Feature Discovery with Deep Learning Algebra Networks

Michael F. Korns
Korns Associates
San Juan, Puerto Rico 00911

ABSTRACT

Deep learning neural networks have produced some notable well publicized successes in several fields. Genetic Programming has also produced well publicized notable successes. Inspired by the deep learning successes with neural nets, we experiment with deep learning algebra networks where the network remains unchanged but where the neurons are replaced with general algebraic expressions. The training algorithms replace back propagation, counter propagation, etc. with a combination of genetic programming to generate the algebraic expressions and multiple regression, logit regression, and discriminant analysis to train the deep learning algebra network. These enhanced algebra networks are trained on ten theoretical classification problems with good performance advances which show a clear statistical performance improvement as network architecture is expanded.

Keywords

Symbolic Regression, Symbolic Classification, Genetic Programming, Deep Learning Algebra Networks.

1. Introduction

Deep learning neural networks have produced some notable successes in several fields [1] [2] [3] [4] [5]. Inspired by the deep learning successes with neural nets, we extend our Abstract Regression Classification (ARC) system to evolve deep learning networks of algebraic expressions. These deep learning algebra networks are such that the network is unchanged but the *neurons* are replaced with general algebraic expressions. The new enhanced system is used to train algebra networks on ten theoretical classification problems with good performance advances. The performance advances are analyzed as the network architecture is expanded both by network depth (*i.e. number of hidden layers in the network*) and by the width of each network layer (*i.e. number of neurons in a layer*). Additionally, the algebra networks will be analyzed from the vantage point of feature discovery, with the layer width being viewed as multiple attempts at discovering the same feature, and the network depth being viewed as attempts to discover multiple new features.

The problems we are attempting to solve herein are described by the simple matrix equation in (E0) where Y is a numeric vector of N elements and X is a numeric matrix of N rows and M columns, $\mathbf{H_y}$ is an optimized function on X and *error* is the term to be minimized. A perfect score would be where *error* = 0.

- (*E0*) $Y = \mathbf{H_y}(X) + error$

In this paper we will restrict our research to basic feed forward deep learning neural networks with multiple hidden layers and a numeric output layer with a single neuron. For basic feed forward deep learning neural nets, each deep learning neural network is composed of an input layer *(with multiple inputs*), multiple hidden layers (*each with multiple self-similar neurons*), and a final output layer (*with one or more neurons*). The input layer is a collection of simple numeric values, while each of the hidden layers and the output layer are a collection of *simulated* neurons. Examining the first hidden layer of a simple feed forward neural network we discover a collection of activation function capped weighted sums of the inputs (*which form the simulated "neurons" in the hidden layer*). Each hidden layer of the network contains many of these simulated neurons [1]. If we arbitrarily choose the hyper tangent function, *for our activation function*, each of our first layer neurons can be expressed as a formula like the following.

- (*E1*) $H_{1j} = \tanh(c_{1j0} + c_{1j1}X_1 + c_{1j2}X_2 + \ldots + c_{1jM}X_M)$

The X_1 thru X_M represent the numeric input values *from the input layer*. The c_{1j0} thru c_{1jM} represent the weights. The term H_{1j} represents the value of the jth neuron in the first layer. There are J such weighted sums, *simulated neurons*, in the first hidden layer. As we can see, each hidden layer of the neural network contains many neurons and even more weights.

Examining the second layer of the neural network we encounter another collection of weighted sums, *with inputs from the first layer neurons*, which form the "*neurons*" in the second layer.

- (*E2*) $H_{2k} = \tanh(c_{2k0} + c_{2k1}H_{11} + c_{2kj}H_{1j} + \ldots + c_{2kJ}H_{1J})$

The H_{11} thru H_{1J} represent the J neurons in the first layer *which are inputs to the second layer neurons*. The c_{2k0} thru c_{2KJ} represent the weights. The term H_{2k} represents the value of the kth neuron in the second layer. There are K such weighted sums, *simulated neurons*, in the second hidden layer. This progression continues neuron by neuron, layer by layer until the final output *neuron* which is also a weighted sum, like the following formula.

- (*E3*) $H_y = \tanh(c_{O0} + c_{O1}H_{N1} + c_{Oq}H_{Nq} + \ldots + c_{OQ}H_{NQ})$

The H_{N1} thru H_{NQ} represent the Q neurons in the output layer *which are inputs to the final hidden layer neurons*. The c_{O0} thru c_{OQ} represent the weights. The term H_{Nq} represents the value of the qth neuron in the output layer. There are Q such weighted sums, *simulated neurons*, in the output layer and the output function, $\mathbf{H_y}$, produces one numeric output value from the output layer. Deep learning neural networks often have a large number of neurons in each layer and many layers – often tens and even hundreds of layers or more. Obviously, as the number of neurons and layers grows, we can easily have an explosively large number of formulas such as (E1), (E2), and (E3) with thousands of weights.

In this paper we view the network from a *feature discovery* vantage point, let us describe the set of all input values and all neurons in the network as follows.

- (*E4*) $S = \{X_1, X_2, ..., X_M, H_{11}, ..., H_{1J}, H_{21}, ..., H_{1N}, ..., H_{O1}, ... H_{OQ}, H_y, Y\}$

Expressed as in formula (E4), S is the set of all input features and all hidden layer neurons including the output layer neuron H_y, and the target variable Y. S may be quite a large set, and it is comprehensive. From S we can derive all of the numeric values of the inputs and each hidden layer neuron, plus the values from output neuron. In a batch supervised learning context, the numeric values (*from the inputs and the neurons*) form an array of rows and columns with each row being a training point and each column being the values of the inputs, simulated neurons, and the output. During training, all neurons in the network have their coefficient weights altered to optimize equation (E0).

From S we can infer the dependency properties of the network (*even though S does not contain information about the physical layout of the network*). For instance, a simple dependency graph of the inputs to every neuron formula will identify the layers in S (*any two neurons are in the same layer if they have identical dependency sets*). From the dependency graph we can tell if the network is acyclic (*such as a feed forward network*) or if the network contains cycles (*i.e. a feedback or other more complex network*). Whether there is a single output or multiple outputs can also be determined from the collection of neuron formulas. Once we have the set, S, we no longer need the physical network to train and/or compute the network output – especially if we are focused on analyzing the network from a *feature discovery* vantage point. In this paper we will be adapting all *feed forward*, acyclic, *single output*, neural networks as similarly structured deep learning algebra networks.

Furthermore, examining the set, S, we can view the neural net *hidden layers* as a form of *new feature discovery*. Each hidden layer neuron is another new feature, *added to the list of original input features*, and available as inputs to selected other up layer neurons based upon the physical network topology. The multiple neuron formulas in any given layer are analogous to *repeated attempts* to find relevant new features from the inputs available to the specified layer (*remember all neurons in a layer have identical dependency sets*). Therefore, they represent repeated attempts to discover new features *from the same input data elements but with different randomized learning parameters*.

Examining the simulated neurons in formulas (E1), (E2), and (E3) we see that they are quite specialized and restricted. A great deal of research has gone into enhancing the basic neuron formula to make it less restrictive, while still keeping the claim to *biological inspiration [1]*. It is inevitable that we may wonder, "*What would result from making these simple simulated neuron formulas more general*?". For instance, we might want to create a recurrent neural network where the neurons have temporal state *[1]*. There are many reasons to generalize the simulated neurons in a deep learning neural network. Assuming one is willing to relinquish the *biologically inspired* claim, the most obvious way to generalize these simulated neuron formulas is to make them activation function capped algebraic general linear models [6] like the following formula.

- (*E5*) $H_{n+1p} = F_{n0}(c_{n0} + c_{n1}F_{n1}(S) + c_{np}F_{np}(S) + \ldots + c_{nP}F_{nP}(S))$

This formula expresses the generalized linear formula for the pth generalized neuron in the $n+1^{st}$ hidden layer. If F_{n0} represents the hyper tangent function and each F_{np} represents the pth projection function, then H_{n+1p} is our *original simple simulated neuron*. However, with proper function substitutions, H_{n+1p} can be any algebraic formula we wish – an *algebraic neuron*. For instance, the following *algebraic neuron* is just one of a countably infinite number of formulas which can be used to represent our new *algebraic* neurons.

- (*E6*) $H_{n+1p} = \mathbf{sin}(c_{n0} + c_{n1}\mathbf{cos}(H_{21}/X_1) + c_{np}\mathbf{square}(\mathbf{if}(H_{81}<H_{48},H_{29},X_{22})) + \ldots + c_{nP}\mathbf{log}(H_{67}))$

As one can easily see, the two equations are very similar with (E3), *the simulated neuron*, being a specific restricted case of the more general *algebraic neuron* (E5). Algebra neurons form a general class of neural expressions of which the more restricted simulated neuron is a subset. We can also retain or drop the activation function with algebra neurons without loss of generality as follows.

- (*E7*) $H_{n+1p} = c_{n0} + c_{n1}F_{n1}(S) + c_{np}F_{np}(S) + \ldots + c_{nP}F_{nP}(S)$

While the restricted neural net expressions (E3) are biologically inspired, they are almost always quite verbose. This verbose property is largely responsible for the neural net's reputation as a *Black Box* learning methodology. For instance, if the actual correct answer to a hypothetical regression problem is a simple sine wave.

- (*E8*) $y = c_0 + \sin(X_{21})$

It will take around ten or more nested basic neuron expressions of format (E3) to simulate this simple sine wave AND it will be unclear to most human readers *what the multiple nested restricted neurons are trying to accomplish*. Whereas the more general algebraic neuron solution is literally the expression (E8). It is this terse quality which allows networks of algebraic neurons to be a more human readable *White Box*

learning methodology. On average, it can often take approximately one hundred to one thousand simple neurons to simulate a single modestly complex algebraic neuron.

Each Abstract Regression Classification (ARC) network *algebraic neuron* has one output signal which may be input multiple times to the algebraic neurons in the layers above. Such algebra neurons can drop the activation function as the remaining generalized linear model will be no less general [6], or they can keep the activation function. A simple ARC network might appear as follows.

- (*E9*) inputs = $x_1, x_2, \ldots, x_M$
- (*E10*) hidden neuron: $h_1 = c_{10} + c_{11*}(x_1) + c_{12*}(\cos(x_{21})) + \ldots + c_{1M*}(x_3/x_6)$

Neural networks can learn in an unsupervised or a supervised setting. In this paper we are concerned only with batch supervised learning. Neural network supervised training is performed by a selection of learning algorithms which can be applied "*batch*" or "*online*" (*i.e. back-propagation, counter-propagation, or RProp to name few[1]*). Most of the popular neural net training algorithms incrementally modify the entire collection of weights in, $\mathbf{H_y}$, *trickling down incrementally*, layer by layer so as to optimize the final output $\mathbf{H_y}$.

ARC Network training proceeds bottom up, layer by layer, where each neuron, $\mathbf{H_{oq}}$, is successively optimized against Y. Training ARC networks is predicated on the fact that algebraic neurons are delivered in the format of general linear models (GLM) [6]. General linear models are amenable to four types of machine learning which we use extensively in ARC. These are (a) *genetic programming* [7] for evolving *concrete* algebraic neuron formulas to be optimized, (b) *multiple regression* [11] for optimizing algebraic neuron formulas with numeric outputs, (c) *logistic regression* [12] for optimizing algebraic neuron formulas with binary outputs, and (c) *linear discriminant analysis* [8][9] for optimizing algebraic neuron formulas with m-class outputs. The ARC learning algorithm employs industrialized versions of these four learning algorithms, as described in [10], and proceeds bottom up, one *algebraic neuron* expression at a time, adjusting that neuron expression to optimize equation (E0), then proceeding up through the dependency network hierarchy until the final output expression is optimized, $\mathbf{H_y}$. Interestingly, one could define an entire ARC network of restricted *simulated neuron* algebra expressions and train it in this bottom-up approach. We will not explore this avenue here; but it would offer an experimental mechanism for comparing the four bottom-up ARC training algorithms versus the several popular trickle-down neural network training algorithms.

Neural network architecture determines the total number of neurons in a neural network. Neurons such as (E1), (E2), and (E3) are fixed unchanging *concrete* formulas wherein only the coefficient weights change during training. One can examine a neural network architecture in advance and compute the maximum number of neurons that will ever be optimized. Conversely, the action of genetic programming makes each ARC network an *abstract* network rather than a fixed *concrete* network. Algebra neurons like (E5) and (E7), *together with genetic programming technology*, can best be thought of as neuron *factories* which act upon abstract neurons such as (E5) and produce multiple concrete neurons such as (E6). ARC networks are abstract in nature - composed of neuron factories rather than concrete neurons. During ARC network training, each abstract neuron factory produces hundreds and thousands of concrete neurons whose coefficients are then optimized against Y. So not only is each algebra neuron, *on average*, far more complex than each neural network neuron, but many hundreds and thousands of concrete algebra neurons are produced and optimized during ARC network training for every single abstract neuron in the ARC network architecture. Even a small ARC network can contain tens of thousands of concrete optimized algebra neurons. Larger ARC networks can contain millions of concrete optimized algebra neurons.

Eliminating poor performing neurons (*pruning*) has been shown to be a vital technique for enhancing neural net training [13]. We have found that pruning of algebra neurons also enhances ARC network training. In fact, *given the large number of algebra neurons produced in training each ARC network*, pruning of poor performing algebra neurons is essential to increase learning efficiency and to reduce bloat. ARC algebraic neurons can be pruned based upon any or more of the following: *coefficient strength*, *principal component analysis*, and *fitness competition.* In this paper, all three pruning methodologies are employed to the highest levels possible.

Of course, the general nature of the algebraic neuron places the claim of *biological inspiration* in jeopardy. Arguments have been made that certain enhancements of the simple neuron are biologically inspired. For instance, some scientists have argued that recurrent neurons are biologically inspired. However, few scientists would argue that the more general *algebraic neuron* is biologically inspired. In the general algebraic case, the term *neuron* is more of a legacy nomenclature than a claim to neuron-similar behavior. So, if *algebraic neurons* are not biological inspired, can networks of algebraic neurons be trained and can they be made useful in any practical sense?

This paper includes a performance comparison of deep learning algebra networks and five well-known commercially available classification algorithms on ten theoretical noiseless classification problems. The five commercially available classification algorithms are found in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL). We show that, on the theoretical problems, the two best classification algorithms are Gradient Boosted Decision Trees (GBTL) and this paper's deep learning algebra networks (ARC). Furthermore, we show that the performance across all ten theoretical problems consistently improves as the algebra network is expanded both in width (*neurons per layer*) and depth (*number of layers*).

2. ARC Background

By way of providing some background, our **A**bstract **R**egression **C**lassification (ARC) system has been under research and development since 2004 [10][15][16][17][18][19][20][21]. ARC has been heavily industrialized and requires no *genetic programming specific* input parameters. Only the names of the training and testing data files and the nature of the target variable (*numeric*, *binary*, or *nary*) need be specified. The selection of the fitness method, running of multiple genetic programming runs with different random number seeds, splicing the different runs together to form a layered network, determining when the system is finished learning, etc., all of these tasks are hidden from the user by the ARC system planning module. Only a single ARC training run is needed per problem, and the ARC system is guaranteed to converge on the best solution it can find in the finite time and computation resources allotted.

The ARC planning module is based around the **R**egression **Q**uery **L**anguage (RQL) which is an SQL inspired search language for specifying genetic programming symbolic regression and classifications runs. The RQL language is briefly described in [15] and can be used to set in motion single island or multiple island genetic programming runs with *aged-layered*, *pareto*, *elitist*, and many other GP methodologies. The RQL language employs industrialized implementations of (a) *genetic programming [2][7]* for evolving algebraic neuron formulas to be optimized, (b) *multiple regression [11]* for optimizing algebraic neuron formulas with numeric outputs, (c) *logistic regression [12]* for optimizing algebraic neuron formulas with binary outputs, and (c) *linear discriminant analysis [8][9]* for optimizing algebraic neuron formulas with m-class outputs. The RQL language is quite sophisticated. For instance, one study includes an RQL specification which is conjectured to be absolutely accurate on certain scientific problems [15]. The ARC planning module currently contains a library of numerous predefined RQL searches known to be effective for specific problems. The planning module applies its library of known RQL searches, *based upon its own heuristic and statistical analysis of the data to be optimized*. Each ARC training run hides thousands of separate genetic programming runs from the user. Human intervention is not required.

3. Regression in Brief

Regression, single and multiple, involves a single dependent variable and one or more independent variables. It is a statistical technique that develops an optimal mathematical relationship between one or more real independent variables and a real dependent variable. Most modern regression tools manage linear regression. Symbolic regression tools attack the mathematical problem of nonlinear regression by employing genetic programming.

The canonical generalization of linear regression into nonlinear regression is the class of Generalized Linear Models (GLMs) as described in [6]. A GLM is a linear combination of **I** algebraic functions B_i; i = 0,1, I, a dependent variable y, and an independent data point with M features $x = \langle x_0, x_1, x_2, , x_{M-1} \rangle$: such that

- (*E11*) $y = \gamma(x) = c_0 + c_1B_1(x) + c_2B_2(x) + \ldots + c_{M-1}B_{M-1}(x) +$ **error**

As a broad generalization, GLMs can represent *any possible nonlinear formula*. However, the format of the GLM makes it amenable to existing linear regression theory and tools since the GLM model is linear on each of the algebraic functions B_i *(although each algebraic function may be nonlinear)*. For a given vector of dependent variables, Y, and a vector of independent data points, X, symbolic regression will search for a set of algebraic functions and coefficients which minimize **error**. In [7] the algebraic functions selected by symbolic regression will be formulas as in the following examples:

- (*E12*) $B_0 = x_3$
- (*E13*) $B_1 = x_1+x_4$
- (*E14*) $B_2 = \text{sqrt}(x_2)/\tan(x_5/4.56)$
- (*E15*) $B_3 = \tanh(\cos(x_2*.2)*\text{cube}(x_5+\text{abs}(x_1)))$

Once a suitable set of algebraic functions B have been selected (*via genetic programming*), we can discover the proper set of coefficients C deterministically using standard simple or multiple regression [11]. The value of the GLM model is that one can use standard regression techniques and theory to optimize for the constants while using genetic programming to search for the optimal algebraic functions.

4. Classification in Brief

For all binary classification, we use Logit Regression (LOG) as in [12]. For all M-Class classification, we use Linear Discriminant Analysis (LDA). Linear discriminant analysis is a generalization of Fischer's linear discriminant, which is a method to find a linear combination of features which best separates K classes of training points [8], [9], [10]. Both LOG and LDA are used extensively in Statistics, Machine Learning, and Pattern Recognition.

In symbolic classification problems, an N by M matrix of independent training points, X, is matched with an N vector of dependent values containing only categorical unordered values between 1 and K. The fitness measure is the *classification error percent* (CEP). Linear

discriminant analysis is employed as the assisted fitness training technique in our ARC system. The CEP is the percent of misclassified testing points (*i.e. the count of misclassifications divided by the number of testing points*).

Our symbolic classification system outputs K predictor functions (*one for each class*). These functions are called discriminants, $D_k(X) \sim Y_k$, and there is one discriminant function for each class. The format of ARC's discriminant function output is always as follows.

- (*E16*) $y = \text{argmax}(D_1,D_2, \dots ,D_K)$

The *argmax* function returns the class index for the largest valued discriminant function. For instance if $D_i = \max(D_1,D_2, \dots ,D_K)$, then $i = \text{argmax}(D_1,D_2, \dots ,D_K)$.

A central aspect of LDA is that each discriminant function is a linear variation of every other discriminant function. For instance, if the ARC symbolic classification system produces a candidate with B algebraic neuron functions, then each discriminant function has the following format.

$$D_0 = \mathbf{c}_{00}+\mathbf{c}_{01}*Bf_1+\mathbf{c}_{02}*Bf_2+\dots+\mathbf{c}_{0B}*Bf_B$$

$$D_1 = \mathbf{c}_{10}+\mathbf{c}_{11}*Bf_1+\mathbf{c}_{12}*Bf_2+\dots+\mathbf{c}_{1B}*Bf_B$$

$$D_k = \mathbf{c}_{k0}+\mathbf{c}_{k1}*Bf_1+\mathbf{c}_{k2}*Bf_2+\dots+\mathbf{c}_{kB}*Bf_B$$

The K*(B+1) coefficients are selected so that the i-th discriminant function has the highest value when the y = i (*i.e. the class is i*). The industrialized LDA technology ARC uses for selecting these optimized coefficients $\mathbf{c}_{00}$ to $\mathbf{c}_{KB}$ is discussed in more detail here [18].

5. Industrial Regression Classification

The single, multiple, and logit regression plus the linear discriminant analysis algorithms in ARC have been industrialized to handle real world problems. These quite exacting deterministic algorithms all suffer from their requirement that certain assumptions about the training data hold true – namely that the data have a normal distribution, that the training matrices be nonsingular, etc. In cases where the data does not strictly conform to these assumptions, these deterministic algorithms can fail or produce inaccurate results. Whenever ARC discovers poorly structured training data, the deterministic regression classification algorithms are forced into approximately accurate coefficients. Next a layer of fast evolutionary algorithms is applied to coerce the approximately accurate coefficients into a more accurate set of coefficients. These evolutionary algorithms include modified sequential minimal optimization and bees swarm optimization [23][24]. These industrial enhancements create regression classification algorithms which are robust even with poorly formed training data.

6. Theoretical Test Problems - Classification

A set of ten artificial classification problems are constructed, with no noise, to test the efficacy of the new ARC deep learning networks. Each of the artificial test problems is created around an **X** training matrix filled with random real numbers in the range [-10.0,+10.0]. The number of rows and columns in each test problem varies from 5000x25 to 5000x1000 depending upon the difficulty of the problem. The number of classes varies from **Y** = {0,1} to **Y** = {0,1,2,3,4} depending upon the difficulty of the problem. The test problems are designed to vary from extremely easy to very difficult. The first test problem is linearly separable with 2 classes on 25 columns. The tenth test problem is nonlinear multimodal with 5 classes on 1000 columns.

Standard statistical best practices out of sample testing are employed. First the training matric X is filled with random real numbers in the range [-10.0,+10.0], and the Y class values are computed from the argmax functions specified below. A champion is trained on the training data. Next a testing matric X is filled with random real numbers in the range [-10.0,+10.0], and the Y class values are computed from the argmax functions specified below.

The argmax functions used to create each of the ten artificial test problems are as follows.

Artificial Test Problems

- **C1**: **y = argmax**(D_1,D_2) where **Y** = {1,2}, **X** is 5000x25, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*x0),(-39.34*x1),(2.13*x2),(46.59*x3),(11.54*x4))
  - D_2=sum((-1.57*x0),(39.34*x1),(-2.13*x2),(-46.59*x3),(-11.54*x4))
- **C2**: **y = argmax**(D_1,D_2) where **Y** = {1,2}, **X** is 5000x100, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((5.16*x0),(-19.83*x1),(19.83*x2),(29.31*x3),(5.29*x4))
  - D_2:=sum((-5.16*x0),(19.83*x1),(-0.93*x2),(-29.31*x3),(5.29*x4))
- **C3**: **y = argmax**(D_1,D_2) where **Y** = {1,2}, **X** is 5000x1000, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((-34.16*x0),(2.19*x1),(-12.73*x2),(5.62*x3),(-16.36*x4))
  - D_2=sum((34.16*x0),(-2.19*x1),(12.73*x2),(-5.62*x3),(16.36*x4))
- **C4**: **y = argmax**(D_1,D_2,D_3) where **Y** = {1,2,3}, **X** is 5000x25, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*cos(x0)),(-39.34*square(x10)),(2.13*(x2/x3)),(46.59*cube(x13)),(-11.54*log(x4)))
  - D_2=sum((-0.56*cos(x0)),(9.34*square(x10)),(5.28*(x2/x3)),(-6.10*cube(x13)),(1.48*log(x4)))

- D_3=sum((1.37*cos(x0)),(3.62*square(x10)),(4.04*(x2/x3)),(1.95*cube(x13)),(9.54*log(x4)))

- **C5**: **y = argmax**(D_1,D_2,D_3) where **Y** = {1,2,3}, **X** is 5000x100, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*sin(x0)),(-39.34*square(x10)),(2.13*(x2*x3)),(46.59*cube(x13)),(-11.54*cube(x4)))
  - D_2=sum((-0.56*sin(x0)),(9.34*square(x10)),(5.28*(x2*x3)),(-6.10*cube(x13)),(1.48*cube(x4)))
  - D_3=sum((1.37*sin(x0)),(3.62*square(x10)),(4.04*(x2*x3)),(1.95*cube(x13)),(9.54*cube(x4)))

- **C6**: **y = argmax**(D_1,D_2,D_3) where **Y** = {1,2,3}, **X** is 5000x1000, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*tanh(x0)),(-39.34*sqroot(x10)),(2.13*(x2*x3)),(46.59*(x13/x23)),(2.54*square(x4)))
  - D_2=sum((-0.56*tanh(x0)),(9.34*sqroot(x10)),(5.28*(x2*x3)),(-6.10*(x13/x23)),(1.48*square(x4)))
  - D_3=sum((1.37*tanh(x0)),(3.62*sqroot(x10)),(4.04*(x2*x3)),(1.95*(x13/x23)),(0.54*square(x4)))

- **C7**: **y = argmax**(D_1,D_2,D_3,D_4,D_5) where **Y** = {1,2,3,4,5}, **X** is 5000x25, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*cos(x0/x21)),(9.34*((square(x10)/x14)*x6)),(2.13*((x2/x3)*log(x8))),(46.59*(cube(x13)*(x9/x2))),(-11.54*log(x4*x10*x15)))
  - D_2=sum((-1.56*cos(x0/x21)),(7.34*((square(x10)/x14)*x6)),(5.28*((x2/x3)*log(x8))),(-6.10*(cube(x13)*(x9/x2))),(1.48*log(x4*x10*x15)))
  - D_3=sum((2.31*cos(x0/x21)),(12.34*((square(x10)/x14)*x6)),(-1.28*((x2/x3)*log(x8))),(0.21*(cube(x13)*(x9/x2))),(2.61*log(x4*x10*x15)))
  - D_4=sum((-0.56*cos(x0/x21)),(8.34*((square(x10)/x14)*x6)),(16.71*((x2/x3)*log(x8))),(-2.93*(cube(x13)*(x9/x2))),(5.228*log(x4*x10*x15)))
  - D_5=sum((1.07*cos(x0/x21)),(-1.62*((square(x10)/x14)*x6)),(-0.04*((x2/x3)*log(x8))),(-0.95*(cube(x13)*(x9/x2))),(0.54*log(x4*x10*x15)))

- **C8**: **y = argmax**(D_1,D_2,D_3,D_4,D_5) where **Y** = {1,2,3,4,5}, **X** is 5000x100, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*sin(x0/x11)),(9.34*((square(x12)/x4)*x46)),(2.13*((x21/x3)*log(x18))),(46.59*(cube(x3)*(x9/x2))),(-11.54*log(x14*x10*x15)))
  - D_2=sum((-1.56*sin(x0/x11)),(7.34*((square(x12)/x4)*x46)),(5.28*((x21/x3)*log(x18))),(-6.10*(cube(x3)*(x9/x2))),(1.48*log(x14*x10*x15)))
  - D_3=sum((2.31*sin(x0/x11)),(12.34*((square(x12)/x4)*x46)),(-1.28*((x21/x3)*log(x18))),(0.21*(cube(x3)*(x9/x2))),(2.61*log(x14*x10*x15)))
  - D_4=sum((-0.56*sin(x0/x11)),(8.34*((square(x12)/x4)*x46)),(16.71*((x21/x3)*log(x18))),(-2.93*(cube(x3)*(x9/x2))),(5.228*log(x14*x10*x15)))
  - D_5=sum((1.07*sin(x0/x11)),(-1.62*((square(x12)/x4)*x46)),(-0.04*((x21/x3)*log(x18))),(-0.95*(cube(x3)*(x9/x2))),(0.54*log(x14*x10*x15)))

- **C9**: **y = argmax**(D_1,D_2,D_3,D_4,D_5) where **Y** = {1,2,3,4,5}, **X** is 5000x1000, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*sin(x20*x11)),(9.34*(tanh(x12/x4)*x46)),(2.13*((x321-x3)*tan(x18))),(46.59*(square(x3)/(x49*x672))),(-11.54*log(x24*x120*x925)))
  - D_2=sum((-1.56*sin(x20*x11)),(7.34*(tanh(x12/x4)*x46)),(5.28*((x321-x3)*tan(x18))),(-6.10*(square(x3)/(x49*x672))),(1.48*log(x24*x120*x925)))
  - D_3=sum((2.31*sin(x20*x11)),(12.34*(tanh(x12/x4)*x46)),(-1.28*((x321-x3)*tan(x18))),(0.21*(square(x3)/(x49*x672))),(2.61*log(x24*x120*x925)))
  - D_4=sum((-0.56*sin(x20*x11)),(8.34*(tanh(x12/x4)*x46)),(16.71*((x321-x3)*tan(x18))),(-2.93*(square(x3)/(x49*x672))),(5.228*log(x24*x120*x925)))
  - D_5=sum((1.07*sin(x20*x11)),(-1.62*(tanh(x12/x4)*x46)),(-0.04*((x321-x3)*tan(x18))),(-0.95*(square(x3)/(x49*x672))),(0.54*log(x24*x120*x925)))

- **C10**: **y = argmax**(D_1,D_2,D_3,D_4,D_5) where **Y** = {1,2,3,4,5}, **X** is 5000x1000, and each $\mathbf{D_i}$ is as follows
  - D_1=sum((1.57*lif(x0<x23,sin(x20*x11),cos(x10))),(9.34*(tanh(x12/x4)*x46)),(2.13*lif(x10<0.0,((x321-x3)*tan(x18)),((x156-x31)/tanh(x21)))),(46.59*(square(x3)/(x49*x672))),(-11.54*log(x24*x120*x925)))
  - D_2=sum((-1.56*lif(x0<x23,sin(x20*x11),cos(x10))),(7.34*(tanh(x12/x4)*x46)),(5.28*lif(x10<0.0,((x321-x3)*tan(x18)),((x156-x31)/tanh(x21)))),(-6.10*(square(x3)/(x49*x672))),(1.48*log(x24*x120*x925)))
  - D_3=sum((2.31*lif(x0<x23,sin(x20*x11),cos(x10))),(12.34*(tanh(x12/x4)*x46)),(-1.28*lif(x10<0.0,((x321-x3)*tan(x18)),((x156-x31)/tanh(x21)))),(0.21*(square(x3)/(x49*x672))),(2.61*log(x24*x120*x925)))
  - D_4=sum((-0.56*lif(x0<x23,sin(x20*x11),cos(x10))),(8.34*(tanh(x12/x4)*x46)),(16.71*lif(x10<0.0,((x321-x3)*tan(x18)),((x156-x31)/tanh(x21)))),(-2.93*(square(x3)/(x49*x672))),(5.228*log(x24*x120*x925)))
  - D_5=sum((1.07*lif(x0<x23,sin(x20*x11),cos(x10))),(-1.62*(tanh(x12/x4)*x46)),(-0.04*lif(x10<0.0,((x321-x3)*tan(x18)),((x156-x31)/tanh(x21)))),(-0.95*(square(x3)/(x49*x672))),(0.54*log(x24*x120*x925)))

7. Base Performance On The Theoretical Classification Problems

Here we compare the out of sample CEP testing scores of five well-known commercially available classification algorithms to determine where basic 1-layer ARC networks rank in competitive comparison. The five commercially available classification algorithms are available in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL). The following table lists each classification algorithm in descending order of average CEP scores on all ten theoretical test problems. The lower the CEP the more accurate the

classification results. The **ARC^{N1}** network is composed of 1 hidden layer which has 5 algebra neuron factories (*width=5, depth=1 - which is to say almost no network and just one training run with five neuron factories*).

Table 2. Test Problem CEP Testing Results *Before* Deep Learning Enhancements

Test	MLP	DTL	TEL	RFL	ARC^{N1}	GBTL
C1	0.0072	0.0724	0.0496	0.0492	0.0138	0.0308
C2	0.0360	0.0740	0.0648	0.0664	0.0116	0.0240
C3	0.0724	0.0972	0.1526	0.1522	0.0132	0.0332
C4	0.0472	0.0174	0.0252	0.0260	0.0194	0.0170
C5	0.3250	0.0858	0.0946	0.0920	0.0712	0.0530
C6	0.6166	0.5396	0.6284	0.6286	0.5420	0.3198
C7	0.4598	0.2834	0.2284	0.2292	0.2272	0.2356
C8	0.4262	0.2956	0.2248	0.2250	0.2302	0.2340
C9	0.6904	0.6058	0.4334	0.4344	0.4188	0.4286
C10	0.5966	0.5966	0.4352	0.4296	0.4186	0.4286
Avg	0.3277	0.2667	0.2337	0.2332	0.2169	0.1804

Note: All scores quoted in these tables are CEP scores which show the percent of misclassified testing points (the lower the score the better). Only out of sample testing scores are shown. All network architectures featured in these tables specify neuron counts before pruning. The five commercially available classification algorithms are available in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL).

The top performer overall by a very small margin is the Gradient Boosted Trees Learner (**GBTL**). The penultimate performer is the **ARC^{N1}** 1-Layer algebra network. The base ARC^{N1} network is composed of 1 layer with 5 algebra neuron factories producing an average of 51.6K concrete algebra neurons per test case. Interestingly, the base ARC network performs reasonably well *before* deep learning enhancements.

8. Thin 2-Layer ARC Performance On The Theoretical Classification Problems

Here we compare the performance of a thin 2-Layer ARC network with the out of sample CEP testing scores of five well-known commercially available classification algorithms to determine where a 2-layer ARC network ranks in competitive comparison. The following table lists each classification algorithm in descending order of average CEP scores on all ten theoretical test problems. The lower the CEP the more accurate the classification results. The **ARC^{N2}** network is composed of 2 hidden layers each of which has 40 algebra neuron factories for a total of 80 algebra neuron factories in the entire network.

Table 3. Test Problem CEP Testing Results *After* Thin 2-Layer Deep Learning Enhancements

Test	MLP	DTL	TEL	RFL	ARC^{N1}	GBTL	ARC^{N2}
C1	0.0072	0.0724	0.0496	0.0492	0.0138	0.0308	0.0004
C2	0.0360	0.0740	0.0648	0.0664	0.0116	0.0240	0.0004
C3	0.0724	0.0972	0.1526	0.1522	0.0132	0.0332	0.0022
C4	0.0472	0.0174	0.0252	0.0260	0.0194	0.0170	0.0158
C5	0.3250	0.0858	0.0946	0.0920	0.0712	0.0530	0.0490
C6	0.6166	0.5396	0.6284	0.6286	0.5420	0.3198	0.2518
C7	0.4598	0.2834	0.2284	0.2292	0.2272	0.2356	0.2264
C8	0.4262	0.2956	0.2248	0.2250	0.2302	0.2340	0.2238
C9	0.6904	0.6058	0.4334	0.4344	0.4188	0.4286	0.4142
C10	0.5966	0.5966	0.4352	0.4296	0.4186	0.4286	0.4160
Avg	0.3277	0.2667	0.2337	0.2332	0.2169	0.1804	0.1600

Note: All scores quoted in these tables are CEP scores which show the percent of misclassified testing points (the lower the score the better). Only out of sample testing scores are shown. All network architectures featured in these tables specify neuron counts before pruning. The five commercially available classification algorithms are available in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL).

The top performer overall by a reasonable margin is now the **ARC^{N2}** thin 2-Layer algebra network (*width=40, depth=2*). The Gradient Boosted Trees Learner (**GBTL**) has fallen behind. Interestingly, adding just two hidden layers and a total of 80 algebra neuron factories (*40 algebra neuron factories per layer producing an average of 188.9K concrete algebra neurons per test case*) was enough to boost performance beyond that of the Gradient Boosted Trees Learner (**GBTL**).

9. Ultrathin 8-Layer ARC Performance On The Theoretical Classification Problems

Here we compare the performance of an ultrathin 8-Layer ARC network with the out of sample CEP testing scores of five well-known commercially available classification algorithms to determine where an 8-layer ARC network ranks in competitive comparison. The following table lists each classification algorithm in descending order of average CEP scores on all ten theoretical test problems. The lower the CEP the more accurate the classification results. The **ARC^{N3}** network is composed of 8 hidden layers each of which has 10 algebra neuron factories for a total of 80 algebra neuron factories in the entire network (*width=10, depth=8*). On average the **ARC^{N3}** network's 80 neuron factories produced 1.4M concrete neurons per test case.

Table 4. Test Problem CEP Testing Results *After* Ultrathin 8-Layer Deep Learning Enhancements

Test	MLP	DTL	TEL	RFL	ARC^{N1}	GBTL	ARC^{N2}	ARC^{N3}
C1	0.0072	0.0724	0.0496	0.0492	0.0138	0.0308	0.0004	0.0000
C2	0.0360	0.0740	0.0648	0.0664	0.0116	0.0240	0.0004	0.0016
C3	0.0724	0.0972	0.1526	0.1522	0.0132	0.0332	0.0022	0.0026
C4	0.0472	0.0174	0.0252	0.0260	0.0194	0.0170	0.0158	0.0132
C5	0.3250	0.0858	0.0946	0.0920	0.0712	0.0530	0.0490	0.0358
C6	0.6166	0.5396	0.6284	0.6286	0.5420	0.3198	0.2518	0.2392
C7	0.4598	0.2834	0.2284	0.2292	0.2272	0.2356	0.2264	0.2264
C8	0.4262	0.2956	0.2248	0.2250	0.2302	0.2340	0.2238	0.2240
C9	0.6904	0.6058	0.4334	0.4344	0.4188	0.4286	0.4142	0.4162
C10	0.5966	0.5966	0.4352	0.4296	0.4186	0.4286	0.4160	0.4162
Avg	0.3277	0.2667	0.2337	0.2332	0.2169	0.1804	0.1600	0.1575

Note: All scores quoted in these tables are CEP scores which show the percent of misclassified testing points (the lower the score the better). Only out of sample testing scores are shown. All network architectures featured in these tables specify neuron counts before pruning. The five commercially available classification algorithms are available in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL).

The **ARC^{N2}** network and the **ARC^{N3}** network have the same total number of algebra neuron factories. A comparison of their results highlights the difference between more thin layers or fewer wide layers. A win for the **ARC^{N3}** network would indicate that new feature discovery is important for network performance, while a win for the **ARC^{N2}** network would indicate that mere repetition is important for network performance. Notice that the dependency set for each of the layer 1 algebra neurons is the set X. While the dependency set for all of the layer 2 algebra neurons is the set $\{X \cup \{H_1\}\}$ where $\{H_1\}$ is the output of all of the layer 1 algebra neurons after pruning. The difference between the **ARC^{N2}** network and the **ARC^{N3}** network is that the layer 2 thru 8 algebra neurons in the **ARC^{N3}** network have access to the outputs of more previous neurons because the network is deep instead of wide.

The top performer overall by a slight margin is now the **ARC^{N3}** ultrathin 8-Layer algebra network (*even though it has exactly the same total number of algebra neurons as the **ARC^{N2}** network*). Whether this slight advantage persists for other wide network versus thin network architecture and on other test problems will require further experiments.

10. Wide 2-Layer ARC Performance On The Theoretical Classification Problems

Here we compare the performance of a wide 2-Layer ARC network with the out of sample CEP testing scores of five well-known commercially available classification algorithms to determine where a wide 2-layer ARC network ranks in competitive comparison. The following table lists each classification algorithm in descending order of average CEP scores on all ten theoretical test problems. The lower the CEP the more accurate the classification results. The **ARC^{N4}** network is composed of 2 hidden layers each of which has 200 algebra neuron factories for a total of 400 algebra neuron factories in the entire network (*width=200, depth=2*).

Table 5. Test Problem CEP Testing Results *After* Wide 2-Layer Deep Learning Enhancements

Test	MLP	DTL	TEL	RFL	ARC^{N1}	GBTL	ARC^{N2}	ARC^{N3}	ARC^{N4}
C1	0.0072	0.0724	0.0496	0.0492	0.0138	0.0308	0.0004	0.0000	0.0000
C2	0.0360	0.0740	0.0648	0.0664	0.0116	0.0240	0.0004	0.0016	0.0000
C3	0.0724	0.0972	0.1526	0.1522	0.0132	0.0332	0.0022	0.0026	0.0008

C4	0.0472	0.0174	0.0252	0.0260	0.0194	0.0170	0.0158	0.0132	0.0068
C5	0.3250	0.0858	0.0946	0.0920	0.0712	0.0530	0.0490	0.0358	0.0190
C6	0.6166	0.5396	0.6284	0.6286	0.5420	0.3198	0.2518	0.2392	0.2074
C7	0.4598	0.2834	0.2284	0.2292	0.2272	0.2356	0.2264	0.2264	0.2262
C8	0.4262	0.2956	0.2248	0.2250	0.2302	0.2340	0.2238	0.2240	0.2230
C9	0.6904	0.6058	0.4334	0.4344	0.4188	0.4286	0.4142	0.4162	0.4148
C10	0.5966	0.5966	0.4352	0.4296	0.4186	0.4286	0.4160	0.4162	0.4156
Avg	0.3277	0.2667	0.2337	0.2332	0.2169	0.1804	0.1600	0.1575	0.1513

Note: All scores quoted in these tables are CEP scores which show the percent of misclassified testing points (the lower the score the better). Only out of sample testing scores are shown. All network architectures featured in these tables specify neuron counts before pruning. The five commercially available classification algorithms are available in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL).

The **ARC^{N4}** network has 5 times the total algebra neuron factories as the **ARC^{N3}** network although its depth is only 25% of the depth of the **ARC^{N4}** network. A comparison of their results highlights the advantages of a few wide layers over more thin layers. The top performer overall by a slight margin is now the **ARC^{N4}** wide 2-Layer algebra network. The Gradient Boosted Trees Learner (**GBTL**) has fallen further behind. Interestingly, a network of two wide hidden layers, with 200 algebra neuron factories per layer, for a total of 400 algebra neuron factories (*200 algebra neuron factories per layer producing an average of 622.6M concrete algebra neurons per test case*) was enough to boost performance significantly beyond the **ARC^{N3}** thin 8-Layer algebra network. Whether this slight advantage persists for other wide network versus thin network architecture and on other test problems will require further experiments.

11. Wide 8-Layer ARC Performance On The Theoretical Classification Problems

Here we compare the performance of a wide 8-Layer ARC network with the out of sample CEP testing scores of five well-known commercially available classification algorithms to determine where a wide 8-layer ARC network ranks in competitive comparison. The following table lists each classification algorithm in descending order of average CEP scores on all ten theoretical test problems. The lower the CEP the more accurate the classification results. The **ARC^{N5}** network is composed of 8 hidden layers each of which has 200 algebra neuron factories for a total of 1600 algebra neuron factories in the entire network (*width=200, depth=8*).

Table 6. Test Problem CEP Testing Results *After* Wide 8-Layer Deep Learning Enhancements

Test	MLP	DTL	TEL	RFL	ARC^{N1}	GBTL	ARC^{N2}	ARC^{N3}	ARC^{N4}	ARC^{N5}
C1	0.0072	0.0724	0.0496	0.0492	0.0138	0.0308	0.0004	0.0000	0.0000	0.0000
C2	0.0360	0.0740	0.0648	0.0664	0.0116	0.0240	0.0004	0.0016	0.0000	0.0000
C3	0.0724	0.0972	0.1526	0.1522	0.0132	0.0332	0.0022	0.0026	0.0008	0.0000
C4	0.0472	0.0174	0.0252	0.0260	0.0194	0.0170	0.0158	0.0132	0.0068	0.0000
C5	0.3250	0.0858	0.0946	0.0920	0.0712	0.0530	0.0490	0.0358	0.0190	0.0000
C6	0.6166	0.5396	0.6284	0.6286	0.5420	0.3198	0.2518	0.2392	0.2074	0.2056
C7	0.4598	0.2834	0.2284	0.2292	0.2272	0.2356	0.2264	0.2264	0.2262	0.2044
C8	0.4262	0.2956	0.2248	0.2250	0.2302	0.2340	0.2238	0.2240	0.2230	0.2228
C9	0.6904	0.6058	0.4334	0.4344	0.4188	0.4286	0.4142	0.4162	0.4148	0.0178
C10	0.5966	0.5966	0.4352	0.4296	0.4186	0.4286	0.4160	0.4162	0.4156	0.4156
Avg	0.3277	0.2667	0.2337	0.2332	0.2169	0.1804	0.1600	0.1575	0.1513	0.1066

Note: All scores quoted in these tables are CEP scores which show the percent of misclassified testing points (the lower the score the better). Only out of sample testing scores are shown. All network architectures featured in these tables specify neuron counts before pruning. The five commercially available classification algorithms are available in the KNIME system [14], and are as follows: Multiple Layer Perceptron Learner (MLP); Decision Tree Learner (DTL); Random Forest Learner (RFL); Tree Ensemble Learner (TEL); and Gradient Boosted Trees Learner (GBTL).

The top performer overall by a larger margin is now the **ARC^{N5}** wide 8-Layer algebra network. The Gradient Boosted Trees Learner (**GBTL**) has fallen still further behind. Interestingly, a network of eight wide hidden layers, with 200 algebra neuron factories per layer, for a total of 1600 algebra neuron factories (*200 algebra neuron factories per layer producing an average of 1.5B concrete algebra neurons per test case*) was enough to boost performance significantly beyond the **ARC^{N4}** wide 2-Layer algebra network.

12. Conclusions

These experiments strongly indicate that there is a performance advantage when GLM algebraic expressions are fitted together in a feed forward acyclic network reminiscent of deep learning neural networks. However, these results are statistically indicative only. A great deal more research remains to be done.

There are other tools which use simple GP and/or GP merged with multiple regression, logit regression, and discriminant analysis. More experiments can help determine whether deep learning algebra networks benefit just the ARC tool; or, if other GP tools also benefit from deep learning algebra networks and whether these tools also benefit in a statistically similar manner?

These experiments were performed on a set of ten specific theoretical classification problems. Are deep learning algebra networks also beneficial with regression problems and in other real world problem domains?

These experiments involve only feed forward, acyclic, symmetric (*all hidden layers are the same width*) algebra networks. More experiments can help determine whether other types of deep learning algebra networks are more advantageous and in which problem domains.

The training method for these deep learning algebra networks involved optimizing the neurons in each layer against the dependent variable Y. This was done forwards, neuron by neuron, layer by layer, from the first hidden layer to the last hidden layer. Most deep learning neural nets are trained with quite different methods, wherein all coefficients in the network are adjusted backwards, from the dependent variable Y back to the first hidden layer. Is there a training method, *for algebra networks*, which works backwards on all coefficients in the network? Would this or some other training method provide statistically superior performance?

A tremendous advantage of GP deep learning is that we have much greater visibility into what is actually going on within the network, since each of the network algebra neurons are human readable. In the future, much of the mystery surrounding deep learning networks may be clarified.

Clearly there is much work remaining in studying deep learning algebra networks – far more work than our single research group has resources. As we pursue our continuing studies of algebra networks, it is our hope that these experiments will excite other researchers to pursue the many questions still remaining with GP deep learning.

13. REFERENCES

[1] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". *Neural Networks*. **61**: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637.

[2] Tsantekidis, Avraam; Passalis, Nikolaos; Tefas, Anastasios; Kanniainen, Juho; Gabbouj, Moncef; Iosifidis, Alexandros (July 2017). "Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks". 2017 IEEE 19th Conference on Business Informatics (CBI). Thessaloniki, Greece: IEEE: 7–12. *doi:10.1109/CBI.2017.23*. *ISBN 978-1-5386-3035-8*.

[3] Mehar Vijh, Deeksha Chandola, Vinay Tikkiwal, Arun Kumar; Stock Closing Price Prediction using Machine Learning Techniques. International Conference on Computational Intelligence and Data Science (ICCIDS 2019).

[4] Selvin, Sreelekshmy, R. Vinayakumar, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. (2017) "Stock price prediction using LSTM, RNN and CNN-sliding window mode." International Conference on Advances in Computing, Communications and Informatics (ICACCI): 1643-1647.

[5] Rout, Ajit Kumar, P. K. Dash, Rajashree Dash, and Ranjeeta Bisoi. (2017) "Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach." Journal of King Saud University-Computer and Information Sciences 29(4): 536-552.

[6] J.A., Nelder, and R. W. Wedderburn (1972). Generalized linear Models, in*Journal of the Royal Statistical Society*, Series A, General, 135:370-384.

[7] Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by means of Natural Selection.* The MIT Press. Cambridge, Massachusetts.

[8] Friedman, J. H. 1989. *Regularized Discriminant Analysis*. Journal of American Statistical Association 84 (405) 165-175.

[9] McLachan, Geoffrey, J. 2004. *Discriminant Analysis and Statistical Pattern Recognition.* Wiley. New York, NY.

[10] Korns, Michael F., 2017. Evolutionary Linear Discriminant Analysis for Multiclass Classification Problems. In GECCO Conference Proceedings '17, July 15-19, Berlin Germany 2017.

[11] Gulden, Kaya Uyanik, Nese, Guller, 2013. A Study On Multiple Linear Regression Analysis. Procedia – Social and Behavioral Sciences 106 (2013) 234 - 240. 4th International Conference on New Horizons in Education, New York, Elsevier

[12] Peng, Joanna, Lee, Kuk Lida, 2002. A Introduction to Logistic Regression Analysis and Reporting. The Journal of Educational Researchc 96(1):3-124., Elsevier

[13] Augasta, M.Gethsiyal & Kathirvalavakumar, T.. (2013). Pruning algorithms of neural networks — a comparative study. Central European Journal of Computer Science. 3. 105-115. 10.2478/s13537-013-0109-x.

[14] Michael R. Berthold and Nicolas Cebron and Fabian Dill and Thomas R. Gabriel and Tobias K\"{o}tter and Thorsten Meinl and Peter Ohl and Christoph Sieb and Kilian Thiel and Bernd Wiswedel, 2007. KNIME: The Konstanz Information Miner}. In Studies in Classification, Data Analysis, and Knowledge Organization, Springer, Berlin. ISBN 978-3-540-78239-1.

[15] Korns, Michael F. 2013. *Extreme Accuracy in Symbolic Regression*. Genetic Programming Theory and Practice XI. Springer, New York, NY.

[16] Korns, Michael F. 2012. *A Baseline Symbolic Regression Algorithm*. Genetic Programming Theory and Practice X. Springer, New York, NY.

[17] Korns, Michael F. 2015. *Highly Accurate Symbolic Regression for Noisy Training Data*. Genetic Programming Theory and Practice XIII. Springer, New York, NY.

[18] Korns, Michael F. 2016. *An Evolutionary Algorithm for Big Data Multiclass Classification Problems*. Genetic Programming Theory and Practice XIV. Springer, New York, NY.

[19] Korns, Michael F., 2018. Strong Typing, Swarm Enhancement, and Deep Learning Feature Selection in the Pursuit of Symbolic Regression-Classification. In Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice XVI, New York, New York, USA. Springer

[20] Korns, Michael F., 2012. Predicting Corporate Forward 12 Month Earnings, 2012. Theory and New Applications of Swarm Intelligence, ISBN 978-953-51-0364-6, edited by Rafael Parpinelli and Heitor S. Lopes, InTech Academic Publishers.

[21] Korns, Michael F., 2015. Trading Volatility Using Highly Accurate Symbolic Regression. In Ryan, et. al., Handbook Of Genetic Programming Applications, New York, New York, USA. Springer.

[22] Billard, Billard., Diday, Edwin. 2003. *Symbolic Regression Analysis*. Springer. New York, NY.

[23] Platt. J., 1998. *Sequential Minimal Optimization: A Fast Algorithm For Training Support Vector Machines*. Technical Report. Advances In Kernel Methods – Support Vector Learning.

[24] Karaboga, D, Akay, B.,. 2009. *A Survey: Algorithms Simulating Bee Swarm Intelligence*. In Artificial Intelligence Review, Springer, New York, NY.