An Alternative Paradigm for Machine Learning

A Detailed Description of the Ryskamp Machine Learning System

By Jeff Horton & Rix Ryskamp

Last Updated October 12, 2017

Executive Summary

useAIble's Ryskamp Learning Machine (RLM) creates a new paradigm for machine learning where a combination of mathematical and logical functions co-exist to create a framework that broadens the capabilities of machine learning in general. This framework is faster, capable of solving more types of problems, completely transparent about the decisions it makes, and adds many other new dimensions to machine learning.

The field of machine learning contains many powerful models that have accomplished some impressive feats. The majority of these machine learning systems are based upon a mathematical model and computations of equations, statistics, and other mathematical devices. While impressive, these traditional math-based systems have some significant drawbacks.

This document will compare the traditional approach to machine learning with the RLM to help readers understand how the different approach to machine learning operates. Although many alternatives exist, for our comparison, we will focus on artificial neural networks, as they are the most widely used at this time. This summary will look at the largest drawbacks with traditional machine learning and compare how the RLM overcomes these drawbacks.

Heavy Computational Requirements

Neural networks contain many neurons that are patterned after the neural network in the human brain. Each of these neurons contains connections to inputs, outputs, or other neurons. Associated with each incoming connection on the neuron is a weight that increases or decreases the significance of a connection to the neuron. For example, a weight of 1 would pass values directly through, while a weight of .5 would cut values in half as they pass through the neuron. A variety of formulas can be used to compute these values and their weights for each neuron, determine its "activation" or lack thereof, and change weight values to perform learning. The nature of traditional neural networks is such that each neuron will converge on its own portion of the problem during the learning cycle. Since neurons do not know in advance which portion of the problem is assigned to them, each neuron must be processed during every learning cycle as any neuron in the network has an equal chance of being relevant to any learning cycle. Thus, each calculation described above will be processed for each neuron in the network once per learning cycle.

This fundamental method creates an exponential computational requirement. The number of neurons required, particularly in what is known as hidden layers, expands as the complexity of the problem expands. As problem complexity increases, the width (number of layers) and depth (number of neurons per layer) increases exponentially.

For example, let's look at The Shape Network, which is solving a problem where the network is passed shapes and colors and must take a different action for each shape/color combination. If the network is known to only need to process three shapes and three colors, a *minimum* of nine (3x3) neurons are required. When the number of colors and shapes moves to 1,000 possible values for both shape and color, our *minimum* requirement moves to 1,000,000 (1,000x1,000). If the designer does not know the limitation of shapes and colors she will need to add additional neurons to handle any inputs that may occur, resulting in a large number of neurons that may or may not be used. Conversely, if an overall pattern exists between color/shape combinations that causes overlapping solutions among neurons then she may choose to use fewer neurons. This example is not meant to be complete, only to demonstrate that the number of neurons in a network grows exponentially as the problem size and complexity grows. Again, each of these neurons and their associated activation and learning functions must be calculated individually with each pass or cycle of learning.

This computational requirement is demonstrated by the hardware industry's effort to keep up with today's machine learning problems. For example, the NVIDIA DGX-1 V100 has 8 Tesla V100 GPUs, 40 Intel Xeon cores, 40,960 CUDA cores, and 5,120 tensor cores. The need for these various and numerous processors is a direct result of the above explained neuron calculation requirements.

With the Ryskamp Machine Learning engine (RLM), the paradigm changes. Let's look at The Shape Network example with 1,000 colors and 1,000 shapes. The RLM would still require 1,000,000 neurons. The difference is in *how* the neurons are processed. In the RLM neurons are preassigned to a specific portion of the problem (technically a unique input combination). So, the RLM would see a red square, find the neuron associated with red and square and process *ONLY* that neuron in the learning cycle. This means roughly 999,999 fewer calculations are required per learning cycle.

Imagine how much information could be processed with that same NVIDIA DGX-1 with a 1,000,000 times lighter load. Now consider that a set of 1,000,000 neurons is nowhere near the maximum requirement for today's machine learning problems. Training that now takes months or years can be reduced to minutes and days.

Limited to Math Based Problem Solving

The vast majority of machine learning is using an overall mathematical equation or statistical analysis to perform machine learning. Not surprisingly, the problems that these systems are good at solving can be generally explained by mathematics. Machine vision, speech analysis, and providing search engine results are all examples of real world problems that can be explained by math. Each of these examples use aggregation of information to find patterns and therefore solutions. A good way to simplify this concept is by using the phrase "problem x can be solved by statistical analysis of patterns in large sets of data". Search engines can find results by statistical analysis of patterns in large sets of data. Speech analysis is performed by statistical analysis of patterns in large sets of data. Problems where machine learning excels can typically fit well into this phrase.

So, do most problems that face today's organizations fit in this phrase? This is where the use of mathematics without logic breaks down. Many problems are better expressed with logical statements. Computer science often uses "IF, THEN" statements. In these statements an expression is solved and a computer will take one of two paths based upon the result. Most of the software we use today is fundamentally based upon the computation of expressions. However, it is critical to note that in a computer these expressions can be mathematical, logical, or both. For example, "IF X+Y*2>Z AND A DOES NOT EQUAL 2 THEN . . . ".

If we were to remove logical operators from computers, programs and their developers would spend an inordinate amount of time trying to use mathematical expressions to simulate logic that could have been easily expressed by a logical statement. It is the combination of logical operations and mathematical operations that make computers powerful.

At the electronic level computers use combinations of transistors to perform simple AND, OR, XOR, and NOT operations. These operations can be combined to solve logical problems, such as "True AND True = True" or "True OR False = True". These operations can also be combined into algorithms that simulate mathematical functions such as multiplication, division, addition, subtraction, etc. So, we learn from computers that the combination of logic and math are where true problem solving power is born.

The RLM uses this underlying concept within the context of machine learning. The RLM uses neurons, called Smart Neurons ®, that use logic first and then combine mathematics with that logic. Let's consider The Shape Network example. Now let's add switches and dials to our example. Switches will represent logical operations since they can only be on or off and dials will represent mathematical operations as they have an infinite number of positions. The weights in a neural network are very analogous to a dial while the RLM's mechanism is very analogous to a combination of switches and dials. So, if the RLM is sent a red square it will assign a Smart Neuron ® to [red], [square]. Let's say that our example must produce an answer of yes or no. The RLM will see red square for the first time and produce a result (we will discuss how later). If our result is yes for a red square and our network receives negative feedback (receiving positive or negative feedback is always part of machine learning) it will immediately store a switch for [red], [square], [yes] is [negative]. Next time we receive a red square, the RLM produces a no and receives positive feedback. It will now store a switch for [red], [square], [no] is [positive]. These two cycles are all that are required to correctly make predictions for red square. Unless something else is added to this problem, the correct switch for red square will always be used to produce a no.

Since our traditional neural network is using only dials, the neurons will have to see this problem many times before the dials "converge on the problem", that is they find a combination of dials that produces correct results for all cases. This is inherent, since dials have an almost infinite number of positions, while switches are simply on or off.

Now let's look at an actual proof of this underlying concept. At http://useaible.com/machine-learning-challenge/ you will find a simple maze puzzle that allows the user to run maze with the RLM against many popular machine learning systems including implementations in Google's TensorFlow ®. A quick review of this challenge shows that after one pass through the maze, the RLM will get a perfect score every subsequent time. No other system or method has been able to solve the maze (even after running for months). While it is certainly likely that one could set up a combination of traditional methods to learn to solve this maze, the point that it is a huge workaround for math to be solving a logical problem remains. The RLM can solve the maze simply as a result of saving the switches mentioned above. For example, [X=5], [Y=2], [Direction to Move=UP] will always be significantly simpler than a series of dials that tries to randomly converge upon this same result. Now consider that this random convergence of dials would be multiplied by the number of locations on the maze.

While this makes sense for logical only problems, we have not yet addressed mathematical and logical problems. We will go back to The Shape Network example. Logic is an easy solution when red is either present or not. What about shades of red? How will the RLM predict correctly if a new shade of red is presented? Let's change our input that was called "color" to "shade of red" and say that it can be a value between 1 and 1000. We will also assume that there is a clear linear pattern to this number, meaning that shades 99 and 100 look almost identical while 10 and 500 would not appear to be close shades of red.

We now present our RLM with square 500. The RLM produces a “yes”, receives positive feedback, and saves a switch for [square], [500], [yes], [positive]. Then we present square 600. The RLM produces a “no”, receives positive feedback, and saves a switch for [square], [600], [no], [positive].

When the RLM subsequently receives square 525 what will it output? It would be poor practice to save a switch for every one of the 1,000 shades or red before being able to make a prediction. This is where the RLM’s equivalent of dials comes into play. A dial, called a linear bracket in the RLM, may be set by default to 10% for shade of red. Since the 10% represents a tolerance value, any value between 450 and 550 will point to [square], [500], [yes], [positive] until additional Smart Neurons® appear in that range. As more Smart Neurons® appear, learning becomes more specific and less generalized. For example, when the 525 above is used it will produce a yes value. If the feedback system tells us that yes was negative then we now know that even though 500 and 525 are close, they are not to be treated the same. This process, called maturing of neurons, allows for an unprecedented combination of categorization and specific learning wherein an “edge case” that is an unexpected value within a certain category is tracked separately from the category around it and handled perfectly.

This handling of edge cases is critical in many fields. For example, in medicine a slight difference in the symptoms on stomach pain can mean the difference between giving a patient a pain reliever or sending her to immediate surgery for appendicitis. Traditional neurons networks are not well suited to find edge cases within broader categories without extensive repetition and training on large neural networks.

Complicated

Traditional machine learning systems are complicated compared to other computer science disciplines. Many machine learning engineering positions require a computer science degree and extensive post graduate work specifically in a machine learning field. This high threshold of required knowledge combined with an expansion of the use of machine learning by organizations has created a shortage of talent and driven up the costs associated with machine learning.

Although a comprehensive look at the reasons for these trends are outside the scope of this paper, we will look at a few factors that contribute to this trend. Firstly, there are many types of machine learning in addition to subtypes of formulas, parameters, and other configuration steps. Matching a problem to the correct type of machine learning with the correct settings is a significant task that requires in-depth training and skills. Secondly, the large dependence on relatively complex mathematics requires users to have a strong background in mathematics.

When compared to most other computer science fields, machine learning is behind the curve for simplifying user experiences and abstracting complicated concepts into simplified encapsulations. For example, a software developer no longer needs to understand the complexities associated with storage and retrieval of information to open a file in a computer program. She does not need to know what type of storage is being used, whether or not the file is local or accessed via a network, or any other myriad complexities associated with file access. These complexities have been abstracted into a simple object that can be accessed through simple commands with no understanding about or training on the underlying systems.

This abstraction has not been popularized with machine learning. There are many reasons that contribute to this situation; however, as long as the user of a system must be educated enough to set every variable perfectly for something to work correctly such encapsulation is not possible. In traditional machine learning, something as a simple as the wrong loss function or the number of hidden layers can have drastic negative impacts that may be costly and time consuming to correct.

The RLM changes this model. Its "mostly logic plus some math" approach takes the bulk of complexity out of the system. There are simply less moving parts. The moving parts that exist are primarily abstracted into simple APIs for use by common software developers. If a user wishes to be an RLM expert, she may do so with much less effort than would be required to master a traditional counterpart machine learning system. The underlying concepts that drive the RLM may be taught in weeks instead of months and mastered in months instead of years. UseAIble frequently teaches people with no previous machine learning experience to use the RLM to solve real world problems.

Not Explainable

Traditional neural networks cannot explain why they make the decisions they make. This problem, often called the "black box" problem, is a serious hindrance to the adoption of machine learning in critical situations where explainability is paramount. Examples include financial, medical, transportation, industrial, and other industries where logic must be scrutinized before it can be used.

A neural network's lack of explainability is easy to understand as is the RLM's ease of explainability. Let's go back to our example from above and think about the dials. Let's present our standard neural network with a red square. Until the feedback systems tells the network that the response for red square is perfect, it will continue to move all or some of the dials in the neurons present in the system until it achieves its desired result. No record is kept of the dial positions over time. Moreover, even if you did track knob position over time, the statistical nature of the reason why a dial was turned mathematically sound yet not understandable by humans trying to find a cause and effect relationship. We call this compound aggregation.

When you say that the average of 4, 5, and 6 is 4, you have aggregated 4, 5, and 6 into 4. Now let's say that the result of another aggregation tells us to make our 4 into a 5 as it deems 5 to now be more correct. Now we will take our 5 and combine with other results, say 10, 15 and average those to 10. Then we are told by another aggregation to move the 10 to 9 as it now deems 9 to be more correct. Now imagine this process taking place billions of times during a training session. Now imagine that your boss walks in and asks you why the final answer is 18 when his experience says it should be 2. This process is analogous to the everyday experience of machine learning engineers. When the answers are what the boss expects it is magic. When they are not the boss wants to start a discussion about manpower and hardware costs.

No complex mathematical formulas means no compound aggregation. The "switches" we saved above (e.g. [red], [square], [yes], [positive]) are all stored in a database along with time and date information and relationships between all inputs ever captured and all decisions ever made. With our visualization tools you can tell your boss exactly why the network chose 18. If he disagrees you can walk through the feedback logic to see if the machine was told the wrong answer or if it figured out something the boss did not already know. This entire process takes minutes and is clearly understandable. On our GitHub page at https://github.com/useaible/ryskamplearningmachine there is a sample application called RetailPoCSimple that demonstrates a very simple version of the explainability visualizations.

Conclusion

The RLM presents us with an alternate paradigm for solving problems with machine learning. While the RLM could be applied in many circumstances, we recommend considering it for problems where traditional machine learning has not already excelled, problems where resources and return on investment are critical, and problems where traditional machine learning is not an option and you desire the benefits of a self-learning system over traditional algorithmic programing, including costs, time-to-market, ROI, and optimization beyond human abilities.

RLM Architecture

High Level ANN Similarities

The RLM was designed and implemented from the ground up and is not a variation or derivation of any particular neural network or other machine learning system. However, there are some similarities, at least at a conceptual level, and some differences with existing machine learning technologies that may be helpful to review to give some context in understanding the RLM.

The most similar and probably the most helpful comparison is to that of a deep neural network. Like a deep neural network, or any neural network for that matter, the RLM is based on the concept of input neurons, hidden neurons and output neurons. As is expected, the input neurons are used to feed data into the network, for both training and prediction or other post-deployment activities, and the output neurons are used by the network to provide conclusions, recommendation, solutions, or any other output designated by the implementer.

As in a deep neural network, the RLM also contains any number of hidden neurons necessary to optimize a given problem. These neurons also contain state as in a traditional ANN. However, this is essentially where the similarity ends. A typical deep neural network is organized into two or more layers with neurons either fully or partially connected. The RLM's hidden neurons are not organized into a static architecture of layers but are more dynamic in organization as well as connections or relationships with other neurons. The neuron population as well as connections or relationships is dynamic in nature and changes over time as learning proceeds. Since the definition of a deep neural network is a network with two or more hidden layers, the RLM is technically not a deep neural network. However, it excels at essentially the same type of problems as a deep neural network with performance similar to or better, so it may be helpful to consider the RLM in the same category as deep neural networks.

Dynamic Neuron architecture

The RLM operates in a fully dynamic manner. Both the creation of neurons and the management of relationships among neurons are handled after network learning has begun. Neither the neuron population nor the connections between neurons need to be specified during upfront configuration. There is also no need to choose activation functions as neuron activation is handled by the RLM system based on incoming data patterns. This is due to that fact that the RLM is implemented in a logical domain instead of a mathematical domain.

The process of neuron creation is simple and straight forward. As data is encountered via the input neurons, SmartNeurons (explained further below) are created to handle each unique data pattern. Data patterns are defined and identified using the input value and input type for each input neuron. To some degree, this is the opposite of what happens in a traditional neural network where the associations between incoming data and a neuron or groups of neurons that handle that case are formed indirectly and independently from network design and configuration. In the RLM, the associations between incoming data and neurons to handle that data are formed deliberately and directly. The RLM refers to this design as Known Neuron Linkage which will be explored further below.

It is also important to understand that because of Known Neuron Linkage, neurons are not fired in the traditional sense of the term. Since SmartNeurons are deliberately created for and assigned to a unique data pattern, when subsequent data of the same values and type are encountered the neuron assigned to handle that case is directly activated by the RLM without the need to go through a chain of neuron activations via activation functions.

SmartNeurons

SmartNeurons are a trademarked term that refers to the type of neurons implemented in the RLM. SmartNeurons differ from the typical neurons found in a neural network, which commonly have a state of two weights and a bias or similar, because their state is comprehensive. The typical mathematical aggregation of neuron and network state from training iteration to training iteration does not exist in SmartNeurons. Every training incident that a SmartNeuron encounters, which may or may not happen on every training iteration or cycle (see Known Linkage explanation below), is logged. This comprehensive log of training includes input values that triggered the event, output neurons with their associated score or reinforcement feedback, and other RLM specific information. The availability of this comprehensive training data makes the learning process transparent to debugging as well as making explanation of output decisions after deployment explainable.

Constrained and typed Input Neurons

RLM input neurons are typed, meaning they are specified as integer/float/bool/string/etc., and their range of values is constrained to what values are applicable in their real world implementation. This may seem counter intuitive in the context of making a system as generalized as possible. However, it has both an important design intention as well as a very practical implementation benefit.

Making the input neurons typed gives the RLM additional knowledge about the incoming data that is useful in the design of Known Neuron Linkage (see explanation below). During the process of allocating and activating neurons based on unique incoming data patterns, data type allows the RLM to be more intelligent and specific in the dynamic process of creating and managing the neuron population and architecture.

Also, constraining the inputs to an appropriate range of values has the very practical use of allowing significant runtime performance optimizations. Since the RLM is directly and deliberately involved in neuron allocation for unique data patterns, having an appropriately limited range of values allows the RLM to intelligently avoid time and resources on a potentially large number of irrelevant cases and focus instead on only those scenarios that are involved in and important to the desired learning process and output solutions.

Scoring mechanism or Reinforcement Learning

The RLM can be considered a Reinforcement type machine learning system where the output that the system generates during each iteration or cycle of the learning process, is given some type of feedback that reinforces either a positive or negative behavior. The RLM's reinforcement or feedback mechanism is known in the RLM as a scoring mechanism.

The RLM's scoring mechanism is actually a dual or two-fold numeric design, meaning there are two numeric scores that are provided to the system during learning. The first score is applied at the most granular training level, the learning cycle or iteration. After each learning cycle, one set of input neurons each with unique values, activation of appropriate hidden neurons and finally a network generated output solution. This cycle-level score allows immediate feedback for each individual record or row of data.

The second level of scoring or reinforcement is at the session level. In typical neural network operation, there is the concept of an epoch, which is the completion of as many iterations as necessary to process the entire data set. The RLM has a concept of Sessions that can be thought of as similar to Epochs though it differs in theory and details as the RLM uses Sessions during unsupervised training. The Session Score allows the RLM to receive feedback with the broader scope of a larger set of data vs. the row or record level of data. For example, in a retail implementation of the RLM, to optimize product placement and revenue, a cycle score could be provided for individual products and shelf slots while a session score could be provided for an entire shelf or section of aisle.

This two part scoring mechanism proves for sophisticated and detailed learning configurations. In any reinforcement system, the quality of learning the network achieves is directly and fundamentally tied to the quality and accuracy of the feedback mechanism. The RLM's dual scoring configuration allows for control of learning that is needed in many complex real world optimizations.

The actual RLM scoring configuration is not part of the core system but is defined and implemented at the time of application to a specific problem or domain. The RLM core includes the reinforcement feedback to the network, but of necessity the actual positive or negative scoring values need to be defined uniquely to a given scenario and provided by the implementer of the RLM to the scenario.

Also, the score received by the system is stored as part of the SmartNeuron state or history. This makes it simple and straight forward to either debug network learning or to explain a decision after deployment. A map of input, learning cycles, output, and corresponding score can be easily generated from the SmartNeuron history

Known Neuron Linkage vs. Neuron Threshold or Activation Firing

Traditional neural networks are based on the fundamental concept of neurons containing a numeric state that if high enough, triggers some type of an activation function that in turn fires a subsequent neuron. This concept is inspired by the biological process of the human brain having neurons that fire other neurons based on the strength of connecting synapses. Neuron activation in the RLM is also inspired by the human brain but it departs from the concept of firing neurons at the biological level. Instead it operates more at the level of human cognitive thought. At this level, the human cognition is not aware of one neuron firing another but functions with the awareness of information received from the outside world, processing of that information (categorization, conclusion or action) and then storage and retrieval of the information.

The RLM implements this concept via a method dubbed *Known Neuron Linkage*. As information or data is passed to the RLM via inputs during the training process, patterns are detected based on data type and input position. Each of these unique data patterns is assigned a corresponding SmartNeuron, which handles that case and tracks a complete learning history for that case. The connections or relationships of a neuron are made directly to a unique set in input values through the neuron itself, to a case history of outputs that have been attempted during learning, and their corresponding scores. This means that the concept of neuron activation in the RLM does not happen via a chain of neurons and threshold functions. Activation of a SmartNeuron simply happens when the unique data pattern assigned to that neuron presents itself through the input neurons.

A significant benefit of Known Neuron Linkage is the possibility of drastically reduced training or learning time. Typically, a neural network with a million hidden neurons will need to tweak the weights and biases of each neuron a million times, as well as apply activation functions to each neuron on each training iteration. Conversely, because a specific set of input data with values is tied directly to one SmartNeuron, each training cycle usually consists only of retrieving and changing the state of that one SmartNeuron. In some cases, this has shown to reduce the training time by orders of magnitude.

Neuron Relationships

Since SmartNeurons are activated directly by the RLM instead of through a chain of fired neuron relationships, it may be suspected that no connections or relationships exist between SmartNeurons. While it is true that static connections between neurons in a traditional network do not exist, in the RLM, the SmartNeurons do in fact have relationships that are fundamental and integral to operation of the system.

These relationships are based on similarity of data (values and types) between neurons. These relationships are completely dynamic, based on state of the network during training as well as network configuration. Since the RLM is implemented in a logical domain, the relationship between neurons and incoming unique data patterns is defined by a replaceable algorithm. The algorithm simply defines how to determine a runtime similarity between neurons. The simplest form of this is known in the RLM as a linear bracket, which is a comparison of data values and how close or distant they are to each other. Another version could be a Euclidean calculation of distance between data patterns of assigned neurons as is common in K-Means algorithms. In fact, the relationship defining algorithm is a plug-able concept in the RLM which allows for adaptation of the system to the requirement of any known problem domain as well as unknown future situations.

The relationships between SmartNeurons are used in two situations. First, when a SmartNeuron is newly created it has no history of learning cases, so a Best Known Solution of the highest scoring neighbor may be used for output until the SmartNeuron has sufficiently matured to provide an intelligent output from its own case history. This is known in the RLM as the Richest Neighbor concept. The second situation for using SmartNeuron relationships is when categorization or classification is required of the network. The RLM provides a plug-able algorithm that is used to determine what SmartNeurons are similar and which are diverse. Also, the RLM allows input neurons to be designated as *distinct* or *linear*. A distinct input is used when values for the given input neuron must be determined exactly or distinctly, such as a social security number. Designating the input neuron as *linear* means that the output values have an acceptable range of values that are all considered more or less equally accurate. When inputs are designated as distinct, the algorithm to determine richest neighbor will not be used as it is critical that an exact or distinct value is used in determining output values.

Cycles and Sessions
Neural networks and typical machine learning systems complete supervised training through a process of iterations and epochs with iterations being one set or row of data passed through inputs, processed, output(s) generated, and a reinforcement process completed if appropriate. An epoch would be the completion of multiple iterations until the entire training dataset is processed.

The RLM has a similar concept for training but it is referred to as cycles and sessions. Cycles are identical to iterations, which operate on one set of input data or one iteration of training. Sessions are similar to epochs as they consist of many cycles or iterations but differ in that they are more oriented to unsupervised training than a finite dataset used for supervised training. As mentioned above in the discussion on scoring, cycles and sessions are used to implement a two part scoring system. This two part scoring mechanism proves for sophisticated and detailed reinforcement learning configurations.

Cases and Best Known Solutions
Cases and Best Known Solutions (BKS) are the organization method used by SmartNeurons to persist and recall the scoring or reinforcement received during training. A case is one instance of a cycle that includes associated inputs and the cycle score. A complete history of cases is stored with the SmartNeuron.

The RLM design and data persistence module has a concept known as the Best Known Solution (BKS). A BKS is simply the highest scoring case in the comprehensive history of SmartNeuron cases, encountered at any given point in time. When learning is sufficiently complete to justify deploying the system, the BKS is the answer that will always be used in generating values for the output neuron(s).   During the learning process, the BKS may or may not be used for output generation. Whether or not a BKS is used as an output during training depends on whether the RLM is still in a Randomized Learning state where it is trying random outputs to see if a solution is better than the current BKS can be found.

API Options
The RLM API provides an Application Programming Interface (API) for the software developer who implements or applies the RLM to a specific problem or optimization. The API allows for input and output neurons to be created and configured, control methods to manage the neural network creation, training, prediction and other operations. Currently APIs are supported for REST, MQTT, Python, & C#.

The system also includes a *Visualizer API* that can be used to access the learning history database to provide a tool for inspecting or debugging the learning process, as well as providing explanations for solutions, actions, or recommendations the system provides through the outputs during system deployment.