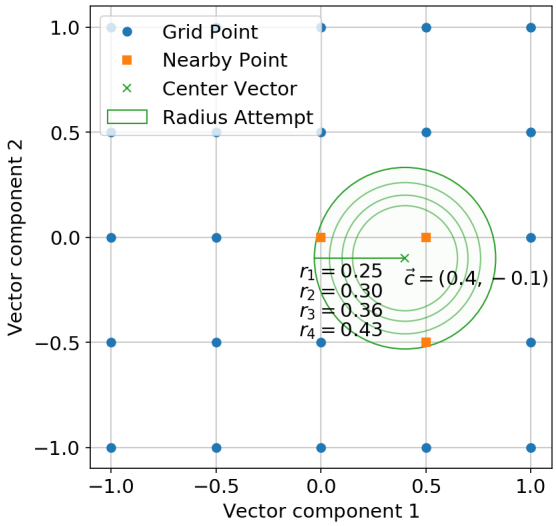

Figure 3: Algorithm 2 discovering the $k = 3$ nearest points. With an initial radius $r_1 = 0.25$, r increases by a factor of $\alpha = 1.2$ until the circle with $r_4 = 0.43$, centered at $\vec{c}$, contains 3 points.

A similarity function is a radial basis function $\phi(\cdot, \cdot)$ that satisfies the condition

$$\phi(\vec{a}_1, \vec{b}) \geq \phi(\vec{a}_2, \vec{b}) \text{ for all } \vec{a}_1, \vec{a}_2, \vec{b} \in \mathbb{R}^n \text{ given } \|\vec{a}_1 - \vec{b}\| < \|\vec{a}_2 - \vec{b}\| \tag{2}$$

A similarity function does not decrease in value when distance between its arguments $\vec{a}$ and $\vec{b}$ decreases. Useful, but not necessary, conditions for a similarity function are: it is continuous; it monotonically increases in value as distance decreases. A similarity function is maximized when $\|\vec{a} - \vec{b}\| = 0$ and minimized when $\|\vec{a} - \vec{b}\| = \infty$. Note that, as a radial basis function, a similarity function also has the condition that its value depends only on the distance between its arguments.

A popular similarity function is the Gaussian

$$\phi(\vec{a}, \vec{b}) = e^{-(\frac{\|\vec{a}-\vec{b}\|^2}{v})} \tag{3}$$

where v is a variance parameter. With this Gaussian similarity function, $\phi(\vec{a}, \vec{b})$ approaches 1 as the distance between $\vec{a}$ and $\vec{b}$ approaches 0. Conversely, $\phi(\vec{a}, \vec{b})$ rapidly approaches 0 as distance approaches ∞.

The Gaussian similarity function is found to perform better with ILL when there is a significant difference between the similarity of points $\vec{p}$ in the neighborhood N around $\vec{x}$. A formula to scale v by distance between furthest $\vec{p} \in N$ and $\vec{x}$ ensures highly variable similarity,

$$v = \frac{\max_{\vec{p} \in N} \|\vec{p} - \vec{x}\|}{\ln(1/s_f)} \tag{4}$$

where s_f is desired similarity of the furthest model point.

Normalizing similarity of all points $\vec{p} \in N$ to a sum of 1 improves the consistency of a similarity function. A Normalized similarity function, called on a point $\vec{p}_0$ and an input vector $\vec{x}$

$$\phi_n(\vec{p}_0, \vec{x}) = \frac{\phi(\vec{p}_0, \vec{x})}{\sum_{\vec{p} \in N} \phi(\vec{p}, \vec{x})} \tag{5}$$

divides each similarity by the sum of all relevant similarities. As is seen in Section 3.3, normalizing similarity can improve generalization of the ILL ensemble by providing a smooth transition between learned samples.

3.3 Output and Learning

After all points $\vec{p}$ in the local neighborhood N around a given input $\vec{x}$ are determined (see Section 3.1 for details), the ILL output $\vec{\varphi}$ is a sum of outputs $\vec{o}_{m\vec{p}} = m_{\vec{p}}(\vec{x})$ from each model $m_{\vec{p}}$ corresponding to points

$\vec{p} \in N$. To better differentiate between points $\vec{p} \in N$, this sum is weighted by similarity $\phi(\vec{p}, \vec{x})$

$$\vec{\varphi} = \sum_{\vec{p} \in N} \phi(\vec{p}, \vec{x}) \vec{o}_{m\vec{p}} \tag{6}$$

Note that normalizing the similarity function (5) can improve generalization.

Every iteration, each model $m_{\vec{p}}$ corresponding to a point $\vec{p} \in N$ is activated to obtain $\vec{o}_{m\vec{p}} = m_{\vec{p}}(\vec{x})$, and all model outputs $\vec{o}_{m\vec{p}} \forall \vec{p} \in N$ are combined to obtain the ILL output $\vec{\varphi}$. Adjusting $\vec{\varphi}$ towards a target vector $\vec{t}$ therefore requires adjusting $\vec{o}_{m\vec{p}} \forall \vec{p} \in N$. This can be accomplished by generating a target variable $\vec{t}_{m\vec{p}}$ for each $\vec{p} \in N$ and training each $m_{\vec{p}}$ to output $\vec{t}_{m\vec{p}}$. Note that the training of an underlying model $m_{\vec{p}}$ depends entirely on the implementation of $m_{\vec{p}}$ and is independent of ILL. The optimal set of model targets give $\vec{\varphi} = \vec{t}$:

$$\vec{t} = \sum_{\vec{p} \in N} \phi(\vec{p}, \vec{x}) \vec{t}_{m\vec{p}} \tag{7}$$

which follows directly from (6).

However, there are infinite solutions to (7). Because ILL aims to enable effective incremental learning, and large changes to model parameters are detrimental to incremental learning, we want to find the solution to (7) that minimizes the difference between $\vec{o}_{m\vec{p}}$ and the desired model output $\vec{t}_{m\vec{p}}$ for all $\vec{p} \in N$. Furthermore, because ILL stores information locally, the closer $\vec{p}$ is to $\vec{x}$, the more we can change the corresponding model $m_{\vec{p}}$ without destroying learned information. To this end, the optimal set of model targets $\vec{t}_{m\vec{p}} \forall \vec{p} \in N$ is given by

$$\underset{\vec{t}_{m\vec{p}} \forall \vec{p} \in N}{\arg\min} \sum_{\vec{p} \in N} \frac{\|\vec{t}_{m\vec{p}} - \vec{o}_{m\vec{p}}\|}{\phi(\vec{p}, \vec{x})} \tag{8}$$

under the constraint of (7). Note that the norm $\| \cdot \|$ is used here as a distance metric, and similar metrics, like mean squared error, are also appropriate.

Fully solving (8) every training iteration is computationally expensive and unnecessary. Every ILL iteration, one step can be taken towards a solution to (8), as given by a single optimization iteration. Over multiple ILL iterations, these sequences of steps act as a full optimization procedure, converging to an optimal set of targets. When an underlying model does not converge to its target in a single iteration, these one-step targets prove more than sufficient, and the burden of fully solving (8) is avoided.

3.4 High Dimensional Data

The number of points in a cartesian grid scales exponentially with dimensionality d. The optimistic outcome for ILL is that an input vector $\vec{x}$ lies near a small number of points, and the neighborhood function (see Section 3.1) selects only these nearby points. However, if $\vec{x}$ lies in the center of a hypercube of grid points, at least 2^d points are equidistant to $\vec{x}$, as depicted in Figure 4. On high dimensional problems, discovering 2^d points with a neighborhood function becomes intractable. Note that even the k-nearest neighbors function presented in Section 3.1 must discover all points in some radius before returning the k nearest.

In lieu of a neighborhood function capable of efficiently discovering an exponential number of points and mechanisms to learn an exponential number of models, dimensionality of the neighborhood must be decreased to a reasonable number: d_N. Note that dimensionality need only decrease with regard to the neighborhood and discovery of nearby points on a cartesian grid. Underlying models can learn the original d dimensional problem space or implement their own mechanisms for handling high dimensional data. This separation of dimensionality allows the underlying model to maximize their learning and generalization while avoiding the 2^d nearby points problem.

A d by d_N reduction matrix M_N can perform the dimensionality reduction: $\vec{x} \times M_N = \vec{x}_N$, giving us the reduced input vector $\vec{x}_N$. Any number of existing methods, such as principle component analysis (PCA) (Pearson, 1901; Bengio et al., 2013; Witten et al., 2016) or linear discriminant analysis (LDA) (Fisher, 1936; Shi et al., 2014), can find an effective reduction matrix M_N. However, these methods require a known dataset. While incremental variants exist (Weng et al., 2003; Zhao et al., 2006; Balsubramani et al., 2013; Pang et al., 2005; Zhao and Yuen, 2008), changing M_N can destroy previously learned data. The retention ability of ILL relies on a static cartesian grid, and changing M_N effectively changes this grid by repositioning

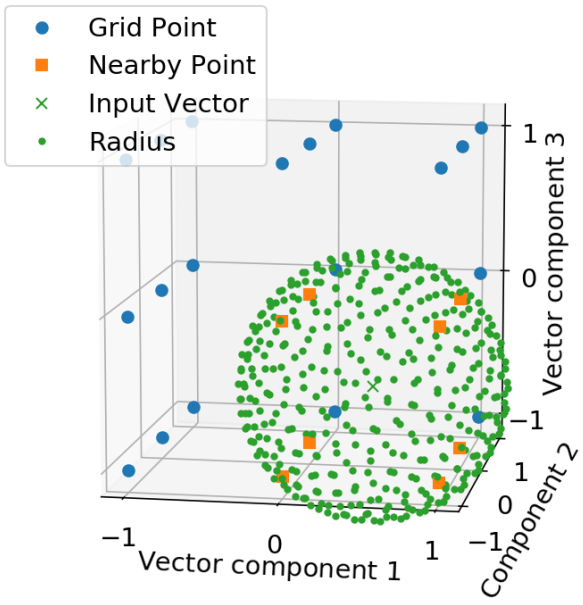

Figure 4: Input vector $\vec{x} = [0.5, -0.5, -0.5]$ in the center of a hypercube of points, on a cartesian grid with spacing $g = 1$. The minimum radius $r = 1.225$ to contain at least 1 point contains 2^d points, where dimensionality $d = 3$.

all input vectors in the reduced neighborhood space. To retain retention ability, M_N should remain unchanged once set. This presents us with two options:

1. Use a random projection (Johnson and Lindenstrauss, 1984; Bingham and Mannila, 2001; Cohen et al., 2015), without relying on a known dataset.

2. Delay learning until dimensionality reduction can be performed on the first n samples discovered during incremental learning, where n can be chosen by the user.

Empirical tests show high performance with random projections. We do not see the performance drop-off typical of a naive random projection because the random projection only reduces the ILL neighborhood space, while underlying models use original higher dimensional input vectors.

4 Neural Field: A Special Case of ILL

The cartesian grid neighborhood and use of a similarity metric are expanded concepts from neural field (NF) (Lovinger and Valova, 2016). Although these techniques are generalized and enhanced in ILL, using the radius neighborhood function given in Algorithm 1 and normalized (5) Gaussian similarity (3) allow ILL to equal NF.

The only missing piece is a choice of underlying model that matches the behavior of NF. In NF, each grid point $\vec{p}$ stores an output vector and the NF output is a weighed summation of these vectors, weighted by similarity between $\vec{p}$ and an input vector $\vec{x}$. ILL performs an equivalent weighted summation (6) of output vectors from each model $m_{\vec{p}}$ corresponding to points $\vec{p}$ in the neighborhood N. To create an ILL model equivalent to NF, we only need an underlying model that stores and returns a vector, which we call a neuron.

Matching the learning rule of NF is similarly straightforward. NF updates the vector of each neuron by taking a step toward a given target vector and scaling step size by normalized Gaussian similarity between the neuron and an input vector. A generalization of the NF learning rule can be applied with ILL:

$$\vec{t}_{m\vec{p}} = \vec{o}_{m\vec{p}} + \phi(\vec{p}, \vec{x})(\vec{t} - \vec{\varphi}) \tag{9}$$

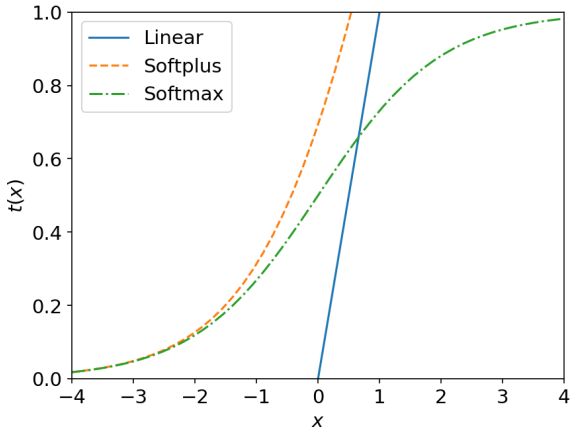

Figure 5: Output of linear, softplus, and softmax transfer functions. Softmax is calculated on a $[x, 0]$ vector, and the first coordinate of softmax(x) is depicted on the y axis.

Where $\vec{t}_{m\vec{p}}$ is a learning target for model $m_{\vec{p}}$ corresponding to point $\vec{p}$, $\vec{o}_{m\vec{p}} = m_{\vec{p}}(\vec{x})$ is the output of $m_{\vec{p}}$, $\phi(\cdot, \cdot)$ is the normalized (5) Gaussian similarity function (3), and $\vec{\varphi}$ is the output of the ILL ensemble.

This presentation shows that NF can be formulated as an ILL ensemble with a particular, and very weak, underlying model. With such a weak underlying model, NF heavily relies on the ILL similarity metric and requires very low grid spacing to learn complex functions. When the ILL grid is a poor prior for a given problem, NF will suffer low performance. By contrast, ILL with a stronger underlying model can increase grid spacing and rely on the prior of its underlying model to achieve high performance.

5 Benchmark and Comparative Analysis

A robust comparison of ILL models with 6 state-of-the-art incremental and batch supervised learning models, proves the effectiveness of ILL. For each comparison model, an ILL ensemble of the model is included. Additionally, neural field, the predecessor of ILL, is included for comparison. Automatic hyperparameter selection optimizes each model, eliminating researcher bias and providing insight into effective hyperparameters. A direct comparison of incremental retention ability is performed using a retention error metric. Finally, a robust comparison of generalization accuracy is performed, with each model learning incrementally. Classification, regression, and high dimensional image datasets are included in this comparison.

5.1 Models

The popular multilayer perceptron (MLP) (Tagliaferri et al., 2003; Hagan et al., 1996; Gatys et al., 2015) has found use in incremental learning problems and is especially popular for incrementally learning a reward function in reinforcement learning (Watkins, 1989; Watkins and Dayan, 1992; Sutton and Barto, 1998; Hausknecht and Stone, 2015; Van Hasselt et al., 2016). An MLP can be described as a function of the form: $f(\vec{x}) = t_n...(t_2(t_1(\vec{x}\boldsymbol{W}_1)\boldsymbol{W}_2)...\boldsymbol{W}_n)$, where t_i is the i^{th} transfer function, $\boldsymbol{W}_i$ is the i^{th} weight matrix, and $\vec{x}$ is an input vector. MLP is trained by finding a set of weight matrices that minimize an objective function, such as mean squared error. Our MLP models use a softplus (Glorot et al., 2011; Tóth, 2013; Maas et al., 2013) transfer function for hidden layers, softmax transfer for output layers of classification problems, and linear transfer for output layers of regression problems. Our chosen transfer functions are depicted in Figure 5. Incremental training is performed with steepest descent minimizing mean squared error and Wolfe line search (Nocedal and Wright, 2006) to select step size. Initial step size for line search is $1.5\alpha_{n-1}$, where α_{n-1} is the step size from the previous iteration. Steepest descent and its gradient descent variants are common in state-of-the-art incremental learning (Hazan et al., 2016; Ruder, 2016; Yoshida et al., 2017; Variddhisaï and Mandic, 2017; Rosasco and Villa, 2015).

Radial basis function networks (RBF) (Lowe and Broomhead, 1988; Broomhead and Lowe, 1988; Tan et al., 2001) use a combination of clustering, a similarity metric, and linear regression to perform classification

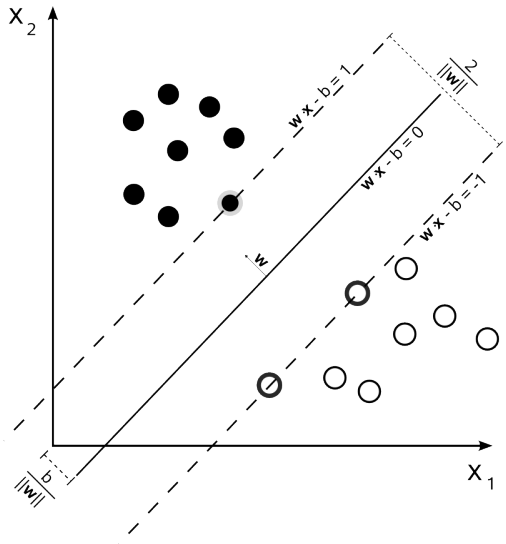

Figure 6: An SVM maximizing the margin between its decision boundary and samples used as support vectors (Cyc, 2008).

and regression. During training the input space is clustered, providing a set of cluster centers $\boldsymbol{C}$. When an RBF network is activated, the similarity between each center $\vec{c} \in \boldsymbol{C}$ and an input vector $\vec{x}$ is calculated, giving a similarity vector $\vec{s}$. The RBF output is given by $f(\vec{s}) = \vec{s}\boldsymbol{W}$, where $\boldsymbol{W}$ is a weight matrix. Our RBF determines $\boldsymbol{C}$ incrementally with a Kohonen self organizing map (SOM) (Kohonen, 1982; Kohonen, 1990; Kalteh et al., 2008) with a linear neighborhood of size 2, a movement rate $\alpha_0 = 0.1$ for the nearest neuron, $\alpha_1 = 0.0368$ for the first neighbor, and $\alpha_2 = 0.0018$ for the second neighbor. Normalized Gaussian similarity (3) is our similarity function. The weight matrix $\boldsymbol{W}$ is trained with the same state-of-the-art incremental steepest descent algorithm as our MLP.

Linear regression (Neter et al., 1996; Seber and Lee, 2012; Montgomery et al., 2015) is a time tested method for supervised and incremental learning. With linear regression, a function of the form $f(\vec{x}) = \vec{x} \cdot \vec{w}$ predicts a single output value, where $\vec{w}$ is a weight vector. Linear regression can be generalized to predict an output vector: $f(\vec{x}) = \vec{x}\boldsymbol{W}$, where $\boldsymbol{W}$ is a weight matrix. A regression model is trained by finding a weight matrix that minimizes an objective function, such as mean squared error. The performance of regression can be improved by adding a penalty term to minimize the magnitude of weights. Using the ℓ_1 norm of weights, $\|\boldsymbol{W}\|_1$, as a penalty factor encourages prediction using a minimum of attributes, through a sparse weight matrix. This method, known as lasso regression (Tibshirani, 1996; Tibshirani, 2011), is found to improve generalization (Wright et al., 2009). Our lasso implementation is provided by the scikit-learn Python library (Pedregosa et al., 2011) and is trained incrementally with stochastic gradient descent. Stochastic gradient descent is common in state-of-the-art incremental learning (Hazan et al., 2016; Ruder, 2016; Variddhisaï and Mandic, 2017; Ramos and Ott, 2016; Song et al., 2016; Soudry et al., 2015; Rosasco and Villa, 2015).

The support vector machine (SVM) (Cortes and Vapnik, 1995; Pradhan, 2013; Rebentrost et al., 2014) method is effectively an evolution of regression and has found similar popularity for incremental learning. Using a function similar to a regression model, SVM is trained to minimize an objective that is robust to outliers and creates an effective decision boundary, as depicted in Figure 6. This use of a unique objective function, such as epsilon insensitive loss (Suykens and Vandewalle, 1999; Huang et al., 2012), differentiates SVM from regression models. We include a linear SVM in our comparison. Our SVM implementation is provided by the scikit-learn Python library (Pedregosa et al., 2011) and is trained incrementally with the same stochastic gradient descent as our linear regression.

Random forest (RF) (Ho, 1995; Díaz-Uriarte and De Andres, 2006; Rodriguez-Galiano et al., 2012) is a modern extension of the classic decision tree (DT) algorithm (Utgoff, 1989; Schmid, 2013; Pradhan, 2013). Each DT in the RF trains on a subset of attributes and samples. In classic ensemble style, the output of

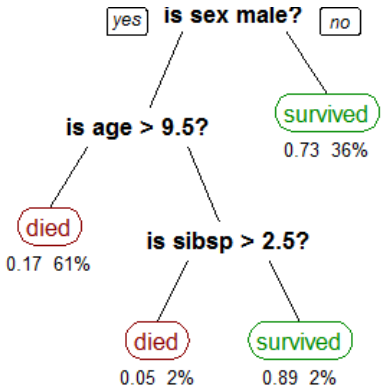

Figure 7: A decision tree predicting the survival of a passenger on the Titanic using the sex, age, and siblings or spouses onboard attributes (Milborrow, 2011). Leaves contain samples from both classes. Under each leaf node is probability of survival given as a decimal value and percent of samples filtered into the leaf.

individual DTs is combined with a competitive vote for classification or average for regression to form the RF output. Individual DTs learn a sequence of if-then rules, applied on each attribute, as depicted in Figure 7. Information theory (Shannon, 2001) remains popular for determining the attribute that best divides samples into accurate classifications, leading to the ID3 algorithm (Quinlan, 1986). RF is a staple of state-of-the-art supervised learning for its impressive performance and ability to avoid over-fitting as complexity increases. Our RF implementation is provided by the scikit-learn Python library (Pedregosa et al., 2011) using default parameters unless otherwise specified.

The k-nearest neighbors (KNN) (Altman, 1992; Weinberger et al., 2006; Muja and Lowe, 2014) method stores training samples and makes predictions with the k samples nearest to a given input vector. KNN models differ by distance metric used to specify *nearest* and how the k neighbors are combined to form a model output. Euclidean distance is a common distance metric, competitive vote is a common method to combine the k-nearest for classification problems, and taking the mean is a common method to combine the k-nearest for regression problems. The KNN algorithm has proven effective despite, or perhaps because of, its simplicity. Our KNN implementation is provided by the scikit-learn Python library (Pedregosa et al., 2011) using default parameters unless otherwise specified.

For each of the above models, a variant using ILL is included. ILL models are designated: ILL(m), where m is the underlying model. Random projections are used for dimensionality reduction.

Incremental variants of RF and KNN rely on an unbounded increase in model size, by either adding new trees to the RF ensemble or storing all samples for KNN. This size complexity is unsuitable for many incremental learning tasks. Instead, RF and KNN are presented with catastrophic forgetting. When new samples are presented, these models discard previously learned information and train on the new samples. These batch learning models provide a baseline comparison to our incremental learning models and illustrate ILL with batch models.

The neural field (NF) (Lovinger and Valova, 2016) algorithm, as detailed in Section 4, is included in our comparison for its similarity to ILL. NF is the precursor and inspiration for ILL and our analysis shows that ILL greatly improves on its predecessor.

Note that a number of incremental learning methods, such as Learn++ (Polikar et al., 2001), are not included in our comparison because they violate our infinite memory criteria (criteria 3 given in Section 1) for incremental learning.

5.1.1 Selecting Hyperparameters

When a researcher manually adjusts hyperparameters, it is possible to bias results in their favor. To eliminate this issue of researcher bias, we automatically discover hyperparameters for all models using a brute force optimizer, also known as grid search. Grid search ensures that all hyperparameter combinations are tested for every model. The grid search optimizer aims to maximize accuracy (or minimize error on regression

problems) on a validation set. The validation set is a subset of the training set with $^2/_3$ samples for training the validation model and $^1/_3$ for validation.

The optimization objective function f for classification problems is $f(m, D) = \text{acc}(m_{tr}, D_{te}) + 0.05 \times \text{acc}(m_{tr}, D_{tr})$, where m_{tr} is model m after training on a training set, acc is the accuracy of a model on a dataset, D_{te} is the testing set of dataset D, and D_{tr} is the training set of dataset D. Some value is added for performance on the training set to discourage over-fitting on the validation set. The objective function for regression problems is $f(m, D) = (1 - \text{mse}(m_{tr}, D_{te})) + 0.05 \times (1 - \text{mse}(m_{tr}, D_{tr}))$, where mse is mean squared error.

MLP models can select a number of hidden neurons n_h in the range $[1, 64]$. RBF models can select a number of neurons n in the range $[1, 128]$. Variance v is selected with a $v = {}^4/_n$ heuristic, or manually set for some problems where low variance leads to computational errors. Lasso and SVM do not have any hyperparameters to automatically optimize. RF models can select a number of ensemble decision trees t in the range $[1, 32]$. KNN models can select a number of nearest neighbors k in the range $[1, 32]$.

ILL models can select: grid spacing $g = 0.001, 0.042, 0.099, 0.180, 0.294, 0.455, 0.681,$ or 1.000; one of 4 neighborhood functions f_N: KNN with $k = 1, 3,$ or 5 written as k-NN or adaptive radius with radius scaling of $^1/_{d_N}$, where d_N is the dimensionality of the neighborhood; dimensionality reduction d_N to 1 or 2 dimensions on problems with more than 4 dimensions. Each ILL model uses an underlying model with its optimized hyperparameters. While hyperparameters of the underlying model can be optimized independently for use with ILL, keeping the same hyperparameters allows easier comparison between ILL and its underlying model.

5.2 Retention of Previously Learned Data

The ability to retain previously learned data enables incremental learning. As a model is presented with new samples, iteration after iteration, its parameters are adjusted to minimize error on each sample as it is seen. Models with high retention ability maintain low error on samples that have not been presented in many iterations.

To test retention ability, we define a retention error metric (Lovinger and Valova, 2016). Retention error e_r for model m on dataset D is given by Algorithm 3. By testing error on a given subset of D, after learning

Algorithm 3 Retention Error

Split D into two disjoint sets D_1 and D_2
$m_1 \leftarrow m$ after training on D_1 until convergence
$m_2 \leftarrow m_1$ after training on D_2 until convergence
$e_r \leftarrow \text{mse}(m_2, D_1)$ ▷ mse is mean squared error

it, and then training on a separate subset, e_r effectively measures a models ability to retain learned data after repeated presentation of different data. Note that this metric depends on both the models ability to learn the first subset D_1 and its ability to retain that information after training on the second subset D_2. Subtracting the error obtained immediately after learning D_1, as given by $\text{mse}(m_2, D_1) - \text{mse}(m_1, D_1)$, disambiguates the ability to retain information from the ability to learn. However, this results in a poor metric that rewards an inability to learn. A hypothetical model m that never changes has high $\text{mse}(m_1, D_1)$, but $\text{mse}(m_1, D_1) \approx \text{mse}(m_2, D_1)$, resulting in $e_r \approx 0$ if $e_r = \text{mse}(m_2, D_1) - \text{mse}(m_1, D_1)$. Algorithm 3, which depends on learning ability, is more useful in practice.

To isolate a models retention ability from its generalization ability, datasets of random samples are generated and used for testing. Each random dataset contains 10 samples, 1 real valued attribute, and 1 real valued target. Retention error is averaged over 100 such random datasets. Every model is tested on the same set of random datasets.

For hyperparameter optimization, retention tests use a validation set containing half as many datasets, each with the same dimensionality and number of samples. The objective function for retention problems is $f(m, D) = (1 - \text{ret}(m, D))$, where ret is retention error, as given by Algorithm 3.

Table 1: Model Retention Comparison

Model	Mean e_r	Hyperparameters
ILL(MLP)	0.0036	g: 0.001, f_N: 5-NN, n_h: 46
ILL(RBF)	0.0069	g: 0.001, f_N: 5-NN, n: 118, v: 0.034
ILL(Lasso)	0.2004	g: 0.001, f_N: 5-NN
ILL(SVM)	0.0684	g: 0.001, f_N: 5-NN
ILL(RF)	0.0079	g: 0.001, f_N: 5-NN, t: 28
ILL(KNN)	0.0078	g: 0.001, f_N: 5-NN, k: 5
NF	0.0001	g: 0.001, f_N: 1-NN
MLP	0.4428	n_h: 46
RBF	0.3528	n: 118, v: 0.034
Lasso	0.3203	
SVM	0.3473	
RF	0.5123	t: 28
KNN	0.4086	k: 5

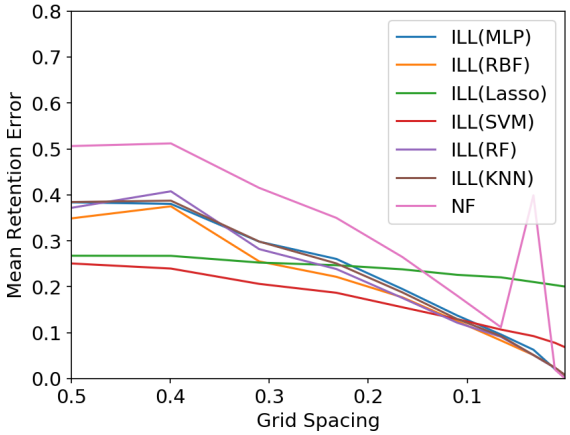

Figure 8: Relationship between grid spacing g and retention error e_r

5.2.1 Retention Comparison

Table 1 presents a comparison of the retention ability of models presented in Section 5.1. Retention error e_r is tested on the random datasets described in Section 5.2 and mean e_r is given. Hyperparameters discovered with the hyperparameter search detailed in Section 5.1.1 are also provided.

All models incorporating ILL show an exceptional ability to retain learned data, with most retention error $e_r < 0.01$, while models without ILL show $e_r > 0.3$. Excluding the ILL(Lasso) outlier, the *best* non-ILL model has more than 4 times higher e_r than the *worst* ILL model. On average, ILL improves the retention ability of its underlying model by 50 times. ILL undoubtedly proves its ability to retain learned data. On the topic of hyperparameters, we see a strong preference for low grid spacing g and k-NN neighborhood functions, with 5-NN most common.

A more extensive exploration of the effect of g on e_r, given in Figure 8 and Tables 4 and 5 at the end of this paper, shows that decreasing g typically improves e_r. This is intuitive, because ILL more finely partitions the input space of a given problem as g decreases. An interesting exception is NF with $g = 0.033$, which features a large spike in e_r. Investigation reveals that the legacy learning rule employed by NF is responsible for this anomaly, because it adjusts model outputs in the direction of the ILL gradient but does not minimize change in model outputs. The NF learning rule can produce very large targets after many iterations, resulting in very high retention error.

5.2.2 The Effect of Dimensionality Reduction on Retention

Tables 6-8, at the end of this paper, portray the effect of ILL dimensionality reduction on retention error e_r. Random datasets are generated with the same procedure described in Section 5.2, varying only by number of attributes d_0. Models tested in Section 5.2.1 are compared, with varying reduced dimensionalities d_N for ILL models. Dimensionality is reduced with a random projection, as described in Section 3.4.

Even massive dimensionality reduction, up to a factor of 40, shows little effect on the retention ability of ILL. ILL models achieve $e_r < 0.14$ on all datasets and $e_r < 0.01$ with most underlying models. Even the NF model, which lacks an underlying model capable of independently retaining data, achieves low e_r.

We note an interesting observation: increasing d_0 has minimal effect on e_r, and many models even achieve lower e_r with higher d_0. With more dimensions, samples are less likely to have high similarity and are thus easier to differentiate and retain. With this phenomena, ILL can retain learned data on problems of high dimensionality, even with significant dimensionality reduction.

5.3 Incremental Classification and Regression

While retention ability is essential for incremental learning, most applications of supervised learning require effective generalization. ILL is compared to a number of state-of-the-art learning models on a variety of datasets, including classification, regression, and image datasets. However, each model is trained incrementally, on 10 samples, before training on the next 10 samples. Models do not have access to the entire training set during any one training step. We show that ILL effectively generalizes while learning incrementally.

5.3.1 Datasets

The popular iris dataset (Lichman, 2013) contains real value attributes describing various characteristics of a flower: sepal width, septal length, petal width, and petal length. The goal is to classify a flower into three types of iris. The iris dataset provides a light challenge with low dimensionality and low noise.

The Wisconsin breast cancer diagnostic dataset (Cancer) (Lichman, 2013) contains real valued attributes of cell nuclei measured from images of fine needle aspirate from breast mass. The goal is to classify each mass as malignant or benign. The cancer dataset provides a moderate classification challenge with medium dimensionality.

California housing (CalHousing) (Pace, ; Pace and Barry, 1997; Huang et al., 2005; Huang et al., 2006) is a regression dataset predicting house value based on neighborhood and house statistics. Attributes are obtained using all block groups in California from the 1990 census. CalHousing provides a comparison of incremental learning ability on regression datasets.

The US postal service hand-written digit dataset (USPS) (Hull, 1994) contains 16×16 grey-scale images of hand written digits, gathered at the Center of Excellence in Document Analysis and Recognition (CEDAR) at SUNY Buffalo, as part of a project sponsored by the US postal service. Images are scanned from post office mail and contain a multitude of writers and styles. USPS compares incremental learning ability on medium dimensional image datasets with a low number of classes.

The CMU Pose, Illumination, and Expression (PIE) image classification dataset (Sim et al., 2002) contains facial images of 68 people in 13 different poses, 43 different illumination conditions, and with 4 different expressions. The PIE dataset benchmarked here is a subset of the original containing 67 people in the frontal view pose, under 21 illumination conditions with background light off, and a neutral expression. Images are made grey scale and scaled to 30×30 pixels. This dataset aims to predict the person. PIE is our most challenging benchmark dataset, given its high dimensional problem space and many classes.

Table 2 presents the problem type of each dataset, number of samples in the dataset, number of samples in the training set, attributes in each sample, and number of classes or regression values. For classification datasets, the training set is given an even distribution of classes. Note that these datasets are used as incremental learning datasets by training each model incrementally, with small batches. To emulate the challenges of incremental learning, no measures are taken to balance class distribution between each training batch, but the training set is shuffled to prevent artificial clumping of classes.

Table 2: Benchmark Datasets

Dataset	Type	Samples	Training Samples	Attributes	Classes/Outputs
Iris	Classification	150	90	4	3
Cancer	Classification	569	280	30	2
CalHousing	Regression	20640	500	8	1
USPS	Image Classification	11000	500	256	10
PIE	Image Classification	1407	670	900	67

5.3.2 Generalization Comparison

Table 3 compares the generalization ability of ILL and several comparison models detailed in Section 5.1. Each model is tested on the datasets presented in Section 5.3.1. However, each model is trained incrementally, in batches of 10 samples. Each batch of 10 samples is presented for training until the model converges before presenting the next 10 samples. Models do not have access to the entire training set during any one iteration. This forces models to learn incrementally, simultaneously testing incremental learning and generalization ability. Accuracy on training (Tr) and testing (Te) sets is presented for classification datasets, and mean squared error (MSE) is presented for regression datasets. Accuracy is defined as percent of correct classifications. Training accuracy and MSE is measured over the entire training set using the final incrementally trained model. Each model is trained and correspondingly tested 10 times on each dataset. The mean and standard deviation (SD) of these runs is presented. Hyperparameters selected for each model, via the hyperparameter search detailed in Section 5.1.1, are also included.

On every dataset, an ILL model achieves the highest testing accuracy or lowest testing MSE. With few exceptions, ILL models improve the performance of their underlying model, regardless of dataset. Even batch learning models, like RF and KNN, can achieve over 90% accuracy with ILL on some datasets. On most datasets, ILL maintains consistent performance with low standard deviation.

We see that choice of underlying model is still essential to maximize accuracy. However, there is a strong correlation between performance of an underlying model and performance with ILL. This greatly eases model selection. If a model proves effective at a given problem, wrapping this model in ILL maintains its effectiveness while improving incremental learning ability.

Among all datasets, we see a variety of ILL hyperparameters. For a given dataset, grid spacing g is relatively consistent among ILL models, indicating that dataset determines effective grid spacing more than underlying model. While the radius neighborhood function sees occasional use, the KNN neighborhood is more common, with $k = 5$ the most frequent choice of k. We see little correlation between choice of reduced neighborhood dimensionality d_N and performance. This indicates that optimizing the d_N hyperparameter may be unnecessary.

6 Conclusion

In Section 5 we see the phenomenal incremental learning ability of ILL, its ability to improve generalization accuracy of its underlying model, and that it provides extreme data retention ability. Incremental learning with high data retention enables efficient retraining of models when new data is available. Real-time learning becomes efficient because incremental learning models do not have to retrain on an entire, rapidly growing, dataset. Likewise, real-time reinforcement learning with complex models becomes feasible.

Rather than provide a single incremental model and its related prior, ILL wraps an existing model, adding the ability to retain previously learned data while maintaining the properties of its underlying model. With a cartesian grid of points, ILL delegates learning to models corresponding to points near an input vector. This local learning mechanism enables the effective incremental learning of an ILL ensemble. Efficient algorithms for discovering points in the neighborhood around a given vector, combined with lazy evaluation, allow ILL to maintain and manage an infinite cartesian grid of models with finite computation and memory. This methodology allows ILL to satisfy our criteria for an incremental learning model, given in Section 1, and re-iterated here:

1. It must learn from presented data.

Table 3: Model Generalization Comparison

Dataset	Model	Tr Mean	Tr SD	Te Mean	Te SD	Hyperparameters
	ILL(MLP)	96.67%	0.00%	93.33%	0.00%	g: 1.000, f_N: 5-NN, n_h: 4
	ILL(RBF)	96.67%	0.00%	93.33%	0.00%	g: 1.000, f_N: 5-NN, n: 32, v: 0.125
	ILL(Lasso)	96.67%	0.00%	**96.67%**	0.00%	g: 0.681, f_N: 5-NN
	ILL(SVM)	95.56%	0.00%	91.67%	0.00%	g: 1.000, f_N: 3-NN
	ILL(RF)	95.56%	0.00%	**96.67%**	0.00%	g: 0.681, f_N: 5-NN, t: 1
	ILL(KNN)	95.56%	0.00%	91.67%	0.00%	g: 1.000, f_N: Radius, k: 1
Iris	NF	95.56%	0.00%	**96.67%**	0.00%	g: 0.681, f_N: 3-NN
	MLP	95.56%	0.00%	93.50%	0.90%	n_h: 4
	RBF	91.89%	4.04%	90.50%	4.48%	n: 32, v: 0.125
	Lasso	65.56%	0.00%	66.67%	0.00%	
	SVM	66.56%	0.33%	66.67%	0.00%	
	RF	66.67%	0.00%	66.67%	0.00%	t: 1
	KNN	66.67%	0.00%	66.67%	0.00%	k: 1
	ILL(MLP)	87.71%	3.16%	**81.94%**	4.73%	g: 0.042, d_N: 1, f_N: 5-NN, n_h: 3
	ILL(RBF)	81.25%	14.75%	72.15%	21.29%	g: 0.042, d_N: 1, f_N: 3-NN, n: 29, v: 0.138
	ILL(Lasso)	81.75%	5.00%	70.83%	7.05%	g: 0.042, d_N: 1, f_N: 1-NN
	ILL(SVM)	78.82%	8.35%	66.37%	10.77%	g: 0.042, d_N: 1, f_N: 3-NN
	ILL(RF)	59.96%	9.19%	39.07%	13.45%	g: 0.042, d_N: 2, f_N: 3-NN, t: 1
	ILL(KNN)	60.14%	6.80%	42.01%	9.27%	g: 0.042, d_N: 2, f_N: 5-NN, k: 1
Cancer	NF	54.07%	8.08%	31.52%	12.17%	g: 0.294, d_N: 2, f_N: 5-NN
	MLP	81.29%	1.75%	72.18%	2.60%	n_h: 3
	RBF	85.29%	5.60%	77.75%	9.09%	n: 29, v: 0.138
	Lasso	65.00%	0.00%	48.79%	0.00%	
	SVM	53.89%	0.25%	32.15%	0.48%	
	RF	50.00%	0.00%	24.91%	0.00%	t: 1
	KNN	50.00%	0.00%	24.91%	0.00%	k: 1
	ILL(MLP)	0.161	0.030	0.180	0.032	g: 1.000, d_N: 1, f_N: 3-NN, n_h: 6
	ILL(RBF)	0.188	0.050	0.209	0.054	g: 0.180, d_N: 2, f_N: 5-NN, n: 95, v: 0.042
	ILL(Lasso)	0.113	0.016	**0.117**	0.015	g: 0.455, d_N: 2, f_N: 5-NN
	ILL(SVM)	0.135	0.029	0.146	0.030	g: 1.000, d_N: 1, f_N: 3-NN
	ILL(RF)	0.186	0.053	0.194	0.050	g: 0.455, d_N: 2, f_N: 5-NN, t: 29
	ILL(KNN)	0.214	0.049	0.207	0.044	g: 1.000, d_N: 2, f_N: 5-NN, k: 4
CalHousing	NF	1.573	3.526	1.561	3.503	g: 0.455, d_N: 2, f_N: 5-NN
	MLP	0.119	0.003	0.136	0.003	n_h: 6
	RBF	0.235	0.036	0.253	0.036	n: 95, v: 0.042
	Lasso	0.111	0.000	0.118	0.000	
	SVM	0.123	0.002	0.132	0.002	
	RF	0.172	0.000	0.188	0.000	t: 29
	KNN	0.280	0.000	0.263	0.000	k: 4
	ILL(MLP)	84.00%	2.19%	74.46%	1.95%	g: 0.180, d_N: 1, f_N: 5-NN, n_h: 14
	ILL(RBF)	24.10%	4.91%	20.83%	4.56%	g: 0.180, d_N: 2, f_N: 3-NN, n: 66, v: 80
	ILL(Lasso)	93.74%	0.71%	**80.63%**	0.81%	g: 0.180, d_N: 2, f_N: 5-NN
	ILL(SVM)	69.30%	2.89%	57.40%	2.22%	g: 0.455, d_N: 1, f_N: Radius
	ILL(RF)	17.58%	1.92%	16.57%	2.17%	g: 0.294, d_N: 2, f_N: 3-NN, t: 23
	ILL(KNN)	23.56%	0.98%	24.36%	1.03%	g: 0.294, d_N: 2, f_N: Radius, k: 1
USPS	NF	24.52%	3.68%	15.44%	2.25%	g: 0.042, d_N: 2, f_N: 1-NN
	MLP	78.38%	3.41%	69.59%	3.36%	n_h: 14
	RBF	35.02%	3.76%	30.64%	3.72%	n: 66, v: 80
	Lasso	83.96%	0.85%	72.46%	0.26%	
	SVM	66.08%	2.26%	56.69%	1.86%	
	RF	15.60%	0.00%	14.09%	0.00%	t: 23
	KNN	23.00%	0.00%	24.51%	0.00%	k: 1
	ILL(MLP)	12.06%	1.79%	7.72%	1.10%	g: 0.042, d_N: 2, f_N: Radius, n_h: 22
	ILL(RBF)	6.16%	0.60%	4.12%	1.00%	g: 0.042, d_N: 2, f_N: 1-NN, n: 112, v: 240
	ILL(Lasso)	78.15%	2.11%	75.59%	2.10%	g: 0.455, d_N: 2, f_N: 5-NN
	ILL(SVM)	84.15%	2.34%	**81.59%**	2.42%	g: 0.099, d_N: 1, f_N: 5-NN
	ILL(RF)	96.88%	0.98%	4.74%	0.77%	g: 0.001, d_N: 2, f_N: 5-NN, t: 15
	ILL(KNN)	7.60%	0.66%	6.34%	1.12%	g: 0.042, d_N: 1, f_N: 3-NN, k: 1
PIE	NF	96.70%	0.86%	4.79%	0.75%	g: 0.001, d_N: 2, f_N: 5-NN
	MLP	6.39%	1.87%	5.43%	1.68%	n_h: 22
	RBF	1.75%	0.31%	1.29%	0.26%	n: 112, v: 240
	Lasso	79.18%	2.32%	77.57%	2.06%	
	SVM	72.61%	4.85%	69.89%	4.10%	
	RF	4.93%	0.00%	3.26%	0.00%	t: 15
	KNN	5.82%	0.00%	5.56%	0.00%	k: 1

2. It must retain data without requiring access to previous data.

3. Its memory usage must be finite and bounded regardless of how many or how often samples are presented.

As an ensemble, the performance of ILL is a function of its underlying model and the number of underlying models activated. Despite maintaining a theoretically infinite grid of models, the neighborhood mechanism described in Section 3.1 allows for easy performance improvements when necessary, by shrinking the ILL neighborhood for faster activation and training with fewer underlying models per activation. Furthermore, by combining multiple underlying models, ILL can achieve high predictive power with simpler underlying models.

This unique ensemble furthers the state-of-the-art in incremental learning, provides robust tuning to solve an array of problems, and integrates seamlessly with existing state-of-the-art incremental and batch models for supervised learning. The customization of ILL provides numerous avenues for extension: neighborhood function (Section 3.1), similarity function (Section 3.2), and underlying model can be interchanged and extended, allowing for extensive improvements and exploration in future works. ILL does not supercede or replace the state-of-the-art but instead extends and improves it. Future advances in machine learning can seamlessly integrate with ILL to add or improve incremental learning ability.

6.1 Future Works

A great advantage that ILL has over many existing incremental learning methods is its constant memory scaling with regard to number of training instances. This provides ILL with the ability to learn on datasets containing more than millions of samples. However, experimental results with massive datasets is beyond the scope of this paper, due to the computational requirements of our hyperparameter optimization strategy. Instead, our experimental analysis emulates the difficulties of big data by partitioning datasets into many small subsets, providing evidence that ILL can learn efficiently with massive datasets. Future exploration of ILL can examine performance with big datasets containing millions of samples and potentially further the state-of-the-art in big data, image recognition, social media analysis, personalized medicine, and marketing.

Table 4: The Effect of Grid Spacing on Retention, Part 1

Model	Mean e_r	Hyperparameters
ILL(MLP)	0.3837	g: 0.500, f_N: 1-NN, n_h: 46
ILL(MLP)	0.3804	g: 0.399, f_N: 1-NN, n_h: 46
ILL(MLP)	0.2982	g: 0.310, f_N: 1-NN, n_h: 46
ILL(MLP)	0.2606	g: 0.232, f_N: 1-NN, n_h: 46
ILL(MLP)	0.1952	g: 0.166, f_N: 1-NN, n_h: 46
ILL(MLP)	0.1376	g: 0.110, f_N: 1-NN, n_h: 46
ILL(MLP)	0.0963	g: 0.066, f_N: Radius, n_h: 46
ILL(MLP)	0.0630	g: 0.033, f_N: 3-NN, n_h: 46
ILL(MLP)	0.0212	g: 0.011, f_N: 3-NN, n_h: 46
ILL(MLP)	0.0036	g: 0.001, f_N: 5-NN, n_h: 46
ILL(RBF)	0.3489	g: 0.500, f_N: 1-NN, n: 118, v: 0.034
ILL(RBF)	0.3751	g: 0.399, f_N: Radius, n: 118, v: 0.034
ILL(RBF)	0.2552	g: 0.310, f_N: 1-NN, n: 118, v: 0.034
ILL(RBF)	0.2217	g: 0.232, f_N: 1-NN, n: 118, v: 0.034
ILL(RBF)	0.1771	g: 0.166, f_N: 1-NN, n: 118, v: 0.034
ILL(RBF)	0.1251	g: 0.110, f_N: 1-NN, n: 118, v: 0.034
ILL(RBF)	0.0832	g: 0.066, f_N: 1-NN, n: 118, v: 0.034
ILL(RBF)	0.0514	g: 0.033, f_N: 3-NN, n: 118, v: 0.034
ILL(RBF)	0.0238	g: 0.011, f_N: 3-NN, n: 118, v: 0.034
ILL(RBF)	0.0069	g: 0.001, f_N: 5-NN, n: 118, v: 0.034
ILL(Lasso)	0.2676	g: 0.500, f_N: 3-NN
ILL(Lasso)	0.2673	g: 0.399, f_N: 3-NN
ILL(Lasso)	0.2527	g: 0.310, f_N: 3-NN
ILL(Lasso)	0.2469	g: 0.232, f_N: 3-NN
ILL(Lasso)	0.2380	g: 0.166, f_N: 3-NN
ILL(Lasso)	0.2260	g: 0.110, f_N: 3-NN
ILL(Lasso)	0.2205	g: 0.066, f_N: 3-NN
ILL(Lasso)	0.2105	g: 0.033, f_N: 5-NN
ILL(Lasso)	0.2037	g: 0.011, f_N: 5-NN
ILL(Lasso)	0.2004	g: 0.001, f_N: 5-NN
ILL(SVM)	0.2507	g: 0.500, f_N: 3-NN
ILL(SVM)	0.2397	g: 0.399, f_N: 3-NN
ILL(SVM)	0.2063	g: 0.310, f_N: 3-NN
ILL(SVM)	0.1870	g: 0.232, f_N: 3-NN
ILL(SVM)	0.1555	g: 0.166, f_N: 3-NN
ILL(SVM)	0.1296	g: 0.110, f_N: 3-NN
ILL(SVM)	0.1069	g: 0.066, f_N: 3-NN
ILL(SVM)	0.0927	g: 0.033, f_N: 3-NN
ILL(SVM)	0.0781	g: 0.011, f_N: 3-NN
ILL(SVM)	0.0684	g: 0.001, f_N: 5-NN
ILL(RF)	0.3718	g: 0.500, f_N: 1-NN, t: 28
ILL(RF)	0.4078	g: 0.399, f_N: Radius, t: 28
ILL(RF)	0.2820	g: 0.310, f_N: 1-NN, t: 28
ILL(RF)	0.2387	g: 0.232, f_N: 1-NN, t: 28
ILL(RF)	0.1757	g: 0.166, f_N: 1-NN, t: 28
ILL(RF)	0.1219	g: 0.110, f_N: 1-NN, t: 28
ILL(RF)	0.0911	g: 0.066, f_N: 3-NN, t: 28
ILL(RF)	0.0513	g: 0.033, f_N: 3-NN, t: 28
ILL(RF)	0.0234	g: 0.011, f_N: 3-NN, t: 28
ILL(RF)	0.0078	g: 0.001, f_N: 5-NN, t: 28

Table 5: The Effect of Grid Spacing on Retention, Part 2

Model	Mean e_r	Hyperparameters
ILL(KNN)	0.3845	g: 0.500, f_N: 1-NN, k: 5
ILL(KNN)	0.3874	g: 0.399, f_N: 1-NN, k: 5
ILL(KNN)	0.2985	g: 0.310, f_N: 1-NN, k: 5
ILL(KNN)	0.2508	g: 0.232, f_N: 1-NN, k: 5
ILL(KNN)	0.1878	g: 0.166, f_N: 1-NN, k: 5
ILL(KNN)	0.1297	g: 0.110, f_N: 1-NN, k: 5
ILL(KNN)	0.0927	g: 0.066, f_N: Radius, k: 5
ILL(KNN)	0.0516	g: 0.033, f_N: 3-NN, k: 5
ILL(KNN)	0.0243	g: 0.011, f_N: 3-NN, k: 5
ILL(KNN)	0.0078	g: 0.001, f_N: 5-NN, k: 5
NF	0.5063	g: 0.500, f_N: 1-NN
NF	0.5120	g: 0.399, f_N: 1-NN
NF	0.4152	g: 0.310, f_N: 1-NN
NF	0.3499	g: 0.232, f_N: 1-NN
NF	0.2649	g: 0.166, f_N: Radius
NF	0.1806	g: 0.110, f_N: 1-NN
NF	0.1121	g: 0.066, f_N: 1-NN
NF	0.3990	g: 0.033, f_N: 3-NN
NF	0.0200	g: 0.011, f_N: Radius
NF	0.0001	g: 0.001, f_N: 1-NN

Table 6: The Effect of Dimensionality Reduction on Retention, Part 1

d_0	Model	d_N	Mean e_r	Hyperparameters
40	ILL(MLP)	4	0.00626	g: 0.001, d_N: 4, f_N: 5-NN, n_h: 20
		2	0.00462	g: 0.001, d_N: 2, f_N: 5-NN, n_h: 20
		1	0.00702	g: 0.001, d_N: 1, f_N: 5-NN, n_h: 20
	ILL(RBF)	4	0.01054	g: 0.001, d_N: 4, f_N: 5-NN, n: 107, v: 0.037
		2	0.00888	g: 0.001, d_N: 2, f_N: 5-NN, n: 107, v: 0.037
		1	0.00736	g: 0.001, d_N: 1, f_N: 5-NN, n: 107, v: 0.037
	ILL(Lasso)	4	0.03730	g: 0.001, d_N: 4, f_N: 5-NN
		2	0.03682	g: 0.001, d_N: 2, f_N: 5-NN
		1	0.03659	g: 0.001, d_N: 1, f_N: 5-NN
	ILL(SVM)	4	0.03112	g: 0.001, d_N: 4, f_N: 5-NN
		2	0.02959	g: 0.001, d_N: 2, f_N: 5-NN
		1	0.02365	g: 0.001, d_N: 1, f_N: 5-NN
	ILL(RF)	4	0.01063	g: 0.001, d_N: 4, f_N: 5-NN, t: 23
		2	0.00824	g: 0.001, d_N: 2, f_N: 5-NN, t: 23
		1	0.01311	g: 0.001, d_N: 1, f_N: 3-NN, t: 23
	ILL(KNN)	4	0.01162	g: 0.001, d_N: 4, f_N: 5-NN, k: 5
		2	0.00841	g: 0.001, d_N: 2, f_N: 5-NN, k: 5
		1	0.01766	g: 0.001, d_N: 1, f_N: 3-NN, k: 5
	NF	4	0.00000	g: 0.001, d_N: 4, f_N: 1-NN
		2	0.00000	g: 0.001, d_N: 2, f_N: 1-NN
		1	0.00890	g: 0.001, d_N: 1, f_N: 1-NN
	MLP	N/A	0.16619	n_h: 20
	RBF	N/A	0.15911	n: 107, v: 0.037
	Lasso	N/A	0.07786	
	SVM	N/A	0.07184	
	RF	N/A	0.38676	t: 23
	KNN	N/A	0.36903	k: 5

Table 7: The Effect of Dimensionality Reduction on Retention, Part 2

d_0	Model	d_N	Mean e_r	Hyperparameters
20	ILL(MLP)	4	0.00624	g: 0.042, d_N: 4, f_N: 5-NN, n_h: 10
		2	0.00479	g: 0.001, d_N: 2, f_N: 5-NN, n_h: 10
		1	0.00721	g: 0.001, d_N: 1, f_N: 3-NN, n_h: 10
	ILL(RBF)	4	0.01256	g: 0.099, d_N: 4, f_N: 5-NN, n: 122, v: 0.033
		2	0.00838	g: 0.001, d_N: 2, f_N: 5-NN, n: 122, v: 0.033
		1	0.00685	g: 0.001, d_N: 1, f_N: 5-NN, n: 122, v: 0.033
	ILL(Lasso)	4	0.04291	g: 0.042, d_N: 4, f_N: 5-NN
		2	0.04075	g: 0.001, d_N: 2, f_N: 5-NN
		1	0.03849	g: 0.001, d_N: 1, f_N: 5-NN
	ILL(SVM)	4	0.03794	g: 0.001, d_N: 4, f_N: 5-NN
		2	0.03605	g: 0.001, d_N: 2, f_N: 5-NN
		1	0.03199	g: 0.001, d_N: 1, f_N: 5-NN
	ILL(RF)	4	0.01144	g: 0.001, d_N: 4, f_N: 5-NN, t: 10
		2	0.01022	g: 0.001, d_N: 2, f_N: 3-NN, t: 10
		1	0.02302	g: 0.001, d_N: 1, f_N: 3-NN, t: 10
	ILL(KNN)	4	0.01047	g: 0.042, d_N: 4, f_N: 5-NN, k: 5
		2	0.00938	g: 0.001, d_N: 2, f_N: 5-NN, k: 5
		1	0.01406	g: 0.001, d_N: 1, f_N: 3-NN, k: 5
	NF	4	0.00000	g: 0.001, d_N: 4, f_N: 1-NN
		2	0.00000	g: 0.001, d_N: 2, f_N: 1-NN
		1	0.00755	g: 0.001, d_N: 1, f_N: 1-NN
	MLP	N/A	0.22435	n_h: 10
	RBF	N/A	0.12337	n: 122, v: 0.033
	Lasso	N/A	0.10928	
	SVM	N/A	0.11395	
	RF	N/A	0.43069	t: 10
	KNN	N/A	0.41234	k: 5

Table 8: The Effect of Dimensionality Reduction on Retention, Part 3

d_0	Model	d_N	Mean e_r	Hyperparameters
5	ILL(MLP)	4	0.00689	g: 0.042, d_N: 4, f_N: 5-NN, n_h: 58
		2	0.00584	g: 0.001, d_N: 2, f_N: 5-NN, n_h: 58
		1	0.00726	g: 0.001, d_N: 1, f_N: 3-NN, n_h: 58
	ILL(RBF)	4	0.01039	g: 0.042, d_N: 4, f_N: 5-NN, n: 111, v: 0.036
		2	0.00838	g: 0.001, d_N: 2, f_N: 5-NN, n: 111, v: 0.036
		1	0.00625	g: 0.001, d_N: 1, f_N: 5-NN, n: 111, v: 0.036
	ILL(Lasso)	4	0.13247	g: 0.001, d_N: 4, f_N: 5-NN
		2	0.13121	g: 0.042, d_N: 2, f_N: 3-NN
		1	0.11402	g: 0.001, d_N: 1, f_N: 5-NN
	ILL(SVM)	4	0.04400	g: 0.001, d_N: 4, f_N: 5-NN
		2	0.04502	g: 0.042, d_N: 2, f_N: 5-NN
		1	0.03851	g: 0.001, d_N: 1, f_N: 5-NN
	ILL(RF)	4	0.01097	g: 0.042, d_N: 4, f_N: 5-NN, t: 27
		2	0.00900	g: 0.001, d_N: 2, f_N: 5-NN, t: 27
		1	0.00902	g: 0.001, d_N: 1, f_N: 5-NN, t: 27
	ILL(KNN)	4	0.01140	g: 0.042, d_N: 4, f_N: 5-NN, k: 5
		2	0.01058	g: 0.001, d_N: 2, f_N: 3-NN, k: 5
		1	0.00820	g: 0.001, d_N: 1, f_N: 5-NN, k: 5
	NF	4	0.00000	g: 0.001, d_N: 4, f_N: 1-NN
		2	0.00000	g: 0.001, d_N: 2, f_N: 1-NN
		1	0.00567	g: 0.001, d_N: 1, f_N: 1-NN
	MLP	N/A	0.42845	n_h: 58
	RBF	N/A	0.12565	n: 111, v: 0.036
	Lasso	N/A	0.21315	
	SVM	N/A	0.21540	
	RF	N/A	0.42173	t: 27
	KNN	N/A	0.39593	k: 5

A Radius Points Algorithm With Floating Point Math

When Algorithm 1 is implemented with floating point math (Whitehead and Fit-Florea, 2011), as is common in computer systems, numerical precision errors can result in points with insignificant differences, when points would be identical with infinite precision. For example, given grid spacing $g = 0.1$, radius $r_1 = 0.1$, and center $\vec{c} = [0.61]$, using double-precision 64-bit format IEEE 754 math, Algorithm 1 will return points $[g\lceil(\vec{c}_1 - r_1)/g\rceil = 0.6000000000000001]$ and $[g\lceil(\vec{c}_1 - r_1)/g\rceil + g = 0.7000000000000001]$. If we expand the radius to $r_2 = 0.11$, Algorithm 1 will return points $[g\lceil(\vec{c}_1 - r_2)/g\rceil = 0.5]$, $[g\lceil(\vec{c}_1 - r_2)/g\rceil + g = 0.6]$, and $[g\lceil(\vec{c}_1 - r_2)/g\rceil + g + g = 0.7]$. Although the larger radius r_2 should encompass at least the same points as the smaller radius r_1, limited precision in double-precision 64-bit format IEEE 754 results in completely different points for r_1 and r_2. Note that the radius parameter varies regularly during recursion of the radius points procedure and with various neighborhood functions utilizing Algorithm 1. Rounding each component to the number of significant digits in g avoids precision errors.

Compliance with ethical standards

Author Justin Lovinger declares that he has no conflict of interest. Author Iren Valova declares that she has no conflict of interest. Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

Balsubramani, A., Dasgupta, S., and Freund, Y. (2013). The fast convergence of incremental pca. In *Advances in Neural Information Processing Systems*, pages 3174–3182.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bingham, E. and Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM.

Broomhead, D. S. and Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document.

Cederborg, T., Li, M., Baranes, A., and Oudeyer, P.-Y. (2010). Incremental local online gaussian mixture regression for imitation learning of multiple tasks. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 267–274. IEEE.

Cohen, M. B., Elder, S., Musco, C., Musco, C., and Persu, M. (2015). Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 163–172. ACM.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cyc (2008). Graphic showing the maximum separating hyperplane and the margin. `https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png`.

Díaz-Uriarte, R. and De Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):1.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Gepperth, A. and Hammer, B. (2016). Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Hagan, M. T., Demuth, H. B., Beale, M. H., et al. (1996). *Neural network design*. Pws Pub. Boston.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*.

Hazan, E. et al. (2016). Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325.

Ho, T. K. (1995). Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE.

Huang, G.-B., Chen, L., Siew, C. K., et al. (2006). Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks*, 17(4):879–892.

Huang, G.-B., Saratchandran, P., and Sundararajan, N. (2005). A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation. *IEEE Transactions on Neural Networks*, 16(1):57–67.

Huang, G.-B., Zhou, H., Ding, X., and Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554.

Jiang, F., Sui, Y., and Cao, C. (2013). An incremental decision tree algorithm based on rough sets and its application in intrusion detection. *Artificial Intelligence Review*, pages 1–14.

Johnson, W. B. and Lindenstrauss, J. (1984). Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1.

Kalteh, A. M., Hjorth, P., and Berndtsson, R. (2008). Review of the self-organizing map (som) approach in water resources: Analysis, modelling and application. *Environmental Modelling & Software*, 23(7):835–845.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Lange, S. and Riedmiller, M. (2010). Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Lovinger, J. and Valova, I. (2016). Neural field: Supervised apportioned incremental learning (sail). In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2500–2506.

Lowe, D. and Broomhead, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex syst*, 2:321–355.

Ma, K. and Ben-Arie, J. (2014). Compound exemplar based object detection by incremental random forest. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 2407–2412. IEEE.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 1.

Milborrow, S. (2011). Titanic decision tree. `https://commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png`.

Montgomery, D. C., Peck, E. A., and Vining, G. G. (2015). *Introduction to linear regression analysis*. John Wiley & Sons.

Muja, M. and Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240.

Neter, J., Kutner, M. H., Nachtsheim, C. J., and Wasserman, W. (1996). *Applied linear statistical models*, volume 4. Irwin Chicago.

Nocedal, J. and Wright, S. J. (2006). Numerical optimization 2nd.

Pace, R. K. California housing. http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html.

Pace, R. K. and Barry, R. (1997). Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297.

Pang, S., Ozawa, S., and Kasabov, N. (2005). Incremental linear discriminant analysis for classification of data streams. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(5):905–914.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

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

Polikar, R., Upda, L., Upda, S. S., and Honavar, V. (2001). Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508.

Pradhan, B. (2013). A comparative study on the predictive ability of the decision tree, support vector machine and neuro-fuzzy models in landslide susceptibility mapping using gis. *Computers & Geosciences*, 51:350–365.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Ramos, F. and Ott, L. (2016). Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent. *The International Journal of Robotics Research*, 35(14):1717–1730.

Rebentrost, P., Mohseni, M., and Lloyd, S. (2014). Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

Ristin, M., Guillaumin, M., Gall, J., and Van Gool, L. (2016). Incremental learning of random forests for large-scale image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(3):490–503.

Rodriguez-Galiano, V. F., Ghimire, B., Rogan, J., Chica-Olmo, M., and Rigol-Sanchez, J. P. (2012). An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 67:93–104.

Rosasco, L. and Villa, S. (2015). Learning with incremental iterative regularization. In *Advances in Neural Information Processing Systems*, pages 1630–1638.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rutkowski, L., Jaworski, M., Pietruczuk, L., and Duda, P. (2014). The cart decision tree for mining data streams. *Information Sciences*, 266:1–15.

Schmid, H. (2013). Probabilistic part-ofispeech tagging using decision trees. In *New methods in language processing*, page 154. Routledge.

Seber, G. A. and Lee, A. J. (2012). *Linear regression analysis*, volume 936. John Wiley & Sons.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

Shi, X., Yang, Y., Guo, Z., and Lai, Z. (2014). Face recognition by sparse discriminant analysis via joint l 2, 1-norm minimization. *Pattern Recognition*, 47(7):2447–2453.

Sim, T., Baker, S., and Bsat, M. (2002). The cmu pose, illumination, and expression (pie) database. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 53–58. IEEE.

Song, W., Zhu, J., Li, Y., and Chen, C. (2016). Image alignment by online robust pca via stochastic gradient descent. *IEEE Transactions on Circuits and Systems for video Technology*, 26(7):1241–1250.

Soudry, D., Di Castro, D., Gal, A., Kolodny, A., and Kvatinsky, S. (2015). Memristor-based multilayer neural networks with online gradient descent training. *IEEE transactions on neural networks and learning systems*, 26(10):2408–2421.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Suykens, J. A. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300.

Tagliaferri, R., Longo, G., Milano, L., Acernese, F., Barone, F., Ciaramella, A., De Rosa, R., Donalek, C., Eleuteri, A., Raiconi, G., et al. (2003). Neural networks in astronomy. *Neural Networks*, 16(3):297–319.

Tan, Y., Wang, J., and Zurada, J. M. (2001). Nonlinear blind source separation using a radial basis function network. *Neural Networks, IEEE Transactions on*, 12(1):124–134.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

Tibshirani, R. (2011). Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):273–282.

Tóth, L. (2013). Phone recognition with deep sparse rectifier neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6985–6989. IEEE.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine learning*, 4(2):161–186.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100.

Variddhisaï, T. and Mandic, D. (2017). Online multilinear dictionary learning for sequential compressive sensing. *arXiv preprint arXiv:1703.02492*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge England.

Weinberger, K. Q., Blitzer, J., and Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480.

Weng, J., Zhang, Y., and Hwang, W.-S. (2003). Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):1034–1040.

Whitehead, N. and Fit-Florea, A. (2011). Precision & performance: Floating point and ieee 754 compliance for nvidia gpus. *rn (A+ B)*, 21(1):18749–19424.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., and Ma, Y. (2009). Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227.

Yoshida, Y., Karakida, R., Okada, M., and Amari, S.-i. (2017). Statistical mechanical analysis of online learning with weight normalization in single layer perceptron. *Journal of the Physical Society of Japan*, 86(4):044002.

Zhao, H. and Yuen, P. C. (2008). Incremental linear discriminant analysis for face recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(1):210–221.

Zhao, H., Yuen, P. C., and Kwok, J. T. (2006). A novel incremental principal component analysis and its application for face recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(4):873–886.