Easy and Efficient Hyperparameter Optimization to Address Some Artificial Intelligence “ilities”

Trevor J. Bihl
Air Force Research Laboratory, USA
Trevor.Bihl.2@us.af.mil

Joseph Schoenbeck
The Perduco Group, USA
joe.schoenbeck@theperducogroup.com

Daniel Steeneck & Jeremy Jordan
Air Force Institute of Technology, USA
{Daniel.Steeneck; Jeremy.Jordan}@afit.edu

Abstract

Artificial Intelligence (AI), has many benefits, including the ability to find complex patterns, automation, and meaning making. Through these benefits, AI has revolutionized image processing among numerous other disciplines. AI further has the potential to revolutionize other domains; however, this will not happen until we can address the “ilities”: repeatability, explain-ability, reliability, use-ability, trust-ability, etc. Notably, many problems with the “ilities” are due to the artistic nature of AI algorithm development, especially hyperparameter determination. AI algorithms are often crafted products with the hyperparameters learned experientially. As such, when applying the same algorithm to new problems, the algorithm may not perform due to inappropriate settings. This research aims to provide a straightforward and reliable approach to automatically determining suitable hyperparameter settings when given an AI algorithm. Results, show reasonable performance is possible and end-to-end examples are given for three deep learning algorithms and three different data problems.

1. Introduction

Analytics and machine learning (ML), colloquially termed Artificial Intelligence (AI) [1], are becoming increasingly ubiquitous for classification and prediction across a broad spectrum of applications due to their ability to learn nonlinear patterns in data [1]. Fundamentally, AI/ML are complex algorithms that automate procedures based on statistics and nonlinear optimization [1] [2]. However, as a result of their complexity, issues exist in broadly adopting AI solutions [3].

Throughout the space of AI/ML, users must not only decide which algorithms to use, but the settings for the selected algorithm, also known as hyperparameters. This is a complex trade space due to ML methods being brittle and not robust to conditions outside of those on which they were trained. While attention is now given to hyperparameter selection [4] [5], in general, as mentioned in Mendenhall [6], there are “no hard-and-fast rules” in their selection. In fact, their selection is part of the “art of [algorithm] design” [6], as appropriate hyperparameters can depend heavily on the data under consideration itself. Thus, ML methods themselves are often hand-crafted and require significant expertise and talent to appropriately train and deploy.

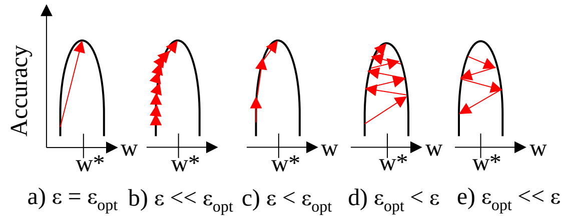

Accuracy
w*
w
a) $\varepsilon = \varepsilon_{opt}$
b) $\varepsilon << \varepsilon_{opt}$
c) $\varepsilon < \varepsilon_{opt}$
d) $\varepsilon_{opt} < \varepsilon$
e) $\varepsilon_{opt} << \varepsilon$

Figure 1. Conceptualization of a general hyperparameter problem: adapted from [2, pp. 312-313].

A conceptual example of this problem is presented in Figure 1, adapted and extended from [2, pp. 312-313]. Here, one is attempting to optimize the learning rate ε, i.e., the rate at which an algorithm converges to a good solution (w*). Ideally, one would want to find $\varepsilon = \varepsilon_{opt}$, the optimal rate, depicted in Figure 1a, but this is largely impossible to find for any meaningful problems due to existence of multiple local optima. A slower, sub-optimal, convergence rate is good when ε is much smaller than ε_{opt}, Figure 1b, but this can take a long time (100s of hours or more in today’s deep learning systems) to train. A reasonable, sub-optimal rate, when $\varepsilon < \varepsilon_{opt}$, Figure 1c, can be ideal since convergence is relatively quick and performance is stable. However, when the learning rate increases above the optimal rate, Figures 1d for $\varepsilon > \varepsilon_{opt}$ and Figure 1e for ε much larger than ε_{opt}, highly oscillatory behaviors can be introduced that bound around local, or global, optima. However, the example in Figure 1 is simple and conceptual; due to the nature of both data and algorithms, there is a naturally complex relationship between hyperparameter settings and results.

The overall opacity of algorithms, required knowledge in tricks of the trade, and the general misunderstandings how algorithms in general work are pervasive [7]. Due to these factors, as noted in [8], issues exist in using ML solutions due to the inability of addressing the ML "ilities" [9], e.g., the reliability, repeatability [8] [3], replicability [3], trust-ability [10], and explain-ability [11] of the algorithms. In general, the "ilities" of can also be further expanded to include general software quality metrics, e.g. ISO 9126, of: functionality, reliability, usability, efficiency, maintainability, and portability [12].

The vast majority of these "ilities" are relate to typical questions users ask of automation [13]:

- What is it doing?
- Why is it doing that?
- What will it do next?

Many of these questions are related to ad hoc algorithm development and deployment methods and a general misunderstanding of capabilities and operations [9]. To understand the "ilities", and likewise address these general questions, [9] presents Shaw's [14] software engineering framework as a general model of maturity of engineering disciplines, presented in Figure 2 which identifies three stages in a field's development: craft, commercial, and professional engineering.

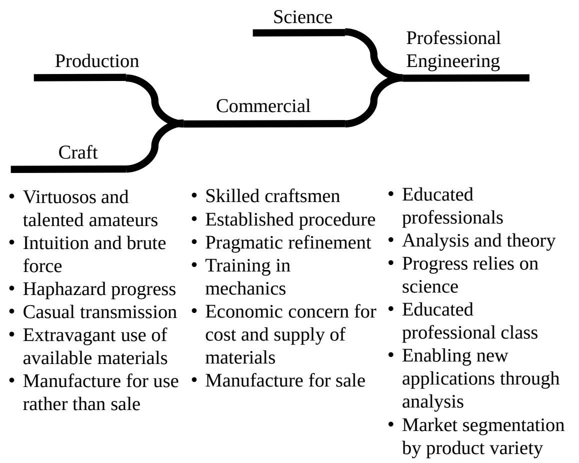

Science
Professional Engineering
Production
Commercial
Craft
• Virtuosos and talented amateurs
• Intuition and brute force
• Haphazard progress
• Casual transmission
• Extravagant use of available materials
• Manufacture for use rather than sale
• Skilled craftsmen
• Established procedure
• Pragmatic refinement
• Training in mechanics
• Economic concern for cost and supply of materials
• Manufacture for sale
• Educated professionals
• Analysis and theory
• Progress relies on science
• Educated professional class
• Enabling new applications through analysis
• Market segmentation by product variety

Figure 2. Shaw's Model of the evolution of engineering disciplines, from [14].

As illustrated in Figure 2, craft involves highly experiential work, which involves tricks of the trade, and various and haphazard approaches to transmit knowledge. When a discipline moves to more established procedures, production, and developing applications for sales, then one has moved to a commercial stage. However, even at this stage, a lack of sound methodologies can limit reproducibility. When one adds scientific approaches to a discipline, one can take commercial art to professional science.

At the professional engineering stage, many of the "ilities" are naturally met since understanding exists about the underlying approaches. As noted in [9], ML development can yield highly sophisticated commercial products yet still be a result of craft and not science as ML requires significant experience to get meaningful results. In contrast, professional engineering disciplines have established methodologies to rigorously develop products [9]. One such approach for this problem in ML is developing a defined process from which to determine the hyperparameters of interest.

While approaches, such as CRISP-DM [15] (CRoss-Industry Standard Process for Data Mining), provide general end-to-end (business concept to deployment) processes for develop data mining solutions, these are high level in nature and do not address the complex space of ML algorithm decisions. What is missing is a general approach that spans the Data, Modeling, and Evaluation layers of CRISP-DM with hyperparameter optimization which enables one to 1) select a dataset, 2) select an AI algorithm from literature or a library, and then 3) automatically determine workable hyperparameter settings without expert algorithmic knowledge.

While general solutions to this problem already exist, they are either 1) automated cloud-based approaches which provide almost unlimited computing resources[1], 2) automated methods which provide a predefined list of algorithms [16], or 3) theoretical approaches to find optimal hyperparameter settings [5]. Each of these solutions presents a challenge: 1) cloud-based solutions introduce security issues can exist if one wishes to analyze proprietary or secured data, 2) automated approaches can be limited in their list of available algorithms, and 3) theoretical optimization approaches require additional subject matter expertise. For example, Snoek et al. [5] propose Bayesian Optimization (BO) for AI hyperparameter optimization; however, implementing [5] requires advanced knowledge of mathematics and algorithms.

The contributions of our paper is fourfold. First, we present a framework to automatically tune AI algorithm hyperparameters, extending upon [5] to create a simple and straightforward process when one is given a general AI algorithm. Next, we show how

[1] For example, DataRobot (https://www.datarobot.com/), AutoML (http://www.automl.org/), and SigOpt (https://sigopt.com/)

this approach provides workable results on different ML recognition tasks. Third, we draw on software engineering body of knowledge, incorporate CRISP-DM and show how this approach facilitates ML moving from craft based implementations to more professional engineering use. Finally, we present a minor contribution by introducing a short taxonomy of AI hyperparameter determination methods. The end result is a further democratization of AI and facilitates wider adoption of AI algorithms.

2. Background

Success or failure in the application of an ML algorithm is a result of multiple factors. Firstly, the quality of the trained ML model is a result of the data itself, the algorithm selected, and model development process. Next, ML performance is heavily linked to assumptions made in the model building process, which includes the initial learning rates as well as architecture decisions.

2.1. AI "ilities"

All of these are requisite to understand, apply, trust, and manage new uses of AI algorithms [11]. Issues abound in AI applications when algorithmic details, references, settings, and training conditions are not mentioned [8]. At first, this appears to be a concern of only repeatability; however, the "ilities" themselves overlap to some degree.

As noted in Zhang [8], to address repeatability and usability problems of AI methods, one needs to provide sufficient details on the algorithm, the data, and experimental conditions. For example, merely reporting that a) a deep learning algorithm was used to develop the b) model on the c) given data is insufficient to be repeatable.

Repeatability and replicability are important in data science [8], and proper reporting involves discussing the data used, what data splitting approaches were applied, and any other data cleaning/wrangling [8]. Beyond this, one needs to mention both the algorithm and any key particulars, e.g. number of layers and nodes in a neural network, in addition to the hyperparameters, initial learning rates, training methods, types of nodes, etc. and etc. From this, one has developed a model which can process data.

However, the results from only one model are insufficient to address reliability and trust-ability concerns since ML algorithms are typically stochastic, and thus appropriate intervals and replications are needed. While additional concerns about explainable and accountable AI have recently extended to extracting fully explainable results [11], we posit that what is of interest for most applications are trustable and reliable AI. Thus, one can consider AI/ML solutions in a similar way as service dogs which, though reliable and trustable, are still opaque since they cannot be queried or questioned.

2.2. Illustrative Examples

An example of good reporting for repeatability is seen in Table 1, from Cireşan et al. [17] in 2012. Table 1 shows a network of some complexity with sufficient details to recreate the overall network structure. In reading the paper, additional assumptions can also be found. However, missing are initial learning rates and other hyperparameters key to repeatable results [4].

Table 1. Example of a Deep Learning architecture and parameters from [17]

Layer	Type		Kern. Size
0	Input	1 map of 95x95 neurons	
1	Convolutional	48 maps of 92x92 neurons	4x4
2	Max pooling	48 maps of 46x46 neurons	2x2
3	Convolutional	48 maps of 42x42 neurons	5x5
4	Max pooling	48 maps of 21x21 neurons	2x2
5	Convolutional	48 maps of 18x18 neurons	4x4
6	Max pooling	48 maps of 9x9 neurons	2x2
7	Convolutional	48 maps of 6x6 neurons	4x4
8	Max pooling	48 maps of 3x3 neurons	2x2
9	Fully connect.	200 neurons	1x1
10	Fully connect.	2 neurons	1x1

Tables such as Table 1 are descriptive and provide most of the details needed to reproduce results. However, for ever larger-and-larger neural networks, a table like this can become cumbersome. One solution is that presented by Cireşan et al. in 2012 [18]. This solution presents the network as an expression, e.g.

$$\text{2x48x48-100C5-MP2-100C5-MP2-100C4-MP2-300N-100N-6N} \tag{1}$$

which encapsulates the general architectural components of the network. Using the Cireşan notation, (1) can be decoded using the following mapping: 2x48x48 represents a network taking inputs of 2 images both of 48x48 pixels, *xCy* a convolutional layer with *x* maps and filters of *y* x *y* weights, *MPy* a max-pooling layer with *y* x *y* pooling size, and *xN* a fully connected layer with x neurons [18].

While both [17] [18] include copious details, they do not (and cannot reasonably due to space) include all possible details needed to best recreate the exact network. However, these details are needed to achieve results similar to those published. For example, consider the LeNet-4 algorithms of [19]; its published accuracy on its benchmark test set is 98.9% [19]. However, while the LeNet-4 architecture itself is known, the hyperparameters that yielded this performance are not. As will be shown in Section 4, these are critical since an accuracy of only 92.23% was achieved when first recreating this network on the same data. However, by using the process presented herein, accuracy of 99.2%,above the published results, can be realized. Notably, the process to reach these results is not manual hyperparameter determination, but an automated process.

2.3. AI Hyperparameter Determination

Hyperparameter determination is an emerging discipline in AI and includes a multitude of methods. A general taxonomy of these approaches is presented in Figure 3. These can largely be separated into model-free and model-based approaches [20].

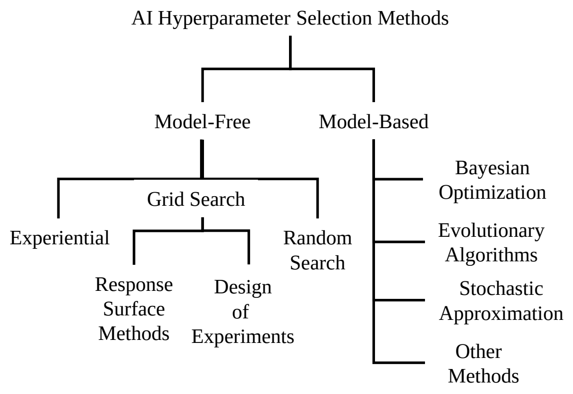


Figure 3. General taxonomy of algorithm setting determination methods, extended from [20]

Model-free approaches can be 1) scientific, e.g. grid searches, or 2) haphazard, e.g. a coder experientially finding settings that "just work," or 3) random searches which use random seeds (notably a competitive method). Grid searches involve creating an experimental design where design points are explored and then one uses either a spreadsheet search or a response surface method to find suitable operating points [21].

Model-based approaches employ what can be considered as a wrapper. Wrappers are essentially another algorithm operating on an outer loop around the function of interest. These methods systematically determine settings for a given algorithm and hopefully converge to a good solution. Important model-based approaches include:

- **Stochastic Approximation** [21], hill climbing where hyperparameters are individually and sequentially changed
- **Evolutionary algorithms** [20], which randomly start, select the best initial results (parents), and then generate multiple possible outcomes (children), and then repeat the process
- **Bayesian optimization (BO)** [5] which treats the objective function as a random function and uses randomly determined hyperparameters to construct a distribution around the results
- **Other approaches** which do not fit cleanly into these three groups, e.g. Radial Basis Functions [22], Hyberband [23], Nedler-Mead [24], and spectral approaches [25].

Beyond this work, further approaches include extensions of BO and combinations of methods. Currently, BO is one of the most competitive hyperparameter optimization methods [5]; however, it should be mentioned that some recently developed methods claim to outperform BO [23] [24]. Despite recent advances, the authors focus on BO since it is readily available, reliable, and well known.

2.4. Bayesian Optimization (BO)

BO tends to find reasonably good choices of hyperparameters [5]. Let $h_{i,j}$ be the value of the i^{th} hyperparameter of the algorithm at the jth evaluation of the algorithm, and let $\boldsymbol{h}_j$ be vector of these hyperparameters. Additionally, $h_{i,j}$ is in the bounded set $\mathcal{H}_i$, which can be continuous or integer valued. Let $f(\boldsymbol{h}_j)$ be the unknown fuction of performance measure of interest of the algorithm versus choice of hyperparameters. Note that $f(\boldsymbol{h}_j)$ is stochastic in nature as it is depends on the training set of data, which is randomly selected. Let $y_j = f(\boldsymbol{h}_j)$ and let $\{y_j, h_j\}_{j=1}^{n}$ be a sequence of y_j and h_j pairs. Based on this sequence, a Gaussian process can be fit to $f(\cdot)$, denoted by $GP\left(\{y_j, h_j\}_{j=1}^{n}\right)$, which in Matlab is done using fitrgp. Finally, an $acquision\ function$, denoted by $a\left(\boldsymbol{h}\middle|GP\left(\{y_j, h_j\}_{j=1}^{n}\right)\right)$ is maximized to find a new set of candiate hyperpameters. The function $a(\cdot)$ can be chosen by the BO designer, but common choices are expected improvement, probability of improvement, and lower confidence bound; herein, the expected improvement acquistion function was used.

The broad outline of BO is given by the following steps:

1. Obtain n_0 initial evaulations of $f(\cdot)$ at randomly selected values of hyperparameters

within the specified hyperparameter bounds. Set $k = 0$.

2. Fit a Gaussian Process onto $\{y_j, h_j\}_{j=1}^{n_k}$, denoted as $GP\left(\{y_j, h_j\}_{j=1}^{n_k}\right)$.
3. Set $\boldsymbol{h}_{j+1} = \underset{\boldsymbol{h}}{\operatorname{argmax}}\, a\left(\boldsymbol{h}\middle|GP\left(\{y_j, h_j\}_{j=1}^{n_k}\right)\right)$
4. Evaluate $y_{j+1} = f(\boldsymbol{h}_{j+1})$, set $n_k = j + 1$ and $k = k + 1$. If termination criteria τ is not met, go-to step 2.

Practically, BO is implemented in many software packages, e.g. bayesopt in Matlab [26], as used herein, and hyperopt in Python [27].

3. Easy and Efficient Hyperparameter Optimization for Initial Settings

Historically, developing AI solutions was the domain of the human expert who implemented the machine learning algorithm. Recently, the mindset has shifted to allowing statistical optimization techniques to assist in finding the best hyperparameters for the algorithm. However, the modeler still must wisely set search spaces, interpret the results, and refine the searches with knowledge of the objective.

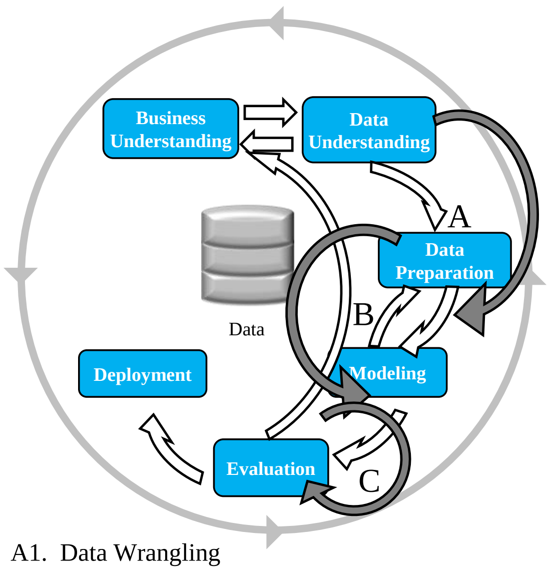
While the BO process in Section 2.3 has been shown to be highly effective in tuning ML algorithms, c.f. [5], it still requires some experience in applying to a ML problem. The authors thus propose that one does not need to report all initial hyperparameter settings, but rather:

1. The architecture, algorithms and software
2. The data and data splitting methods
3. The hyperparameter determination method and initialization settings
4. Results with sufficient replications and intervals.

The authors aim to codify these components by introducing a simple workflow which then improves the general CRISP-DM process with an aim towards addressing the “ilities”.

While CRISP-DM includes a step for revising parameter settings within the Modeling task and Assess Model output, this can be a discipline in itself as discussed in Section 2.3. Of interest for this paper are the general Data Preparation, Modeling, and Evaluate steps of CRISP-DM. To address the hyperparameter problem, the authors overlay their proposed solution onto the CRISP-DM process in Figure 4. Here, the additions to include: A) preprocessing, B) initial training and optimization, and C) evaluating results.

A1. Data Wrangling This step implies the collection and preconditioning of data so it can be analyzed by algorithms. It is estimated that this step can consume 80% of a data scientists time since data quality is key to further successes [28]. This step also involves dividing the data into a training set, for model development, and a testing set for model verification. No consensus exists except that: 1) the model is not trained on the test set, 2) approaches to dividing are well stated (random, deterministic) and discussed, and 3) percentages of the total data are reasonable (generally 10-50% for testing) [8]. Beyond these matters, data wrangling is outside the scope of this paper, and solid methodologies can be found in [28].



A1. Data Wrangling
A2. Select ML Architecture
B1. Train ML Model Using Default Weights
B2. Optimize Hyperparameters
C. Test & Compare Optimized Models

Figure 4. CRISP-DM Model overlaid with general steps (A, B, C) to develop ML solutions

A2. Select ML Architecture involves finding the desired, prescribed/given, or a suitable algorithm to explore for the data. This step can involve significant research in itself. For an example, we will consider MATLAB code from the MATLAB example “Classify Fashion Items with a Convolutional Neural Network” [29]. Here, a deep learning neural network is used to classify grayscale images of clothes.

Using notation presented in Table 2, adapted from the Cireşan notation of (1), the example algorithm can be represented as

28x28-8C3-BN-ReLu-MP2_2-10N-SM-CL (2)

which is decoded per Table 2, an extension of the notation in [18].

Table 2. Brief handbook of Cireşan-style notation for neural networks, extended from [18]

Notation	Meaning	Example
y x *z*	Input size is of dimensionality *y* x *z*	Input is 48x48 pixels
xCy	Convolutional layer with *x* maps and filters of *y* x *y* weights	8C3
MPy_p	Max pooling layer with *y* x *y* pooling and *p* stride	MP2_2
APy	Average pooling layer of	AP2
ReLu	rectified linear units layer of size *y* x *y*	ReLu
SM	Softmax layer	SM
BN	Batch normalization Layer	BN
DOx	Drop out layer with x nodes	DO25
xN	Fully connected layer of *x* neurons	100N
CL	Classification layer	CN

Notably, Table 2 is not an exhaustive list of all possible neural network architecture parameters, but one to begin a discussion on report-ability standards in ML, and especially when using neural networks. If one were considering a non-neural network, one would adequately describe the general function and input settings, i.e. as if one were going to call the function within a program. For example, this could appear as

$$Algorithm(\alpha = c, \beta = d, \ldots, \omega = z), \qquad (3)$$

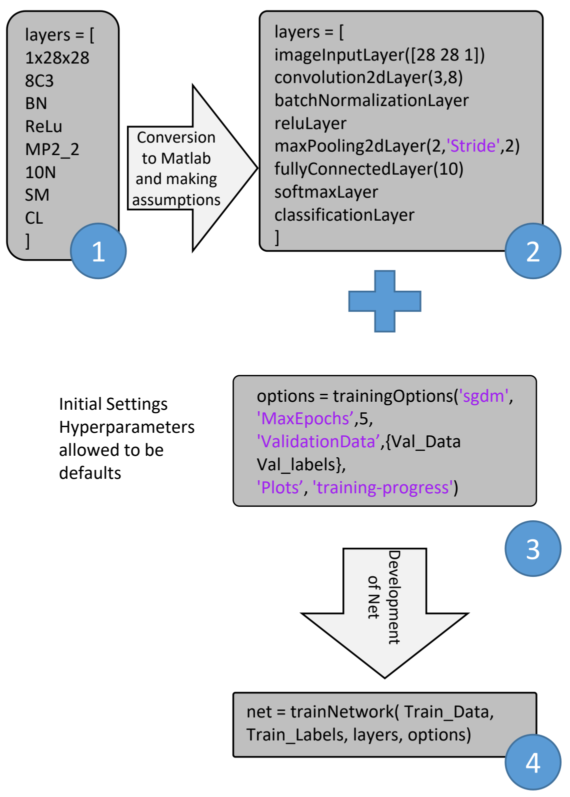
where α, β, and ω are hyperparameter, and c, d, and z are the algorithmic or operational settings (continuous, integer, categorical, etc.) used to achieve stated results.

B1. Train ML Model Using Default Weights involves taking the algorithm from A2 into the programming environment to train and explore results. At this step, the authors recommend using default settings from the functions themselves or example settings from help documentation. The purpose of this is to find baseline results since the next step will be to find reasonable settings. For example, we likely don't know the optimal settings for a given algorithm on a given dataset, but we do know the default, or example, settings in software.

Figure 5 illustrates the B1 process advocated by the authors. For this example, we will consider Matlab (2019a, Mathworks, Natick, MA) with examples of how to quickly convert a simple description of an ML algorithm to a trained model. Here, the notation from A2 is seen in step 1. This is converted to MATLAB notation in step 2, the mapping from equation (2) to step 2 is rather straightforward and logical, e.g. *xCy* is convolutional2dLayer(y,x), and involves being mindful of notation and syntax.

From step 2, one must find default/example settings for typical hyperparameters. While this can involve some investigation, in general, these are inputs to the functions in step 2 and consists of learning rates and other factors. An example of default settings from general help documents is seen in step 3. With this setup, the algorithm is then trained using the selected data in step 4 which results in a fully trained algorithm.

Notably, the setup in Figure 5 ignores the default settings of the MATLAB example in [29]. This is purposeful to illustrate the process and to provide a comparison of this process to the default example results.



Conversion to Matlab and making assumptions
1
2
Initial Settings Hyperparameters allowed to be defaults
3
Development of Net
4

```
layers = [
1x28x28
8C3
BN
ReLu
MP2_2
10N
SM
CL
]
```

```
layers = [
imageInputLayer([28 28 1])
convolution2dLayer(3,8)
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(10)
softmaxLayer
classificationLayer
]
```

```
options = trainingOptions('sgdm',
'MaxEpochs',5,
'ValidationData',{Val_Data
Val_labels},
'Plots', 'training-progress')
```

```
net = trainNetwork( Train_Data,
Train_Labels, layers, options)
```

Figure 5. Step B1: Conceptualization and Matlab code from [29] **illustrate converting the function (2) as a black box for hyperparameter optimization**

B2. Optimize Hyperparameters involves finding reasonable settings via hyperparameter optimization. As discussed in Section 2, BO will be used for this process; however, determining the hyperparameters to optimize is important. Difference between architectural settings and hyperparameters are important to note. Changing architectural settings, e.g. number of nodes in a layer or the number of layers, yields a new method entirely, but changing hyperparameters is merely finding settings that tune the algorithm.

If one is given a general function, e.g. the SVDD function briefly mentioned in A2, one would optimize its two hyperparameters. But, for more complicated examples, e.g. (2), then one must determine what parameters are available. Even in the example of [29], some parameters are left to internal defaults, e.g. the batch size.

In general, for neural networks and deep learning, the following hyperparameters are ones to consider which do not involve specifying a new architecture. These hyperparameters, are roughly described as in Table 3. One further challenge exists when optimizing via BO, or most hyperparameter optimization methods. This is where one must specify upper and lower bounds for each hyperparameter. Here, the authors adopt a process of using wide intervals because the goal herein is to achieve suitable (or acceptable) performance results when given an architecture. For this, the authors adopt the intervals in the last column of Table 3 where N_C is the number of classes.

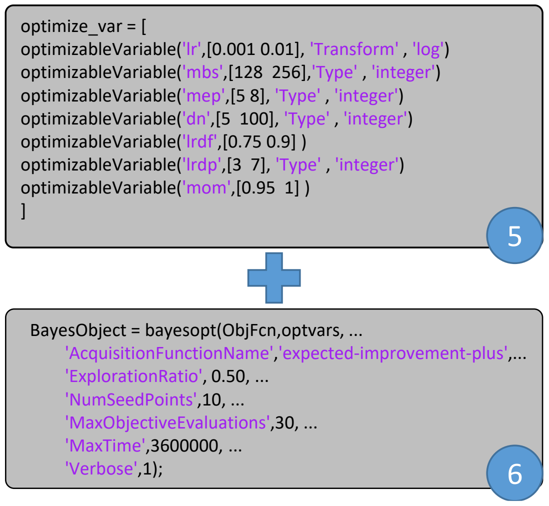
Table 3. General Hyperaparameters for Deep Learning, with Initial Search Region

Param.	Meaning	Initial Search Interval
lr	Learning Rate - update speed at each training step	[0.0001, 0.01]
mep	Number of Epochs - An epoch is one time through the entire training data	[5, 8]
lrdf	Learn Rate Drop Factor - Percentage of the Learn Rate to retain after a specified period	[0.75, 0.9]
lrdp	Learn Rate Drop Period - The epoch at which the Learn Rate Drop Factor is employed	[3, 7]
mom	Momentum - the carryover of the Learning Rate from one epoch to the next	[0.95, 1.0]
mbs	The Batch Size - the number of training samples to consider at one time	[128, 256]
dn	Number of Dense Nodes - The size of the fully connected classifier layer	[1/2•NC, 10•NC]

With the bounds from Table 3, the authors have setup a similar process in Figure 6 to optimize the algorithm from Figure 5. Here, the optimization variables are setup in step 5 and the function in (2) is treated as the objective function in Step 6, along with the optimization variables and some basic settings.

C. Test & Compare Optimized Model involves assessing performance of the model from both B2 and B1. Once a baseline is found in step B1 and the model is optimized in step B2, one must use effective performance measures to evaluate results. Here one is interested in various aspects: overall accuracy on the sequestered test set, training and test set accuracy, accuracy by class, etc. Various discussions exist on this matter, e.g. [30].

One critical aspect of this assessment is considering multiple replications of the same model on the same data. Since ML algorithms are typically stochastic, random variation in the results exists since randomness exists at almost all steps. Thus, running the algorithm multiple times and reporting the average accuracy and the confidence interval from the results is important. For all examples herein, the authors will consider reporting test set accuracy with the mean and a 95% confidence intervals from 10 replications.



```
optimize_var = [
optimizableVariable('lr',[0.001 0.01], 'Transform' , 'log')
optimizableVariable('mbs',[128  256],'Type' , 'integer')
optimizableVariable('mep',[5 8], 'Type' , 'integer')
optimizableVariable('dn',[5  100], 'Type' , 'integer')
optimizableVariable('lrdf',[0.75 0.9] )
optimizableVariable('lrdp',[3  7], 'Type' , 'integer')
optimizableVariable('mom',[0.95  1] )
]
```

```
BayesObject = bayesopt(ObjFcn,optvars, ...
      'AcquisitionFunctionName','expected-improvement-plus',...
      'ExplorationRatio', 0.50, ...
      'NumSeedPoints',10, ...
      'MaxObjectiveEvaluations',30, ...
      'MaxTime',3600000, ...
      'Verbose',1);
```

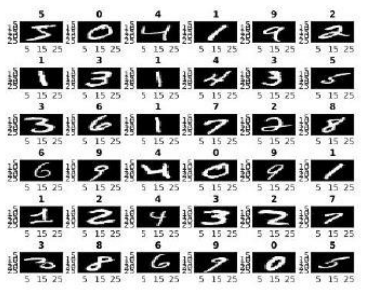
Figure 6. Step B2: Example and conceptualization of hyperparameter optimization for Equation (2) using the net from Figure 4 and the Bayesian Optimization (BO) approach within MATLAB

4. Example Application Results

For an illustration of example application, the authors considered three common benchmarking datasets, as presented in Figure 7: MNIST [19] in Figure 7a, Fashion-MNIST [31] in Figure 7b, and CIFAR-10 [32] in Figure 7c. Along with these datasets, the authors consider one representative algorithm for each dataset.

4.1. Example Datasets

MNIST is a collection of handwritten digits (0, 1, …, 9) in grayscale. Each digit is size-normalized to 20-by-20 pixels, and centered within a 28-by-28 pixel


a)

b)

c)

Figure 7. Example Datasets: a) the first 36 observations from MNIST, b) a randomly selected set of 25 examples from Fashion-MNSIT, c) representative sample of CIFAR-10

box [19]. Figure 7a presents a representative example of MNIST data by visualizing 36 digits. MNIST is composed of 70,000 observations with a predefined n_{TST} = 10,000 observations sequestered for testing.

Fashion-MNIST [31] is conceptually similar to MNIST in being a large grayscale dataset, with images of comparable size, while being harder to accurately classify. For this task, the originators took pictures of clothes from a sales website and downsampled to 28x28 grayscale images [31]. The result is a dataset of 70,000 fashion products, equally distributed into 10 categories (T-shirt, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag, and ankle boots) [31]. Fashion-MNIST is also similarly pre-divided into n_{TNG} = 60,000 for training and n_{TST} = 10,000 [31].

CIFAR-10 (Figure 7c) [32] is a set 60,000 color images (32x32x3), equally distributed into 10 categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), and grouped into n_{TNG} = 50,000 images and n_{TST} = 10,000 images [32]. This dataset is more difficult than MNIST and Fashion-MNIST since each image is more complex, with color and rich background details.

4.2. Algorithms and Results

To illustrate the applicability of the process in Section 3, one algorithm is considered for each problem. First, the example algorithm from the example of [29], equation (2), is presented to illustrate the overall process. This is beneficial since we have access to suitably performing settings from Mathworks. Next, the authors consider the seminal LeNet-4 algorithm of [19], one of the first Deep Learning algorithms that was applied to MNIST. Finally, the authors apply the methodology to a CIFAR-10 example.

4.2.1 MATLAB Example for Fashion-MNIST. This example, equation (2), was built around Fashion-MNIST as an example of MATLAB deep learning capabilities. The framework and initial settings for using the authors' process for this algorithm are discussed in Section 3. Of interest, to illustrate the process from Section 3, is considering four sets of results: 1) baseline results on the test set from step B1, 2) BO results from step B2, 3) model training accuracy results, showing BO progression, and 4) results using the prescribed example settings (lr = 0.001, lrdf = 0.40, lrdp = 9).

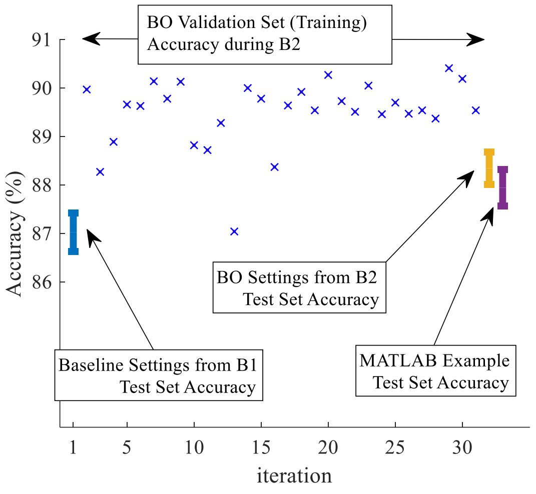


Figure 8. Accuracy of Deep Learning Algorithm in Equation (2) on the Fashion MNIST Data

Figure 8 presents these accuracy results on Fashion-MNIST. Notably, this figure has both test set results and model building results and some care is thus needed in reading results. The first set of results are a 95% t-test confidence interval on test set results using default settings (per step B1). This is centered at 87.03% and is located at iteration 1. The results from using BO are seen at the right and centered at 88.35%. Next to these results are the baseline performance from the MATLAB settings, centered at 87.94%. Notably the B2 optimized results and the MATLAB results are similar.

In Figure 8, between the baseline results and the B2 and MATLAB results we present an illustration of the

BO training set results that yielded the final model. These are "x's" since these are individual accuracy scores on the training set. Notably these results are overall higher in accuracy than the test set, this is to be expected since the test set is unknown to the models.

4.2.2 Further Results: MNIST and CIFAR-10

LeNet-4 is considered because its development largely started the Deep Learning domain in 1995. While Deep Learning methods have advanced significantly since the introduction of LeNet, they are of interest in reproducing for providing a baseline and continued research. Using Table 2, LeNet-4 can be represented as:

32x32-4C5-AP2-16C5-AP2-120N-10N-SM-CL (5)

Notably, the accuracy on MNIST test set is reported as 98.9% [19]. But neither are the settings that produced this know, or reported in [19], nor is the interval on the average performance known. The authors note that reporting such details was not common practice at that time, see [8].

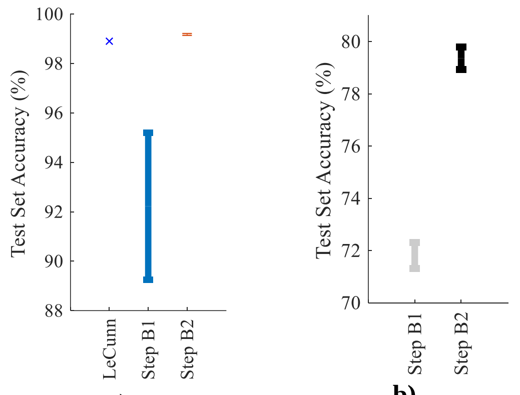


Figure 9. Test Set Accuracy of a) LeNet-4 on MNIST and b) Keras Example on CIFAR-10

Figure 9a presents the results of the authors' process with the results of [19] noted as LeCun. The initial results from Step B1 show a wide 95% t-test confidence interval (standard error of 0.013) which does not reach the published results. However, after 30 iterations of Step B2, the authors reach a very stable solution at 99.17% (standard error of $1.7 \cdot 10^{-4}$). Thus, without this process, one might be suspect of the posted results; but, with the process of Section 3 we have both validated, and exceeded, the results of [19] while showing the importance of hyperparameters.

As a further example, the authors consider the examples found within Keras, an open source Python neural network library, specifically for CIFAR-10 [33]. Using notion from Table 2, we represented it as:

32x32x3-32C3-32C3-MP2-DO25-64C3-64C3-MP2-DO25-576N-640N-DO50-10N (6)

Notably, the originators reported that 75% test set accuracy is possible after 25 epochs, and 79% after 50 epochs [33]. When using the process from Section 3, again ignoring the provided hyperparameter settings, the authors realized the results in Figure 9b, which are similar to the posted results.

5. Conclusions

The authors presented a systematic approach to developing AI/ML models when given an algorithm and data. This contribution addresses recent concerns in AI/ML literature involving the "ilities", e.g. explainability, repeatability, and usability, of AI/ML, with an aim of making AI/ML algorithm development more scientific. To this aim, we explored hyperparameter optimization methods and introduced a short taxonomy of methods. With an understanding of the problem and possible mathematical solutions, the authors presented a straightforward framework, with example code, which 1) provides the ability to automatically find algorithm hyperparameters, 2) enables one to verify the posted results of others' algorithms, and 3) provides reasonable results when getting started with complex algorithms. Furthermore, the authors illustrated this approach on three different ML algorithms on three image recognition datasets.

6. References

[1] N. Kühl, M. Goutier, R. Hirt and G. Satzger, "Machine Learning in Artificial Intelligence: Towards a Common Understanding," *Hawaii International Conference on System Sciences.*, pp. 5236-5245, 2019.

[2] R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification*, 2nd ed., John Wiley & Sons, 2001.

[3] C. Drummond, "Replicability is not reproducibility: nor is it good science.," *International Conference on Machine Learning (ICML)*, 2009.

[4] D. Mishkin and J. Matas, "All you need is a good init.," *arXiv preprint* arXiv:1511.06422., 2015.

[5] J. Snoek, H. Larochelle and R. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, pp. 2951-2959, 2012.

[6] M. J. Mendenhall, *A Neural Relevance Model for Feature Extraction from Hyperspectral Images, and its Application in the Wavelet Domain*, PhD Dissertation: Rice University, 2006.

[7] D. Desai and J. Kroll, "Trust but verify: A guide to algorithms and the law.," *Georgia Tech Scheller College of Business Research Paper No. 17-19*, 2017.

[8] G. Zhang, "Avoiding pitfalls in neural network research," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 1, pp. 3-16, 2007.

[9] R. Potember, *Perspectives on Research in Artificial Intelligence and Artificial General Intelligence Relevant to DoD*, McLean, VA: JASON, The MITRE Corporation, 2017.

[10] A. Shadowen, *Ethics and Bias in Machine Learning: A Technical Study of What Makes Us "Good"*, MS Thesis: CUNY John Jay College of Criminal Justice, 2017.

[11] D. Gunning, *Explainable artificial intelligence (xai)*, Defense Advanced Research Projects Agency (DARPA), 2017.

[12] A. Abran, A. Khelifi, W. Suryn and A. Seffah, "Consolidating the ISO usability models," *International Software Quality Management Conference*, pp. 23-25, 2003.

[13] D. D. Woods, "Decomposing automation: Apparent simplicity, real complexity," in *Automation and human performance: Theory and applications*, 1996, pp. 3-17.

[14] M. Shaw, "Prospects for an Engineering Discipline of Software," *IEEE Software*, vol. 7, no. 6, pp. 15-24, 1990.

[15] C. Shearer, "The CRISP-DM model: the new blueprint for data mining," *Journal of data warehousing*, vol. 5, no. 4, pp. 13-22, 2000.

[16] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter and K. Leyton-Brown, "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 826-830, 2017.

[17] D. Ciresan, A. Giusti, L. Gambardella and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," *Advances in neural information processing systems*, pp. 2843-2851, 2012.

[18] D. Cireşan, U. Meier and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint* arXiv:1202.2745, 2012.

[19] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard and V. Vapnik, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural networks: the statistical mechanics perspective*, 1995.

[20] P. Lorenzo, J. Nalepa, M. Kawulok, L. Ramos and J. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 481-488, 2017.

[21] T. Bihl and D. Steeneck, "Multivariate Stochastic Approximation to Tune Neural Network Hyperparameters for Criticial Infrastructure Communication Device Identification," *Hawaii International Conference on System Sciences (HICSS)*, pp. 2225-2234, 2018.

[22] G. Diaz, A. Fokoue-Nkoutche, G. Nannicini and H. Samulowitz, "An effective algorithm for hyperparameter optimization of neural networks," *IBM Journal of Research and Development*, vol. 61, no. 4/5, 2017.

[23] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765-6816., 2017.

[24] Y. Ozaki, M. Yano and M. Onishi, "Effective hyperparameter optimization using Nelder-Mead method in deep learning," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, 2017.

[25] E. Hazan, A. Klivans and Y. Yuan, "Hyperparameter optimization: A spectral approach.," *arXiv preprint* arXiv:1706.00764., 2017.

[26] R. Martinez-Cantin, "Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3735-3739, 2014.

[27] J. Bergstra, D. Yamins and D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," *Proceedings of the 12th Python in science conference*, pp. 13-20, 2013.

[28] B. Boehmke, *Data Wrangling with R*, Springer International Publishing, 2016.

[29] Mathworks, "Classify Fashion Items with a Convolutional Neural Network," Mathworks Help, 2019.

[30] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 867-874, 2006.

[31] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint* arXiv:1708.07747., 2017.

[32] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, Toronto, CA: University of Toronto, 2009.

[33] Keras, "Train a simple deep CNN on the CIFAR10 small images dataset.," *Keras Documentation*, [Online]. Available: https://keras.io/examples/cifar10_cnn/. [Accessed 10 May 2019].