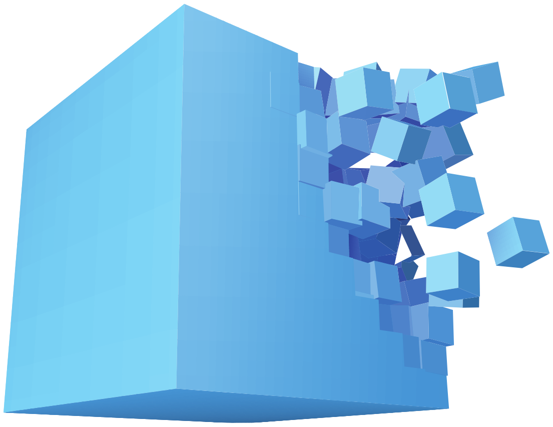


The Ryskamp Learning Machine – A Retail Implementation

Streamlined Version of Full Retail Application Showing Technical Implementation and Operation

By Brandon Taylor & Jeff Horton

Last Updated November 28, 2017

Executive Summary

RLM Machine Learning Breakthroughs

The Ryskamp Learning Machine (RLM) is a revolutionary improvement in machine learning that is unlike any currently available deep learning system. It is a business oriented solution envisioned, designed and implemented to provide artificial intelligence solutions that businesses need now.

The RLM solves a number of challenges and roadblocks that exist to wider spread adoption of current Artificial Neural Networks. Namely it, 1) provides complete transparency into the learning process, as well as explainability for all system solutions and output, 2) solves problems current ANNs can't, such as specific memory recall and immediate identification of edge cases or anomalous data currently requiring lengthy training periods, 3) provides a much-simplified ANN architecture and easy to understand API that can be used by regular software engineers, as well as a number of other significant advances.

AI and the Retail Store Planogram

useAIble has created an implementation of the RLM for the retail sector to demonstrate how the system operates, show that it functions in a distributed environement at scale and provide a competetive analysis. The purpose and function of this Retail Implementation is to apply machine learning to create a plan for product choices and shelf placement of those products that optimizes sales revenue in a retail store. In the retail industry, this plan for product choices and shelf placement is called a *Planogram*. Planograms are created to cover a given length of retail store shelf space and a given number of shelves. They take into account historical sales data and product attributes, but also require custom constraints on product choices and placement based on the particular and individual needs of a specific store.

At a high level, creating a planogram based on product information and sales data may seem like a straight forward problem for machine learning. However, with the addition of customized constraints that pose direct conflicts with sales data, product info and possibly with each other, the problem becomes much more difficult. useAIble conducted research where traditional ANN implementations, created in TensorFlow and Encog, were applied to planograms creation. The implementations were able to successfully create an optimized planogram with small data sets, however, as the datasets grew to medium and large size both systems became unable to complete the optimization in a reasonable amount of time for real world deployment (see Appendix A for a full report of the results and benchmarks.)

In discussions with Walmart, it was discovered that they also had made attempts to apply a traditional ANN implementation to the creation of planograms. The attempt had apparently not succeeded and planograms are still created today by teams of specialists, without the use of AI.

The RLM and Planograms

The RLM does not have the same difficulty with planograms as a traditional ANN. This is due to the fact that the RLM is not based on the statistical and mathematical architectures of current ANNs. It is instead designed to identify and handle, in addition to categorize, exceptional edge cases in data and is therefore able to easily solve the planogram problem. The RLM creates and optimizes a planogram in a fraction of the time it currently takes a team of human specialists and with results that surpasses those of the specialists.

Two RLM Retail Implementations

Full Retail Version

The useAIble team has created two versions of the Retail Implementation of the RLM. The first version is a near-production ready application with a distributed architecture capable of handling worldwide product planning. The system was created for Walmart but can easily be adapted to any retail scenario. The data used to train this system is real product data obtained from Walmart suppliers and kept current on a weekly basis. The data consists of thousands of items, each with numerous attributes such as size, color, scent, sales history and others. This version exists to demonstrate that the RLM and accompanying technology is capable of handling a full scale, world-wide deployment scenario.

Streamlined Technical Version

A second version of the retail implementation is targeted to technical evaluators and developers. This version is a streamlined derivative of the full retail system implementation. It optimizes the exact same scenario, in the same way as the full version, with three exceptions; 1) the code is designed and written in a streamlined way making it easy for technical evaluators and software developer to see how the RLM is implemented and functions 2) the product and sales data, unlike the full version, is generated by the system to facilitate simulation and validation of many more cases than would be available otherwise and 3) the UI is substantially different in order to accommodate a technical evaluation version vs. a full retail version.

This paper is provided to cover and accompany the Streamlined Technical Version of the RLM Retail Implementation.

It should also be noted, in order to understand useAIble's business relationship with Walmart and also for full transparency, that the discussions and technical engagement between the two companies is not for the purpose of providing a product for Walmart but for exploring the capabilities and viability of the RLM System. Walmart has indicated they are interested in the technology but desire to purchase it through a larger company that acquires useAIble and then productizes the technology.

Implementation Concepts and Detail

Design Theory

As mentioned above, the Streamlined Technical version or Retail PoC, provides a straight forward and easy way to watch the RLM perform in a retail setting, as well as get insight into some of the concepts of the system. The design intention of this version is therefore not to provide a UI that would be used in a production application, but a UI to give an understanding of and insight into the system as it creates and optimizes a planogram. To that end, the UI is laid out with a dual purpose, 1) to provide a flowchart type view of the high level steps the system goes through as it is learning, from data generation to training progress to planogram completion and 2) to provide technical information about the system and the training process including system configuration, benchmarking scores and training statistics.

Data Generation

In the full Retail Version, real-world data is used which includes historical sales data, product information and custom constraints. In the Streamlined Technical Version, data is generated to simulate these three categories for testing and validations purposes.

In the UI of this Streamlined Technical version, historical sales data is referred to as *Historical Metric Data,* Product Data is referred to as *Random Product Data* and Custom Constraints are referred to as *Retail Metric Constraints*.

In order to compare and demonstrate the RLM to its fullest, a number of scenarios of varying complexity and size were created and benchmarked. These scenarios increase in size and complexity in three dimensions; 1) the size of the retail shelf space, measured in number of shelves and number of slots, 2) the number of available products to choose from for shelf placement and 3) the minimum or target score required to achieve a successful completion. The smallest case is a 3 x 24 slot shelf, 500 available products and a target score of 76% and the most complex case is a 10 x 24 slot shelf, 5,000 products and a required score of 85%.

Configuration and Training Cycles

For this Retail Implementation, the organization of data and expected output is based on a given length of retail aisle space. This aisle space is then broken down into available shelves which is then broken down into available product slots within each shelf. In retail display planning, there is also a concept known as *number of facings* which specifies how many rows of a certain item should be placed on the shelf. Sales revenue is impacted by varying numbers of facings for different products. For example, optimal sales for a certain SKU of shampoo may indicate four facings, while optimal sales for a certain brand of pillow should only have one facing. The RLM will learn that the feedback it gets based on which products it picks from the pool of available products, where it places those products on the shelf and how many facings are optimal for each specific product.

Training of an RLM network consists of a series of cycles (also known as iterations) within a series of sessions (which are similar to epochs). The implementation of this retail PoC consists of a number of loops, in addition to cycles and sessions, that comprise the learning process. At the lowest or most granular level, the system will be fed one product slot at a time, which is unique to both the shelf number and position on shelf. The system will then propose a product to be placed in that slot. This completes one cycle of RLM training which also includes feedback. It is also at this slot or cycle level that number of facings are checked and determined. Over time the RLM will learn which product should be placed in the given slot, also taking into account number of facings, in order to maximize sales.

The RLM will proceed through training by looping through each slot on a shelf, one training cycle for each slot, and then looping through each shelf in the position on the shelf. Once each slot on every shelf has been processed, the RLM has completed one *session* of training. At this point two things happen, 1) Retail Metric Constraints are considered and compared to the product choices and placement that the RLM has made and 2) a session level score is given to the RLM. The RLM has now received two levels of scoring or feedback; a cycle score at the product slot level and a session score at the overall planogram level. Scoring will be discussed in more detail in the next section.

One additional aspect of interest in this Retail PoC benchmark is the repeating of the entire preceding training process. This is done solely for research purposes where it is desirable to repeat and compare a number of identical training processes in order to validate the accuracy of the outcome and identify any existing outliers.

Scoring Mechanism and Reinforcement Training

The RLM functions as a reinforcement type learning system; meaning, it goes through cycles or iterations of output proposals and receives feedback on the accuracy, effectiveness or completeness of the output. Within the RLM this is known as a scoring mechanism and provides positive or negative numeric feedback to the system. The scoring mechanism and how it is designed is the facility that determines exactly what the system learns.

For the RLM Retail Implementation the scoring mechanism is setup to consider several factors. The system needs to not only learn how to choose products and where to place them for a maximum score, but it also needs to learn how many facings are optimal for each chosen product while still honoring each of the specified Retail Metric Constraints. The scoring for this PoC is designed to teach the RLM at the cycle score level to consider number of facings and to honor Retail Metric Constraints at the session score level.

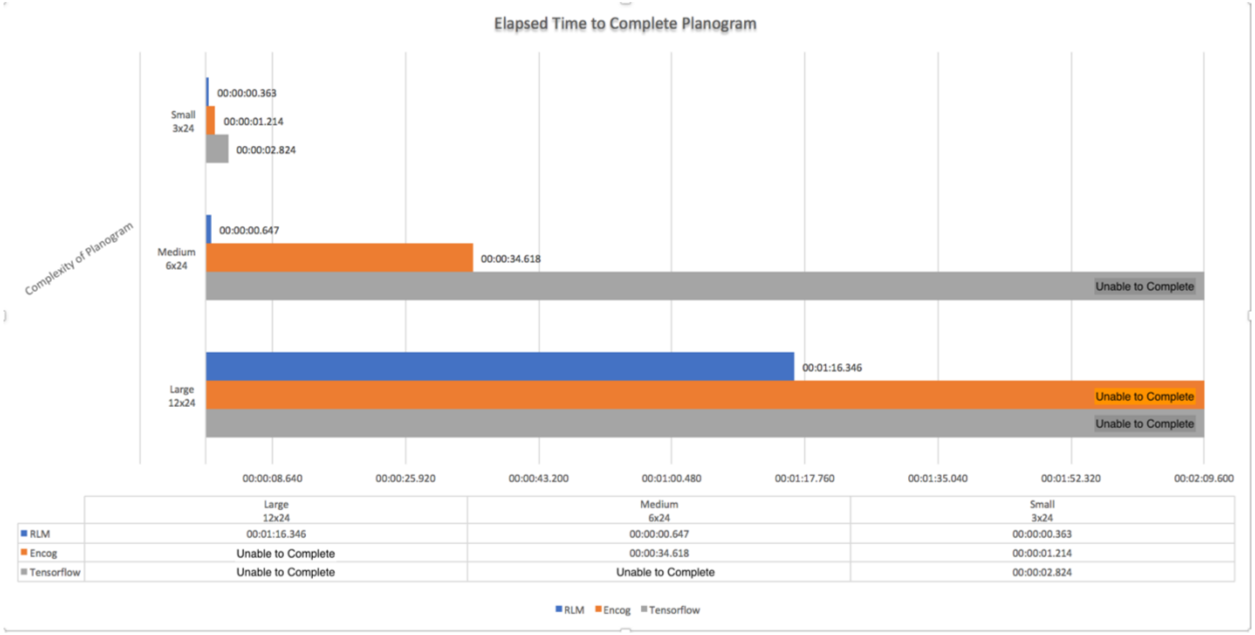
Benchmark Results

In order to benchmark this Retail Implementation of the RLM, ten scenarios of increasing size and complexity were setup. As mentioned above, this includes size of the retail shelf space in product slots, number of available products to choose from for placement on the shelf and the minimum or target score required to be considered a viable planogram. In addition to benchmarking the RLM itself, two additional traditional ANNs were created for competitive analysis - an Encog Genetic algorithm and a TensorFlow implementation using the Adam Optimizer and Softmax Activation. Each of the engines were allowed to complete their training process three times for each of the ten scenarios and then an average was taken of each for comparison.

Several key outcomes were observed.

1) The RLM was the only engine able to solve a large retail space planogram.
2) On a small problem (the only size with all three comparable ANNs), the outcome was:
    a. The RLM was 3.3 times faster than Encog.
    b. The RLM was 7.8 times faster than TensorFlow.
3) The Encog implementation was able to solve small and medium sized problems, but unable to solve large problems.
4) The TensorFlow implementation was able to solve small problem sizes, but unable to solve either medium or large sized problems.

The following graph shows the elapsed time for a small, medium and large planogram test. The horizontal bars are elapsed time, which means shorter is better. The numbers are times in the format of hour:minute:second. See Appendix A the for full benchmark report.



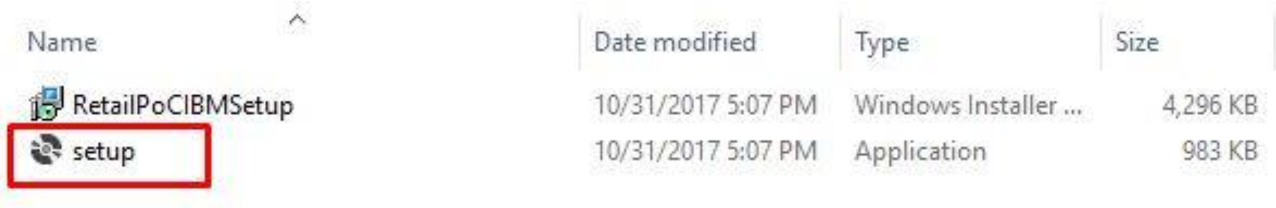
	Large 12x24	Medium 6x24	Small 3x24
■ RLM	00:01:16.346	00:00:00.647	00:00:00.363
■ Encog	Unable to Complete	00:00:34.618	00:00:01.214
■ Tensorflow	Unable to Complete	Unable to Complete	00:00:02.824

Executing and Operating the Proof of Concept

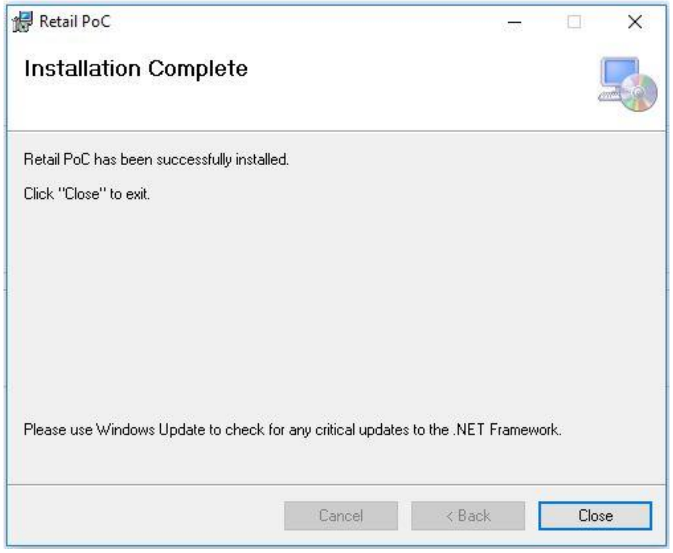
Set Up and Installation

Users will download an installer containing everything needed to run and operate the application. The installer can be accessed at the following location https://github.com/useaible/RyskampLearningMachine/blob/master/ExampleApps/CSharp/RetailPoC%202.0/Installers/RetailPoCIBMSetup.msi . Once the file is downloaded select the "setup" option to initiate the installation process.

A setup wizard will guide you through the steps required to install the Retail Shelf Builder application on your computer. The user will be notified when the setup is complete and in icon to launch the application will be placed on the user’s desktop.



Name	Date modified	Type	Size
RetailPoCIBMSetup	10/31/2017 5:07 PM	Windows Installer ...	4,296 KB
setup	10/31/2017 5:07 PM	Application	983 KB



Installation Complete

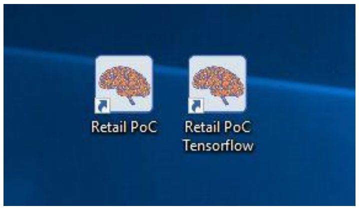
Retail PoC has been successfully installed.

Click "Close" to exit.

Please use Windows Update to check for any critical updates to the .NET Framework.

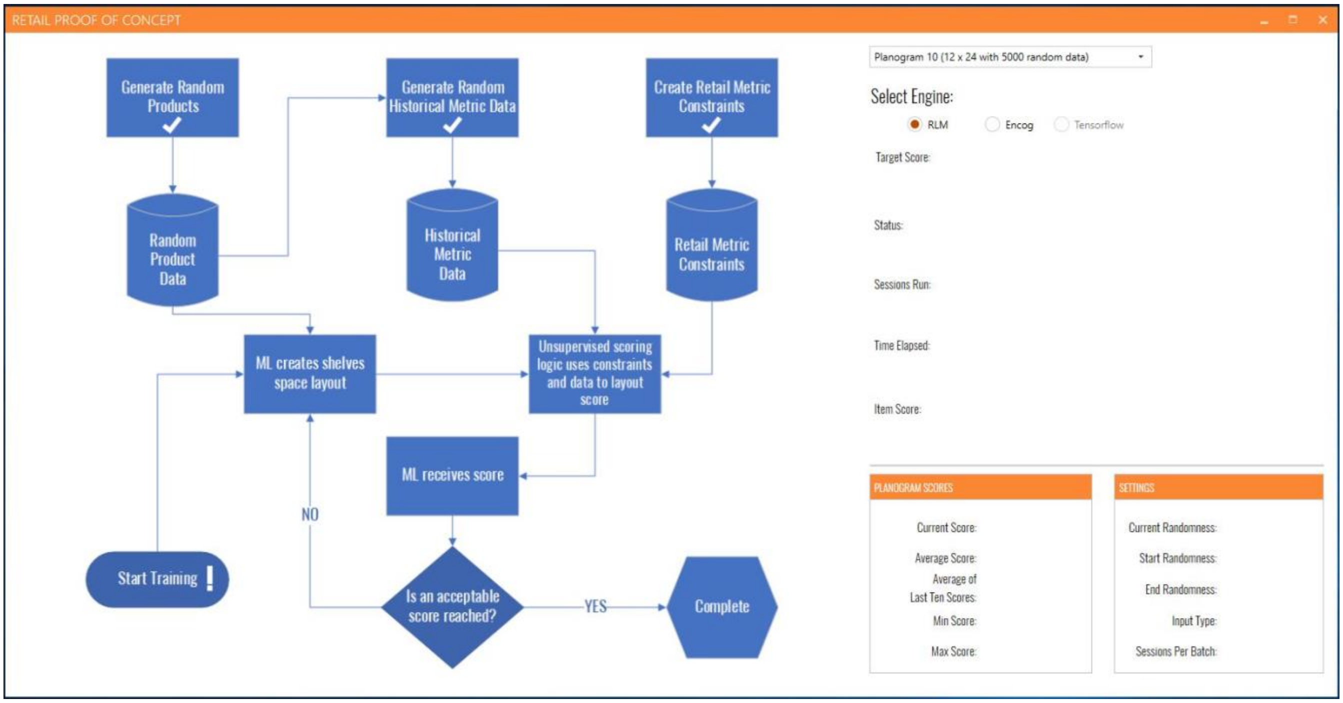
Running the Application

After installation, the application icon will be located on your desktop. You will have both a useAIble/Encog version (RetailPOC) and a TensorFlow version (RetailPOC TensorFlow) of the application. Upon clicking the useAIble/Encog application icon, the application will begin running and the user will be taken to the application home screen. The TensorFlow version will only work if you already have a working version of python and TensorFlow on your machine. Upon opening the TensorFlow version you will be taken to a screen to confirm where TensorFlow is installed on your machine. If the address for your installation is correct, simply press okay and the application will begin running.



Application Layout and Functionality

The main page of the application is divided into three separate sections. The first part is the flow chart that shows the flow of the data simulation and training processes. The second section is the engine settings. These engine settings give the ability to select the engine and data settings for the users. The final section is the data visualization table, which provides users with the ability to view your selected settings and offers details around the current status of the engine as it is running and training.


RETAIL PROOF OF CONCEPT
Generate Random Products ✓
Generate Random Historical Metric Data ✓
Create Retail Metric Constraints ✓
Random Product Data
Historical Metric Data
Retail Metric Constraints
ML creates shelves space layout
Unsupervised scoring logic uses constraints and data to layout score
ML receives score
NO
Start Training !
Is an acceptable score reached?
YES
Complete

Planogram 10 (12 x 24 with 5000 random data)

Select Engine:

● RLM ○ Encog ○ Tensorflow

Target Score:

Status:

Sessions Run:

Time Elapsed:

Item Score:

PLANOGRAM SCORES

Current Score:

Average Score:

Average of Last Ten Scores:

Min Score:

Max Score:

SETTINGS

Current Randomness:

Start Randomness:

End Randomness:

Input Type:

Sessions Per Batch:

The Flow Chart

The Flow Chart is an interactive display that gives users the ability to visualize the learning process as it is happening. Following is a description of each item on the Flow Chart.

- **Start Training** - This is a selectable object that will initiate the training and learning process.
- **Generate Random Products** - This creates the product catalogue and generates the product data the application will use for training. This step is selectable and upon selection will bring up the data generation panel.
- **Generate Random Historical Metric Data** - This is not a selectable step. This step represents the generation of historical data that is tied to the randomly generated products in the previous step.

- **Create Retail Metric Constraints** - This is a selectable step. Upon selection, the user will be presented with metrics sliders that can be adjusted and customized to the user's preference. Each metric represents a different weight that will be used for scoring during the training process.
- **Data Storage Containers** - These icons represent storage of the generated data.
- **ML Creates Shelves and Spacing Layout** - This step is not selectable. It represents the creation of a virtual retail shelf and provides the allotted amount of space available for product placement.
- **Unsupervised Scoring Logic Uses Constraints and Data to Layout Score** - This step is not selectable. It represents the creation of dynamic scoring based on the metric constraints selected above.
- **ML Receives Score** - This step is not selectable. It is where the shelf layout provided by the engine is scored based on the dynamic scoring discussed above.
- **Is an Acceptable Score Reached?** - This step is not selectable. When an acceptable score is reached the machine learning stops.

Engine Settings

- **Data size drop down** - This drop-down box gives the user the ability to change the size and complexity of the problem the engine will be solving. A larger dataset will take longer to train and solve.
- **Select Engine** - Currently you can select either the RLM or the Encog engine to run the problem through. TensorFlow currently can only be run through a separate python based application. Only one engine can be selected at a time.
- **Status** - This indicates the current status of the simulation. The available status codes are "Ready" (ready to run but not started), "Initializing" (simulation has been started and is loading in preparation to run), "Running" (Training simulation is in progress) and "Done" (training is complete and you can now view results).
- **Sessions Run** - This field provides a real-time view on how many sessions have been run during the simulation.
- **Time Elapsed** - This field displays the time elapsed from when the simulation was initiated, and will stop running when the simulation completes.
- **Item Score** - This will display the score for a single selected item within the shelf space.

Data Visualization Table

The data visualization table gives a real-time overview of the simulation including statistics and selected settings.

- Program Scores
  - Current Score: The score of the current running session
  - Average Score: The average score achieved across all sessions in the simulation

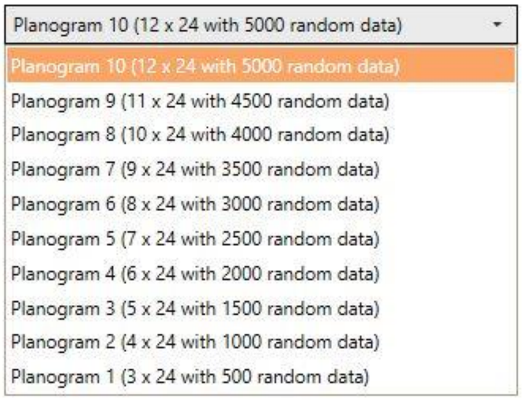
    - Average of Last Ten Scores. The average score of the last ten sessions in the current simulation
    - Min Score: The minimum score achieved across all sessions in the simulation
    - Max Score: The maximum score achieved across all sessions in the simulation

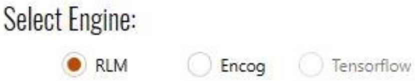
- Settings
    - Current Randomness: The current randomness being applied to the running session
    - Start Randomness: The initial randomness at the beginning of the simulation
    - End Randomness: The final randomness being applied when the simulation ends
    - Input Type: Specifies the input classification
    - Sessions Per Batch: The number of sessions run per batch during the simulation

Running the Simulation

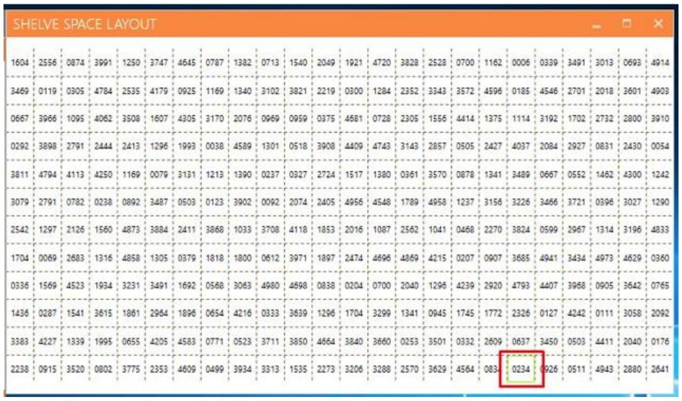
In order to begin running the simulation the user will need to select the size of the dataset they would like to use. The default is a 12x24 shelving space with 5000 random products in the data set. If you would like to use a different setting you can click on the drop-down box and select another option.

Once users have selected the size and layout for shelving, the machine learning engine can be selected. Currently you can select RLM for the UseAIble engine or Encog for the Encog engine. TensorFlow is currently only accessible through the TensorFlow python based app. Only one engine can run at a time but users do have the ability to use the same data set across all engines to compare results.


Planogram 10 (12 x 24 with 5000 random data)
Planogram 10 (12 x 24 with 5000 random data)
Planogram 9 (11 x 24 with 4500 random data)
Planogram 8 (10 x 24 with 4000 random data)
Planogram 7 (9 x 24 with 3500 random data)
Planogram 6 (8 x 24 with 3000 random data)
Planogram 5 (7 x 24 with 2500 random data)
Planogram 4 (6 x 24 with 2000 random data)
Planogram 3 (5 x 24 with 1500 random data)
Planogram 2 (4 x 24 with 1000 random data)
Planogram 1 (3 x 24 with 500 random data)



To begin the session and start training, the user will click on the "Start Training" icon in the flowchart section of the page. This will begin the data generation and run the simulation. Once the simulation is started the shelf space layout screen will appear. This screen gives you a virtual representation of your available shelf space and the products that are being placed within each space on the shelf. Each space on the shelf will have a SKU number representative of a product out of the database. When users hover over an item, the item's current score will be displayed on the main page of the application in the Settings section in the “Item Score” field. Double Clicking on an item will open the RLM Visualizer Panel so that users can see a visualization of the learning evolution for that product.

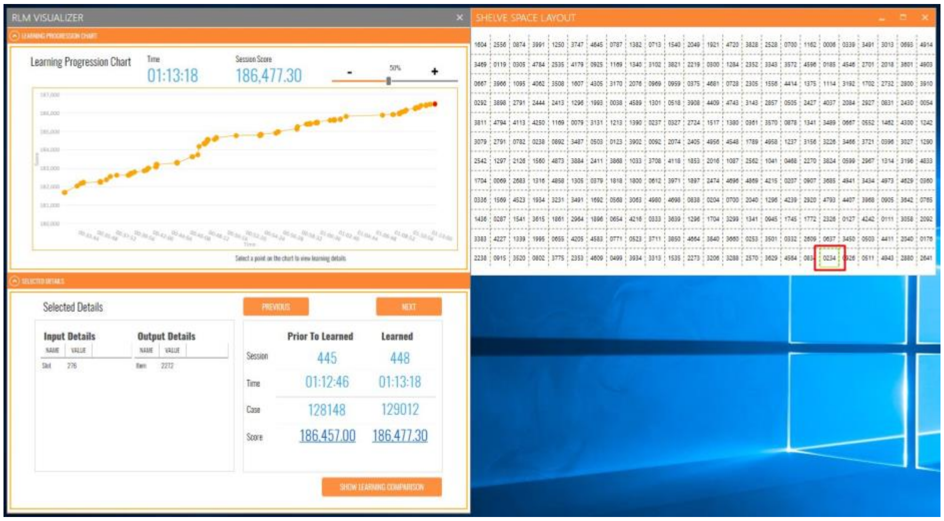


SHELVE SPACE LAYOUT

1604	2556	0874	3991	1250	3747	4645	0787	1382	0713	1540	2049	1921	4720	3828	2528	0700	1162	0006	0339	3491	3013	0693	4914
3469	0119	0305	4784	2535	4179	0925	1169	1340	3102	3821	2219	0300	1284	2352	3343	3572	4596	0185	4546	2701	2018	3601	4903
0667	3966	1095	4062	3508	1607	4305	3170	2076	0969	0959	0375	4681	0728	2305	1556	4414	1375	1114	3192	1702	2732	2800	3910
0292	3898	2791	2444	2413	1296	1993	0038	4589	1301	0518	3908	4409	4743	3143	2857	0505	2427	4037	2084	2927	0831	2430	0054
3811	4794	4113	4250	1169	0079	3131	1213	1390	0237	0327	2724	1517	1380	0361	3570	0878	1341	3489	0667	0552	1462	4300	1242
3079	2791	0782	0238	0892	3487	0503	0123	3902	0092	2074	2405	4956	4548	1789	4958	1237	3156	3226	3466	3721	0396	3027	1290
2542	1297	2126	1560	4873	3884	2411	3868	1033	3708	4118	1853	2016	1087	2562	1041	0468	2270	3824	0599	2967	1314	3196	4833
1704	0069	2683	1316	4858	1305	0379	1818	1800	0612	3971	1897	2474	4696	4869	4215	0207	0907	3685	4941	3434	4973	4629	0360
0336	1569	4523	1934	3231	3491	1692	0568	3063	4980	4698	0838	0204	0700	2040	1296	4239	2920	4793	4407	3968	0905	3642	0765
1436	0287	1541	3615	1861	2964	1896	0654	4216	0333	3639	1296	1704	3299	1341	0945	1745	1772	2326	0127	4242	0111	3058	2092
3383	4227	1339	1995	0655	4205	4583	0771	0523	3711	3850	4664	3840	3660	0253	3501	0332	2609	0637	3450	0503	4411	2040	0176
2238	0915	3520	0802	3775	2353	4609	0499	3934	3313	1535	2273	3206	3288	2570	3629	4564	0834	0234	0926	0511	4943	2880	2641

RLM Visualizer

The visualizer tool gives users specific insights into the decisions made by the engine throughout the learning process. It displays a learning progression chart that shows the scores of each item placed in that shelf space so that users can visualize the learning curve of the engine. Upon selecting a specific change along the learning timeline, the user will be able to see the inputs and outputs that lead the engine to make that decision.



Time 01:13:18

Session Score 186,477.30

Select a point on the chart to view learning details

Selected Details

Input Details

NAME	VALUE
Slot	276

Output Details

NAME	VALUE
Item	2272

	Prior To Learned	Learned
Session	445	448
Time	01:12:46	01:13:18
Case	128148	129012
Score	186,457.00	186,477.30

1604	2556	0874	3991	1250	3747	4645	0787	1382	0713	1540	2049	1921	4720	3828	2528	0700	1162	0006	0339	3491	3013	0693	4914
3469	0119	0305	4784	2535	4179	0925	1169	1340	3102	3821	2219	0300	1284	2352	3343	3572	4596	0185	4546	2701	2018	3601	4903
0667	3966	1095	4062	3508	1607	4305	3170	2076	0969	0959	0375	4681	0728	2305	1556	4414	1375	1114	3192	1702	2732	2800	3910
0292	3898	2791	2444	2413	1296	1993	0038	4589	1301	0518	3908	4409	4743	3143	2857	0505	2427	4037	2084	2927	0831	2430	0054
3811	4794	4113	4250	1169	0079	3131	1213	1390	0237	0327	2724	1517	1380	0361	3570	0878	1341	3489	0667	0552	1462	4300	1242
3079	2791	0782	0238	0892	3487	0503	0123	3902	0092	2074	2405	4956	4548	1789	4958	1237	3156	3226	3466	3721	0396	3027	1290
2542	1297	2126	1560	4873	3884	2411	3868	1033	3708	4118	1853	2016	1087	2562	1041	0468	2270	3824	0599	2967	1314	3196	4833
1704	0069	2683	1316	4858	1305	0379	1818	1800	0612	3971	1897	2474	4696	4869	4215	0207	0907	3685	4941	3434	4973	4629	0360
0336	1569	4523	1934	3231	3491	1692	0568	3063	4980	4698	0838	0204	0700	2040	1296	4239	2920	4793	4407	3968	0905	3642	0765
1436	0287	1541	3615	1861	2964	1896	0654	4216	0333	3639	1296	1704	3299	1341	0945	1745	1772	2326	0127	4242	0111	3058	2092
3383	4227	1339	1995	0655	4205	4583	0771	0523	3711	3850	4664	3840	3660	0253	3501	0332	2609	0637	3450	0503	4411	2040	0176
2238	0915	3520	0802	3775	2353	4609	0499	3934	3313	1535	2273	3206	3288	2570	3629	4564	0834	0234	0926	0511	4943	2880	2641

Technical Intro to Implementing the RLM

The RLM, as defined and patented by useAIble, is a set of discrete as well as interoperable machine learning concepts and technologies. These concepts and technologies are independent of any particular software and hardware system or specific development environment. useAIble also provides a reference implementation that is complete and robust enough to be used both for evaluation purposes as well as a deployable system.

The provided reference implementation is built using Microsoft Visual Studio and C#. The system can be used via provided interfaces for Python, REST, MQTT and of course C#. The full version is buildable and runnable on Microsoft Windows with a slightly limited version available with Visual Studio for Apple's OS X.

RLM API Overview

The RLM provides an API that is used to configure the system before training, to manage and control the system during training, to operate the network after deployment for providing predictions and also to access the transparency and explainability aspects of the RLM.

Configuration
Before training or deployment, the RLM needs to be configured and created. This process is quite straightforward in relation to the setup and configuration of a tradition ANNs. It consists of API calls to complete three steps.

1. **Specify and Create Input and Output Neurons**. These neurons need to represent the data that will be passed into the system for training and predicting, as well as the output expected from the system. This implies that the usual data analysis, cleaning and structuring needs to be completed to ensure correct training of the network. Also, for optimization and architecture reasons, a data type and range needs to be specified for each input neuron.
2. **Create or Load an Instance of the RLM.** When a new training project begins, a new RLM instance needs to be created based on the previously defined input and output neurons as well as a few other configuration parameters. The RLM also includes the option to load a preexisting instance. This is useful for scenarios where full training must consist of more than one intermediate training session. This could be for purposes of iterative training, trial and error approaches, debugging data applicability and validity or other needs.
3. **Define a Scoring (or Reinforcement) Mechanism.** The RLM is a reinforcement type machine learning system. This means that training consists of a number of iterations where each cycle consists of the RLM receiving a set of input data values, processing them, providing the specified output and then receiving feedback on the correctness or effectiveness of the output. The RLM architecture consists of a two-level feedback

mechanism; a first for fine grained control at the data record level and a second for larger grained or higher-level feedback (see Additional Info below for a more detailed explanation). This provides for very sophisticated control of learning in complex real-world scenarios. In the RLM architecture, the feedback is defined and provided to the system by the implementer of the RLM for a certain domain. The scoring mechanism needs to be precisely understood and specified as this determines what the system does and does not learn.

Training and Prediction

RLM operation is organized into levels of training control referred to as cycles and sessions. A cycle consists of one pass of RLM training, which includes one set of values for each input neuron and one network output (which may be one or more output neurons). A cycle score is provided for each cycle of training. A session is a specified number of training cycles. The number of cycles a session consists of is defined by the implementer of the RLM to a given domain and allows for a broad view control of the learning process. A session score is also provided to the RLM at the end of each session. The API provides methods to control and execute cycle and sessions training.

Additional Information

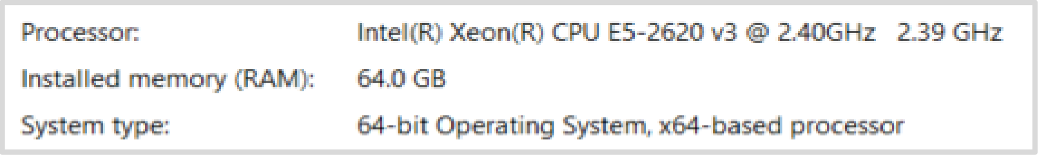
1. RLM Developer Guide
2. RLM API Reference
3. RLM - A Brief Technical Intro
4. RLM - A White Paper Description
5. The RLM and Explainability
6. RLM on GitHub

Appendix A

Full Benchmark Report

System Configuration

The system used for benchmark tests were run on a Windows 64-bit Xeon processor with 64GB or RAM. A GPU was not used for the tests.



Processor:	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz 2.39 GHz
Installed memory (RAM):	64.0 GB
System type:	64-bit Operating System, x64-based processor

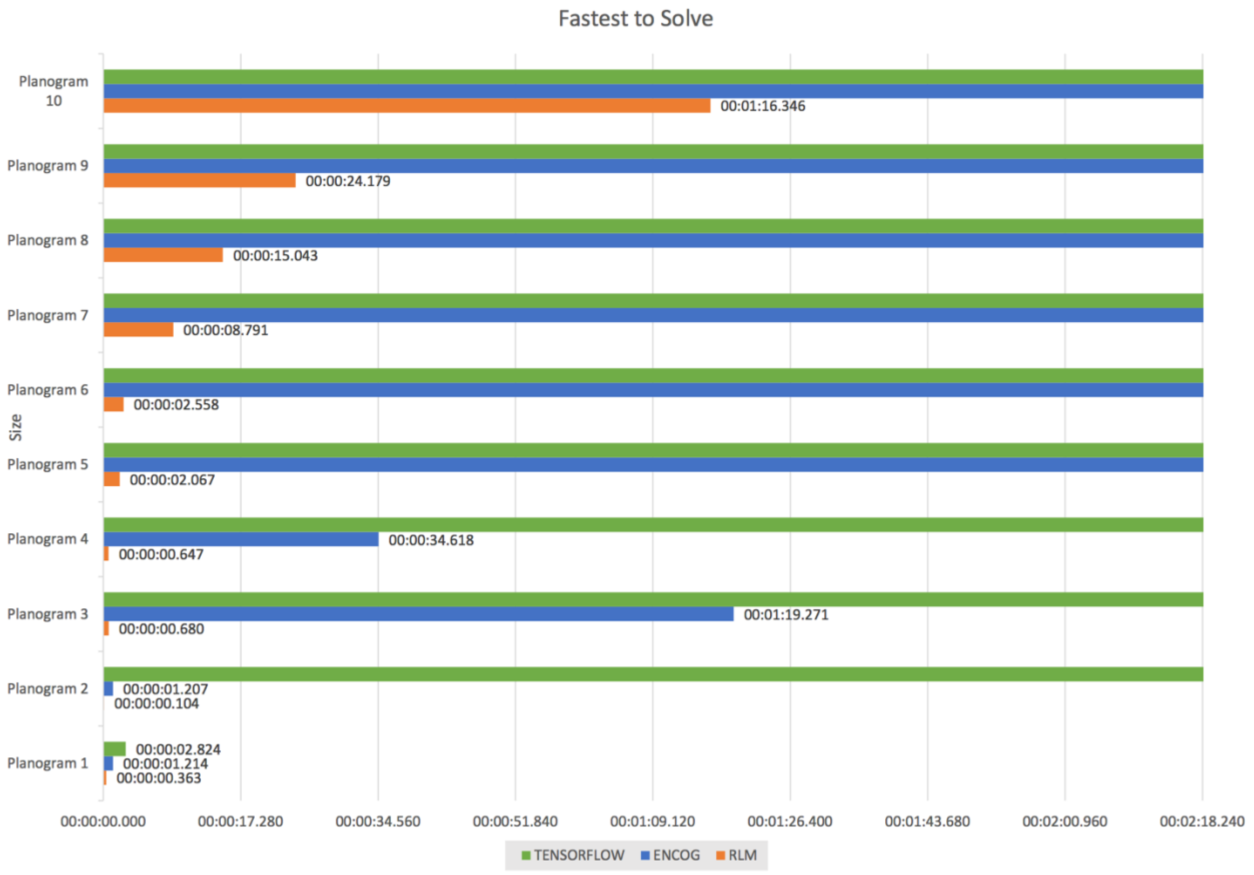
Machine Learning Engine Configuration

Each of the three systems have diverse methods for configuration and therefore aren't directly comparable. However, the attempt was to configure each system in the manner that it performed best at. The configurations are shown below.

RLM SETTINGS	
Start Randomness	2
End Randomness	0
Input Type	Distinct
ENCOG SETTINGS	
Training Type	Genetic Algorithm
Population Size	200
Network Pattern	Feed Forward
Activation Function	Elliot Symmetric
Hidden Layers	1
Hidden Layer Neurons	200
TENSORFLOW SETTINGS	
Optimizer	Adam Optimizer
Activation Function	softmax
Hidden Layers	3
Hidden Layer Neurons	500

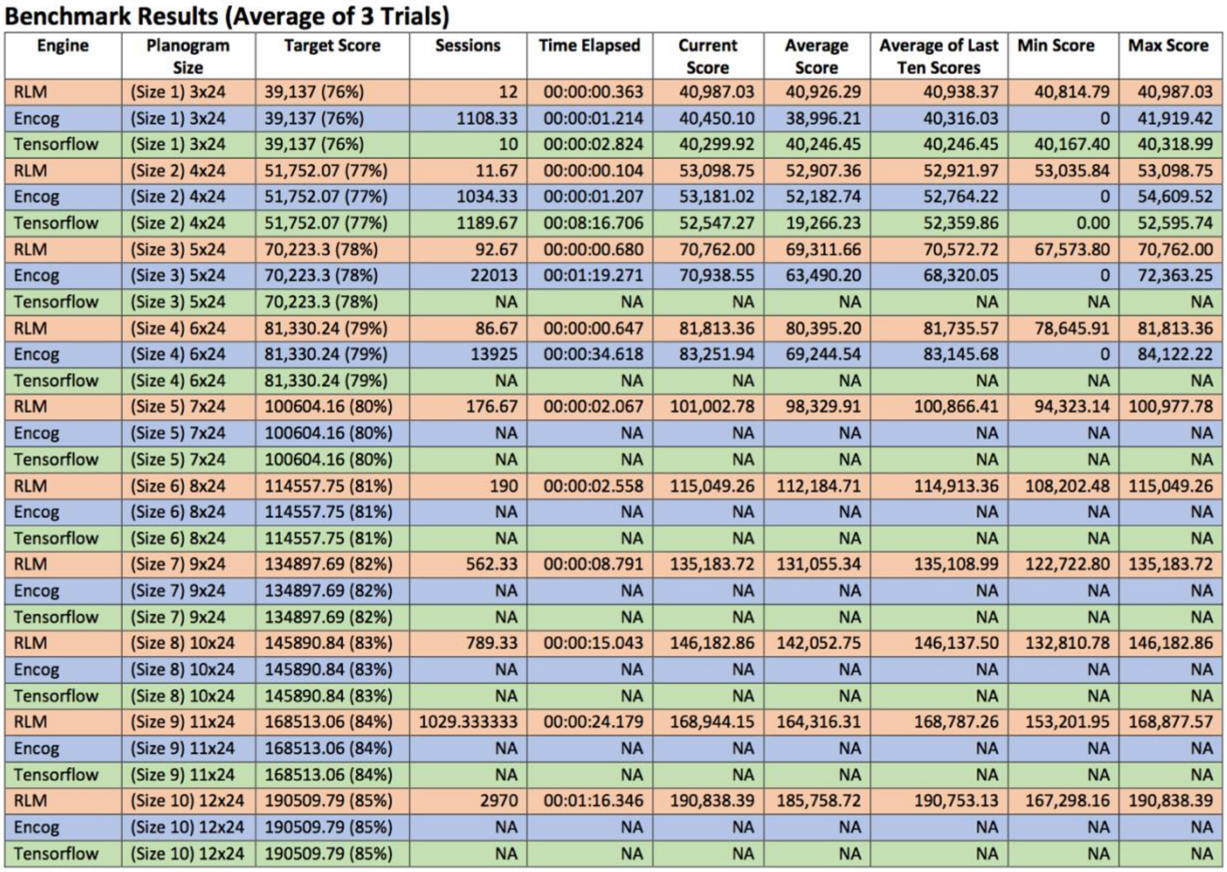
Comparison of Fastest Times to Solve Each Problem Size

The following table shows the elapsed time it took the RLM, Encog and TensorFlow to solve each of the various plan-o-gram sizes. Ten sizes were benchmarked ranging from plan-o-gram 1, the smallest and easiest, to plan-o-gram 10 the largest and most complex. If the engine was not able to solve the plan-o-gram no score is given and the corresponding bar extends to the right of the graph. TensorFlow was only able to solve Plan-o-gram 1 with no score for the remaining and Encog was able to solve Plan-o-Gram 1 - 4 with no score for the more complex scenarios.


Fastest to Solve
Planogram 10
00:01:16.346
Planogram 9
00:00:24.179
Planogram 8
00:00:15.043
Planogram 7
00:00:08.791
Planogram 6
00:00:02.558
Size
Planogram 5
00:00:02.067
Planogram 4
00:00:34.618
00:00:00.647
Planogram 3
00:01:19.271
00:00:00.680
Planogram 2
00:00:01.207
00:00:00.104
Planogram 1
00:00:02.824
00:00:01.214
00:00:00.363
00:00:00.000
00:00:17.280
00:00:34.560
00:00:51.840
00:01:09.120
00:01:26.400
00:01:43.680
00:02:00.960
00:02:18.240
🟩 TENSORFLOW
🟦 ENCOG
🟧 RLM

Raw Benchmark Numbers for Each Problem Size

Below are the raw results of the plan-o-gram benchmark tests. Each of the three engines, the RLM, Encog and TensorFlow was tested with ten different plan-o-gram complexities ranging from the easiest Size 1 to the hardest Size 10. The *Target Score* represents a plan-o-gram that is considered sufficiently optimized for use in a retail store. *Sessions* is the number of learning sessions or epochs that the engine took to complete the test. Five types of scores for each engine are included to give various viewpoints and insights into each engines progress and status.



Benchmark Results (Average of 3 Trials)

Engine	Planogram Size	Target Score	Sessions	Time Elapsed	Current Score	Average Score	Average of Last Ten Scores	Min Score	Max Score
RLM	(Size 1) 3x24	39,137 (76%)	12	00:00:00.363	40,987.03	40,926.29	40,938.37	40,814.79	40,987.03
Encog	(Size 1) 3x24	39,137 (76%)	1108.33	00:00:01.214	40,450.10	38,996.21	40,316.03	0	41,919.42
Tensorflow	(Size 1) 3x24	39,137 (76%)	10	00:00:02.824	40,299.92	40,246.45	40,246.45	40,167.40	40,318.99
RLM	(Size 2) 4x24	51,752.07 (77%)	11.67	00:00:00.104	53,098.75	52,907.36	52,921.97	53,035.84	53,098.75
Encog	(Size 2) 4x24	51,752.07 (77%)	1034.33	00:00:01.207	53,181.02	52,182.74	52,764.22	0	54,609.52
Tensorflow	(Size 2) 4x24	51,752.07 (77%)	1189.67	00:08:16.706	52,547.27	19,266.23	52,359.86	0.00	52,595.74
RLM	(Size 3) 5x24	70,223.3 (78%)	92.67	00:00:00.680	70,762.00	69,311.66	70,572.72	67,573.80	70,762.00
Encog	(Size 3) 5x24	70,223.3 (78%)	22013	00:01:19.271	70,938.55	63,490.20	68,320.05	0	72,363.25
Tensorflow	(Size 3) 5x24	70,223.3 (78%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 4) 6x24	81,330.24 (79%)	86.67	00:00:00.647	81,813.36	80,395.20	81,735.57	78,645.91	81,813.36
Encog	(Size 4) 6x24	81,330.24 (79%)	13925	00:00:34.618	83,251.94	69,244.54	83,145.68	0	84,122.22
Tensorflow	(Size 4) 6x24	81,330.24 (79%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 5) 7x24	100604.16 (80%)	176.67	00:00:02.067	101,002.78	98,329.91	100,866.41	94,323.14	100,977.78
Encog	(Size 5) 7x24	100604.16 (80%)	NA	NA	NA	NA	NA	NA	NA
Tensorflow	(Size 5) 7x24	100604.16 (80%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 6) 8x24	114557.75 (81%)	190	00:00:02.558	115,049.26	112,184.71	114,913.36	108,202.48	115,049.26
Encog	(Size 6) 8x24	114557.75 (81%)	NA	NA	NA	NA	NA	NA	NA
Tensorflow	(Size 6) 8x24	114557.75 (81%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 7) 9x24	134897.69 (82%)	562.33	00:00:08.791	135,183.72	131,055.34	135,108.99	122,722.80	135,183.72
Encog	(Size 7) 9x24	134897.69 (82%)	NA	NA	NA	NA	NA	NA	NA
Tensorflow	(Size 7) 9x24	134897.69 (82%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 8) 10x24	145890.84 (83%)	789.33	00:00:15.043	146,182.86	142,052.75	146,137.50	132,810.78	146,182.86
Encog	(Size 8) 10x24	145890.84 (83%)	NA	NA	NA	NA	NA	NA	NA
Tensorflow	(Size 8) 10x24	145890.84 (83%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 9) 11x24	168513.06 (84%)	1029.333333	00:00:24.179	168,944.15	164,316.31	168,787.26	153,201.95	168,877.57
Encog	(Size 9) 11x24	168513.06 (84%)	NA	NA	NA	NA	NA	NA	NA
Tensorflow	(Size 9) 11x24	168513.06 (84%)	NA	NA	NA	NA	NA	NA	NA
RLM	(Size 10) 12x24	190509.79 (85%)	2970	00:01:16.346	190,838.39	185,758.72	190,753.13	167,298.16	190,838.39
Encog	(Size 10) 12x24	190509.79 (85%)	NA	NA	NA	NA	NA	NA	NA
Tensorflow	(Size 10) 12x24	190509.79 (85%)	NA	NA	NA	NA	NA	NA	NA

Appendix B

For the purpose of full disclosure for each of the three engine implementations, the source code for each system is included below. This will give detail on the configuration and operation for each case as well as insight into the divergent manner in which each is implemented.

RLM Source Code Implementation

```
int MAX_ITEMS = RPOCSimulationSettings.MAX_ITEMS; //10 Facings max of any single item
const int START_RANDOMNESS = 2;
const int END_RANDOMNESS = 0;
const int DEFAULT_SESSIONS_PER_BATCH = 800;
const int PREDICT_SESSIONS = 10;
const bool ENABLE_RLM_OUTPUT_LIMITER = true;

private RlmNetwork network;
private Item[] items;
private RPOCSimulationSettings simSettings;
private List<int> currentItemIndexes;

private List<double> metricScoreHistory = new List<double>();
private Queue<double> metricAvgLastTen = new Queue<double>();
public int totalSessions = 0;
private SimulationCsvLogger logger;
private int numScoreHits = 0;
private RLM.Enums.RlmInputType inputType;

public PlanogramOptimizer(Item[] items, RPOCSimulationSettings simSettings,
                                      UpdateUICallback updateUI = null,
                                      UpdateStatusCallback updateStatus = null,
                                      SimulationCsvLogger logger = null,
                                      string dbIdentifier = null)

{
    this.logger = logger;
    this.items = items.ToArray();
    this.simSettings = simSettings;
    UpdateUI = updateUI;
    UpdateStatus = updateStatus;
    if (ENABLE_RLM_OUTPUT_LIMITER)
    {
        currentItemIndexes = new List<int>();
    }


    UpdateStatus?.Invoke("Initializing...");
       network = new RlmNetwork(dbIdentifier != null ?
       dbIdentifier : "RLM_planogram_" + Guid.NewGuid().ToString("N"));



    inputType = RLM.Enums.RlmInputType.Distinct;
    if (!network.LoadNetwork("planogram"))
    {
        string int32Type = typeof(Int32).ToString();


        var inputs = new List<RlmIO>();
       inputs.Add(new RlmIO() { Name = "Slot", DotNetType = int32Type, Min = 1, Max = simSetting
                       s.NumSlots * simSettings.NumShelves, Type = inputType });

        var outputs = new List<RlmIO>();
        outputs.Add(new RlmIO() { Name = "Item", DotNetType = int32Type, Min = 0, Max = this.item
                       s.Length - 1 });

        // creates the network
        network.NewNetwork("planogram", inputs, outputs);
    }
}

private int GetEquivalentIndex(int index)
{
    var item = currentItemIndexes.ElementAt(index);
    return item;
}

private void ResetCurrentItemIndexes()
{
    if (!ENABLE_RLM_OUTPUT_LIMITER) return;

    currentItemIndexes.Clear();
    for (int i = 0; i < items.Length; i++)
    {
        currentItemIndexes.Add(i);
    }
}

public PlanogramOptResults StartOptimization(CancellationToken? cancelToken = null)
{
    UpdateStatus?.Invoke("Training...");

    var retVal = new PlanogramOptResults();

    // checks what type of simulation we are running
    int sessions = simSettings.SimType == SimulationType.Sessions ? simSettings.Sessions.Value :
                                                             DEFAULT_SESSIONS_PER_BATCH;
    double hours = simSettings.SimType == SimulationType.Time ? simSettings.Hours.Value : 0;
    simSettings.StartedOn = DateTime.Now;
    simSettings.EndsOn = DateTime.Now.AddHours(hours);

    do
    {
        // settings
        network.NumSessions = sessions;

        network.StartRandomness = START_RANDOMNESS;
        network.EndRandomness = END_RANDOMNESS;
       network.ResetRandomizationCounter();


        // training, train 90% per session batch if not Session Type
        var trainingTimes = (simSettings.SimType == SimulationType.Sessions) ? sessions :
                                                            sessions - PREDICT_SESSIONS;
        retVal = Optimize(trainingTimes, true, simSettings.EnableSimDisplay);

        if (cancelToken.HasValue && cancelToken.Value.IsCancellationRequested)
            return retVal;

        // for non Sessions type, we predict {PREDICT_SESSIONS}-times per session batch
        if (simSettings.SimType != SimulationType.Sessions)
        {
            int predictTimes = PREDICT_SESSIONS;
            if (simSettings.SimType == SimulationType.Score && predictTimes <
                                             RPOCSimulationSettings.NUM_SCORE_HITS)
               {
                predictTimes = RPOCSimulationSettings.NUM_SCORE_HITS;
               }
            retVal = Optimize(predictTimes, false, simSettings.EnableSimDisplay);
        }

    } while ((simSettings.SimType == SimulationType.Time && simSettings.EndsOn > DateTime.Now)
               ||(simSettings.SimType == SimulationType.Score &&
                       RPOCSimulationSettings.NUM_SCORE_HITS > numScoreHits));

    retVal = Optimize(1, false, true);

    network.TrainingDone();

    UpdateStatus?.Invoke("Done", true);

    return retVal;
}

private bool CheckDuplicateItem(IDictionary<int, int> itemDict, int itemIndex,
                                      int numFacings = 1)
{
    bool retVal = false;

    if (MAX_ITEMS < 0)
    {
        retVal = true;
    }
    else
    {
        if (itemDict.ContainsKey(itemIndex))
        {
            int count = itemDict[itemIndex];
            count += numFacings;
            if (count <= MAX_ITEMS)
            {
                itemDict[itemIndex] = count;
                retVal = true;
            }
        }
        else
        {
            // update the current item indexes for the rlm output limiter
            currentItemIndexes?.Remove(itemIndex);

            itemDict.Add(itemIndex, numFacings);
            retVal = true;
        }
    }

    return retVal;
}

private PlanogramOptResults Optimize(int sessions, bool learn = true,
               bool enablePlanogramDisplay = false, CancellationToken? cancelToken = null)
{
    var output = new PlanogramOptResultsSettings();     var hashedSku = new HashSet<int>();
    var itemDict = new Dictionary<int, int>();

    var shelves = new List<Shelf>();
    double totalMetricScore = 0;

    for (int i = 0; i < sessions; i++)
    {
        // reset for next session
        shelves.Clear();
        hashedSku.Clear();
        itemDict.Clear();
        ResetCurrentItemIndexes();
        totalMetricScore = 0;
        totalSessions++;
        DateTime startSession = DateTime.Now;

        long sessionId = network.SessionStart();

        int numSlotFlattened = 0;
        // iterates for how many number of shelves our planogram has
        for (int shelf = 1; shelf <= simSettings.NumShelves; shelf++)
        {
            // this shelf instance will hold the items the RLM will output
            var shelfInstance = new Shelf() { Number = shelf };
            int itemIndex;
            int numFacings;            for (int slot = 1; slot <= simSettings.NumSlots; slot += n
umFacings)
            {
                itemIndex = -1;
                numFacings = -1;
                bool isValid = false;
                numSlotFlattened++;
                do
                {
                    // create the inputs with their corresponding values
                    var inputs = new List<RlmIOWithValue>();

                    inputs.Add(new RlmIOWithValue(network.Inputs.First(a => a.Name == "Slot"),
                                      numSlotFlattened.ToString()));

                    var rlmItemOutput = network.Outputs.FirstOrDefault();
                    var rlmIdeas = new List<RlmIdea>()
                    {
                        new RlmOutputLimiter(rlmItemOutput.ID, currentItemIndexes.Count - 1,
                                                     GetEquivalentIndex)
                    };

                    // runs a cycle with the sessionId passed in
                    var cycle = new RlmCycle();
                    var rlmOutput = cycle.RunCycle(network, sessionId, inputs, learn, ideas:
                                      rlmIdeas);

                    // get the outputs

                    itemIndex = Convert.ToInt32(rlmOutput.CycleOutput.Outputs
                              .First(a => a.Name == "Item").Value);
                    numFacings = 1;
                    int remainingSlots = simSettings.NumSlots - shelfInstance.Items.Count();
                    bool isWithinLimit = CheckDuplicateItem(itemDict, itemIndex, numFacings);

                    if (isWithinLimit && remainingSlots >= numFacings)
                    {
                        isValid = true;
                        Item itemReference = items[itemIndex];
                        double itemMetricScore = PlanogramOptimizer.GetCalculatedWeightedMetrics(
                                             itemReference, simSettings);

                        totalMetricScore += (itemMetricScore * numFacings);

                        // add the items to the shelf container depending on how many facings
                        for (int n = 0; n < numFacings; n++)
                        {
                            shelfInstance.Add(itemReference, itemMetricScore);
                        }

                        // A non-duplicate is good.
                        network.ScoreCycle(rlmOutput.CycleOutput.CycleID, 1);
                    }
                    else
                    {

                        isValid = false;
                        network.ScoreCycle(rlmOutput.CycleOutput.CycleID, -1);
                        //System.Diagnostics.Debug.WriteLine("try again");
                    }
                } while (!isValid
            }

            shelves.Add(shelfInstance);
        }
        // ends the session with the summed metric score for all items in the planogram
        network.SessionEnd(totalMetricScore);
        System.Diagnostics.Debug.WriteLine($"Session #{i}, Score: {totalMetricScore}");

        // set statistics and the optimized planogram shelves (and items inside them)
        output.Shelves = shelves;
        metricScoreHistory.Add(totalMetricScore);
        metricAvgLastTen.Enqueue(totalMetricScore);
        if (metricAvgLastTen.Count > 10)
        {
            metricAvgLastTen.Dequeue();
        }
        output.Score = totalMetricScore;
        output.AvgScore = metricScoreHistory.Average();
        output.AvgLastTen = metricAvgLastTen.Average();
        output.MinScore = metricScoreHistory.Min();
        output.MaxScore = metricScoreHistory.Max();
        output.TimeElapsed = DateTime.Now - simSettings.StartedOn;
        output.CurrentSession = totalSessions;
        output.MaxItems = MAX_ITEMS;
        output.StartRandomness = network.StartRandomness;
        output.EndRandomness = network.EndRandomness;
        output.SessionsPerBatch = DEFAULT_SESSIONS_PER_BATCH;
        output.InputType = inputType.ToString();
        output.CurrentRandomnessValue = network.RandomnessCurrentValue;

        if (logger != null)
        {
            SimulationData logdata = new SimulationData();

            logdata.Session = output.CurrentSession;
            logdata.Score = output.Score;
            logdata.Elapse = DateTime.Now - startSession;

            logger.Add(logdata);
        }

        // update the numScoreHits if the sim type is Score
        if (simSettings.SimType == SimulationType.Score)
        {
            if (totalMetricScore >= simSettings.Score.Value)
            {
                numScoreHits++;
                output.NumScoreHits = numScoreHits;
            }
            else
            {
                numScoreHits = 0;
            }
        }

        //OnSessionDone?.Invoke(output);

        // updates the results to the UI
        //bool enableSimDisplay = (!learn) ? true : (learn && simSettings.EnableSimDisplay) ? tru
e : false;
        if (enablePlanogramDisplay)
        {
            output.MetricMin = simSettings.ItemMetricMin;
            output.MetricMax = simSettings.ItemMetricMax;
            output.CalculateItemColorIntensity();
        }
        UpdateUI?.Invoke(output, enablePlanogramDisplay);

        // checks if we have already by passed the time or score that was set
        // if we did, then we stop the training and end it abruptly
        if ((simSettings.SimType == SimulationType.Time && simSettings.EndsOn.Value <=
               DateTime.Now) ||(simSettings.SimType == SimulationType.Score &&
                       numScoreHits >= RPOCSimulationSettings.NUM_SCORE_HITS))
            break;
        if (cancelToken.HasValue && cancelToken.Value.IsCancellationRequested)
            return output;
    }

    return output;
}

public static double GetCalculatedWeightedMetrics(Item item, RPOCSimulationSettings simSettings)
{
    double retVal = 0;
    double[] metrics = GetCalculatedWeightedMetricArray(item, simSettings);

    retVal = metrics.Sum();

    return retVal;
}

public static double[] GetCalculatedWeightedMetricArray(Item item, RPOCSimulationSettings simSett
ings)
{
    double[] retVal = new double[10];

    // we go through each attribute for the item and sum up each of its metric
    foreach (var attr in item.Attributes)
    {
        retVal[0] += attr.Metric1;
        retVal[1] += attr.Metric2;
        retVal[2] += attr.Metric3;
        retVal[3] += attr.Metric4;
        retVal[4] += attr.Metric5;
        retVal[5] += attr.Metric6;
        retVal[6] += attr.Metric7;
        retVal[7] += attr.Metric8;
        retVal[8] += attr.Metric9;
        retVal[9] += attr.Metric10;
    }

    retVal[0] = simSettings.Metric1 == 0 ? 0 : retVal[0] * (simSettings.Metric1 / 100D);
    retVal[1] = simSettings.Metric2 == 0 ? 0 : retVal[1] * (simSettings.Metric2 / 100D);
    retVal[2] = simSettings.Metric3 == 0 ? 0 : retVal[2] * (simSettings.Metric3 / 100D);
    retVal[3] = simSettings.Metric4 == 0 ? 0 : retVal[3] * (simSettings.Metric4 / 100D);
    retVal[4] = simSettings.Metric5 == 0 ? 0 : retVal[4] * (simSettings.Metric5 / 100D);
    retVal[5] = simSettings.Metric6 == 0 ? 0 : retVal[5] * (simSettings.Metric6 / 100D);
    retVal[6] = simSettings.Metric7 == 0 ? 0 : retVal[6] * (simSettings.Metric7 / 100D);
    retVal[7] = simSettings.Metric8 == 0 ? 0 : retVal[7] * (simSettings.Metric8 / 100D);
    retVal[8] = simSettings.Metric9 == 0 ? 0 : retVal[8] * (simSettings.Metric9 / 100D);
    retVal[9] = simSettings.Metric10 == 0 ? 0 : retVal[9] * (simSettings.Metric10 / 100D);

    return retVal;
}
```

TensorFlow Python Source Code

```
    def __init__(self, items, simSettings, token):
        self.items = items
        self.simSettings = simSettings
        self.cancelToken = token
        self.numSlots = simSettings.NumShelves * simSettings.NumSlots
        self.numOutputs = len(self.items)
        self.simType = self.simSettings.SimType
        self.totalSeconds = (3600 * self.simSettings.Hours + time.time())

        if (self.simSettings.SimType == 1) else 0
        self.randomReset = 100

    def normalize(self, originalStart, originalEnd, newStart, newEnd, value):
        scale = (newEnd - newStart) / (originalEnd - originalStart);
        return (newStart + ((value - originalStart) * scale));

    def endTraining(self, criteria):
        if (self.cancelToken.IsCancellationRequested):
            return True;

        if (self.simType == 0):
            if (criteria >= self.simSettings.Sessions):
                return True
        elif (self.simType == 1):
            if (criteria >= self.totalSeconds):
                return True
        else:
            if (criteria >= SimulationSettings.NUM_SCORE_HITS):
                return True
        return False

    def train(self, lr, ui_results_callback, ui_status_callback, ui_logger, tf_settings):

        lr = 0.001

        NUM_HIDDEN_LAYERS = 3
        HIDDEN_LAYER_NEURONS = 500
        TF_ACTIVATION = tf.nn.softmax
        TF_OPTIMIZER = tf.train.AdamOptimizer(learning_rate=lr)

        n_nodes_hl1 = HIDDEN_LAYER_NEURONS
        n_nodes_hl2 = HIDDEN_LAYER_NEURONS
        n_nodes_hl3 = HIDDEN_LAYER_NEURONS

        ui_status_callback("Initializing...")

        tf.reset_default_graph() #Clear the Tensorflow graph.

        #These lines established the feed-
        forward part of the network. The agent takes a state and produces an action.
        self.slot_in = tf.placeholder(shape=[1],dtype=tf.int32)
        slot_in_OH = slim.one_hot_encoding(self.slot_in, self.numSlots)

        hidden_1_layer = {'weights':tf.Variable(tf.random_normal([self.numSlots, n_nodes_hl1]),
                       name="hl_w_1"),
                      'biases':tf.Variable(tf.ones([n_nodes_hl1]), name="hl_b_1")}

        hidden_2_layer = {'weights':tf.Variable(tf.random_normal([n_nodes_hl1, n_nodes_hl2]),
                       name="hl_w_2"),
                          'biases':tf.Variable(tf.ones([n_nodes_hl2]), name="hl_b_2")}

        hidden_3_layer = {'weights':tf.Variable(tf.random_normal([n_nodes_hl3, self.numSlots]),
                       name="hl_w_3"),
                          'biases':tf.Variable(tf.ones([self.numSlots]), name="hl_b_3")}

        output_layer = {'weights':tf.Variable(tf.random_normal([self.numSlots, self.numOutputs]),
                        name="out_w"),
                        'biases':tf.Variable(tf.ones([self.numSlots, self.numOutputs]),
                              name="out_b"),}


        l1 = tf.add(tf.matmul(slot_in_OH,hidden_1_layer['weights']), hidden_1_layer['biases'])
        l1 = TF_ACTIVATION(l1)

        l2 = tf.add(tf.matmul(l1,hidden_2_layer['weights']), hidden_2_layer['biases'])
        l2 = TF_ACTIVATION(l2)

        l3 = tf.add(tf.matmul(l2,hidden_3_layer['weights']), hidden_3_layer['biases'])
        l3 = TF_ACTIVATION(l3)

        output = tf.matmul(l3,output_layer['weights']) + output_layer['biases']

        #self.output = tf.reshape(output,[-1])

        #The next six lines establish the training proceedure. We feed the reward and chosen
        #action into the network
        #to compute the loss, and use it to update the network.
        self.reward_holder = tf.placeholder(shape=[1],dtype=tf.float32)
        self.action_holder = tf.placeholder(shape=[1],dtype=tf.int32)
        self.chosen_action = tf.argmax(output[self.slot_in[0]], 0)
        self.responsible_weight = tf.slice(output, [self.slot_in[0],self.action_holder[0]],[1,1])
        self.loss = -(tf.log(self.responsible_weight[0][0])*self.reward_holder)
        optimizer = TF_OPTIMIZER#tf.train.AdamOptimizer(learning_rate=lr)
        self.update = optimizer.minimize(self.loss)
        weights = tf.trainable_variables()[6]
        #The weights we will evaluate to look into the network.

        #get tensorflow current settings and pass to UI
        optimizerName = optimizer.__class__.__name__
        activationName = TF_ACTIVATION.__name__
        tf_settings(optimizerName, activationName, NUM_HIDDEN_LAYERS, HIDDEN_LAYER_NEURONS, lr)

        print(tf.trainable_variables())
        totalMetricScoreArr = []
        avgMetricScoreLastTen = []
        flag = 0
        startTime = time.time()
        sessionCounter = 0
        init = tf.initialize_all_variables()
        randomness = 1

        ui_status_callback("Training...")

        trainingNum = -1
        predictNum = 0
        doPredict = False
        interval = 1 / (self.randomReset - SimulationSettings.NUM_SCORE_HITS)

        # Launch the tensorflow graph
        with tf.Session() as sess:
            sess.run(init)
            highestPossibleSessionScore = self.simSettings.ItemMetricMax * self.numSlots
            lowestPossibleSessionScore = self.simSettings.ItemMetricMin * self.numSlots

            matrix = {};
            while (not self.endTraining(flag)):
                totalMetricScore = 0
                results = PlanogramOptResults()
                sessionStartTime = time.time()
                actionsPerSlot = {}
                hasDupItems = False

                if (not doPredict):
                    trainingNum += 1
                    if (trainingNum != 0 and trainingNum % (self.randomReset -
                                      SimulationSettings.NUM_SCORE_HITS) == 0):
                        doPredict = True
                    else:
                        randomness -= interval;

                if (doPredict):
                    predictNum += 1
                    randomness = 0
                    if (predictNum > SimulationSettings.NUM_SCORE_HITS):
                        doPredict = False
                        predictNum = 0
                        randomness = 1
                        trainingNum = 0

                #Reset facings list
                facingsList = {}
                for it in self.items:
                    facingsList[it.ID] = 0

                #Train
                hasDupItems = False
                action = 0
                for sl in range(self.numSlots):
                    #get an item randomly
                    if (np.random.rand(1) <= randomness):
                        action = np.random.randint(self.numOutputs)
                    else:
                        action = sess.run(self.chosen_action, feed_dict={self.slot_in:[sl]})

                    actionsPerSlot[sl] = action

                    #Get item reference and calculate metric score
                    item = self.items[action]
                    metricScore = PlanogramOptimizer.GetCalculatedWeightedMetrics(
                                             item, self.simSettings)

                    dupItemVal = facingsList[item.ID] #Get item number of facings

                    if(not hasDupItems and dupItemVal < 10):
                        reward = 1
                        dupItemVal = dupItemVal + 1 #Increment item number of facings
                        facingsList[item.ID] = dupItemVal
                       #Update item number of facings in the list
                        totalMetricScore += metricScore
                    else:
                        totalMetricScore = 0
                        reward = -1
                        hasDupItems = True

                #Update network
                #print (totalMetricScore)
                if (hasDupItems):
                    totalMetricScore = 0
                    sessionScoreNormalized = -1
                else:
                    sessionScoreNormalized = self.normalize(lowestPossibleSessionScore,
                              highestPossibleSessionScore, -1, 1, totalMetricScore)

                if (not doPredict or (doPredict and predictNum <= 1)):
                    for sl in range(self.numSlots):
                        feed_dict={self.reward_holder:[sessionScoreNormalized],
                              self.action_holder:[actionsPerSlot[sl]],self.slot_in:[sl]}
                        loss,update,matrix = sess.run([self.loss,self.update,weights],
                              feed_dict=feed_dict)

                #Reset facings list
                facingsList = {}
                for it in self.items:
                    facingsList[it.ID] = 0
```

```
totalMetricScore = 0
#Extract updated results from matrix
shelfItems = []
for slotIndex in range(self.numSlots):

    slotArr = matrix[slotIndex]
    itemIndex = np.argmax(slotArr)
    item = self.items[itemIndex]
    shelfItems.append(item)
    #actionsPerSlot[slotIndex] = itemIndex

    dupItemVal = facingsList[item.ID] #Get item number of facings

    if(dupItemVal < 10):
        dupItemVal = dupItemVal + 1 #Increment item number of facings
        facingsList[item.ID] = dupItemVal

       #Update item number of facings in the list
    else:
        hasDupItems = True

    if ((slotIndex+1) % 24 == 0):
        shelf = Shelf()
        for ii in range(len(shelfItems)):
            metricScore = PlanogramOptimizer.GetCalculatedWeightedMetrics
                      (shelfItems[ii], self.simSettings)
            totalMetricScore += metricScore
            shelf.Add(shelfItems[ii], metricScore)
        shelfItems = []
        results.Shelves.Add(shelf)

if (hasDupItems):
    totalMetricScore = 0

sessionCounter+=1
currentTime = time.time()

#Setup planogram results and stats
totalMetricScoreArr.append(totalMetricScore)
avgMetricScoreLastTen.append(totalMetricScore)
if (len(avgMetricScoreLastTen) > 10):
    avgMetricScoreLastTen.pop(0)

results.Score = totalMetricScore
results.AvgScore = np.average(totalMetricScoreArr)
results.AvgLastTen = np.average(avgMetricScoreLastTen)
results.MinScore = np.amin(totalMetricScoreArr)
results.MaxScore = np.amax(totalMetricScoreArr)
results.TimeElapsed = TimeSpan.FromSeconds(float(currentTime - startTime))
results.CurrentSession = sessionCounter

#Update the training flag
if (self.simType == 0):
    flag = sessionCounter
elif (self.simType == 1):
    flag = currentTime
else:
    if (totalMetricScore >= self.simSettings.Score):
        flag+=1
        results.NumScoreHits = flag
    else:
        flag = 0

ui_logger(sessionCounter, totalMetricScore, float(currentTime-sessionStartTime))

#Pass results to ui
if (self.simSettings.EnableSimDisplay):
    results.MetricMin = self.simSettings.ItemMetricMin
    results.MetricMax = self.simSettings.ItemMetricMax
    results.CalculateItemColorIntensity()
```

```
            ui_results_callback(results, self.simSettings.EnableSimDisplay)
        ui_results_callback(results, True)
```

Encog C# Source Code

```
public class PlanogramOptimizerEncog
{
    //private Item[] items;
    private RPOCSimulationSettings simSettings;
    private SimulationCsvLogger logger;

    private BasicNetwork network;
    private IMLTrain train;
    private PlanogramScore planogramScore;

    private UpdateUINonRLMCallback updateUI;
    private UpdateStatusCallback updateStatus;


    const int POPULATION_SIZE = 200;

    // 10-24 set anneal to default false
    public PlanogramOptimizerEncog(Item[] items, RPOCSimulationSettings simSettings,
                          UpdateUINonRLMCallback updateUI = null,
                          UpdateStatusCallback updateStatus = null,
                          SimulationCsvLogger logger = null, bool anneal = false)
    {
        updateStatus?.Invoke("Initializing...");

        //this.items = items;
        this.simSettings = simSettings;
        this.logger = logger;
        this.updateUI = updateUI;
        this.updateStatus = updateStatus;

        network = CreateNetwork();
        planogramScore = new PlanogramScore()
        {
            SimSettings = simSettings,
            Items = items,
            UpdateUI = updateUI,
            Logger = logger
        };

        if (anneal)
        {
            train = new NeuralSimulatedAnnealing(network, planogramScore, 18, 2, 50);

        }
        else
        {
            train = new MLMethodGeneticAlgorithm(() => {
                ((IMLResettable)network).Reset();
                return network;
            }, planogramScore, POPULATION_SIZE);
        }
    }

    public void StartOptimization(CancellationToken? cancelToken = null)
    {
        updateStatus?.Invoke("Training...");

        double hours = simSettings.SimType == SimulationType.Time ?simSettings.Hours.Value:0;
        simSettings.StartedOn = DateTime.Now;
        simSettings.EndsOn = DateTime.Now.AddHours(hours);

        do
        {
            train.Iteration();

            if (cancelToken.HasValue && cancelToken.Value.IsCancellationRequested)
                break;
        } while ((simSettings.SimType == SimulationType.Sessions &&
                  simSettings.Sessions > planogramScore.SessionNumber) ||
                   (simSettings.SimType == SimulationType.Time && simSettings.EndsOn
                  > DateTime.Now) ||
                   (simSettings.SimType == SimulationType.Score &&
                  RPOCSimulationSettings.NUM_SCORE_HITS >
                  planogramScore.NumScoreHits));

        // display for final results only
        if (!simSettings.EnableSimDisplay)
        {
            updateUI?.Invoke(planogramScore.LastResult, true);
        }

        updateStatus?.Invoke("Done", true);
    }

    private BasicNetwork CreateNetwork()
    {
        var pattern = new FeedForwardPattern { InputNeurons = 1 };

        for (int i = 0; i < simSettings.HiddenLayers; i++)
        {
            pattern.AddHiddenLayer(simSettings.HiddenLayerNeurons);
        }

        pattern.OutputNeurons = 1;
        pattern.ActivationFunction = new ActivationElliottSymmetric();
        //pattern.ActivationFunction = new ActivationTANH();
        var network = (BasicNetwork)pattern.Generate();
        network.Reset();
        return network;
    }
}

public class PlanogramScore : ICalculateScore
{
    private ConcurrentBag<double> metricScoreHistory = new ConcurrentBag<double>();
    private ConcurrentQueue<double> metricAvgLastTen = new ConcurrentQueue<double>();

    public int SessionNumber { get; set; } = 0;
    public int NumScoreHits { get; set; } = 0;
    public RPOCSimulationSettings SimSettings { get; set; }
    public Item[] Items { get; set; }
    public UpdateUINonRLMCallback UpdateUI { get; set; }
    public SimulationCsvLogger Logger { get; set; }
    public PlanogramOptResults LastResult { get; set; }

    public double CalculateScore(IMLMethod network)
    {
        SessionNumber++;

        DateTime startSession = DateTime.Now;

        PlanogramSimulation pilot = new PlanogramSimulation(
                                         (BasicNetwork)network, Items, SimSettings);
        var result = LastResult = pilot.ScorePilot();

        result.CurrentSession = SessionNumber;
        metricScoreHistory.Add(result.Score);
        metricAvgLastTen.Enqueue(result.Score);
        if (metricAvgLastTen.Count > 10)
        {
            double outVal = 0;
            metricAvgLastTen.TryDequeue(out outVal);
        }
        result.MaxScore = metricScoreHistory.Max();
        result.MinScore = metricScoreHistory.Min();
        result.AvgScore = metricScoreHistory.Average();
```

```
            result.AvgLastTen = metricAvgLastTen.Average();

            if (SimSettings.Score.HasValue)
            {
                if (result.Score >= SimSettings.Score.Value)
                {
                    NumScoreHits++;
                }
                else
                {
                    NumScoreHits = 0;
                }
            }

            result.NumScoreHits = NumScoreHits;

            if (Logger != null)
            {
                SimulationData logdata = new SimulationData();

                logdata.Session = result.CurrentSession;
                logdata.Score = result.Score;
                logdata.Elapse = DateTime.Now - startSession;

                Logger.Add(logdata, false);
            }

            UpdateUI?.Invoke(result, SimSettings.EnableSimDisplay);

            return result.Score;
        }


        public bool ShouldMinimize { get { return false; ; } }

        public bool RequireSingleThreaded { get { return true; } }
    }

    public class PlanogramSimulation
    {
        const int MAX_ITEMS = 10;

        private readonly NormalizedField slotNormalizer;
        private readonly NormalizedField itemNormalizer;
        private readonly BasicNetwork network;
        private readonly RPOCSimulationSettings simSettings;
        private readonly Item[] items;

        public static int CycleCount { get; set; }

        public PlanogramSimulation(BasicNetwork network, Item[] items,
                                      RPOCSimulationSettings simSettings)
        {
            slotNormalizer = new NormalizedField(NormalizationAction.Normalize, "Slot",
                       (simSettings.NumSlots * simSettings.NumShelves) - 1, 0, -1, 1);
            itemNormalizer = new NormalizedField(NormalizationAction.Normalize, "Item",
                       items.Length - 1, 0, -1, 1);

            this.items = items;
            this.simSettings = simSettings;
            this.network = network;
        }

        private bool CheckDuplicateItem(IDictionary<int, int> itemDict, int itemIndex,
                                             int numFacings = 1)
        {
            bool retVal = false;

            if (MAX_ITEMS < 0)
            {
                retVal = true;
            }
            else
            {
                if (itemDict.ContainsKey(itemIndex))
                {
                    int count = itemDict[itemIndex];
                    count += numFacings;
                    if (count <= MAX_ITEMS)
                    {
                        itemDict[itemIndex] = count;
                        retVal = true;
                    }
                }
                else
                {
                    itemDict.Add(itemIndex, numFacings);
                    retVal = true;
                }
            }

            return retVal;
        }

        public PlanogramOptResults ScorePilot()
        {
            PlanogramOptResults retVal = new PlanogramOptResults();

            var shelves = new List<Shelf>();
            double totalMetricScore = 0;
            int slotNumber = 0;
            Dictionary<int, int> itemDict = new Dictionary<int, int>();
            bool hasExceedMax = false;

            for (int i = 0; i < simSettings.NumShelves; i++)
            {
                Shelf shelfInstance = new Shelf();
                for (int p = 0; p < simSettings.NumSlots; p++)
                {
                    var inputs = new BasicMLData(1);
                    inputs[0] = slotNormalizer.Normalize(slotNumber);

                    ////Actual Code
                    //IMLData output = network.Compute(inputs);
                    //int index = Convert.ToInt32(itemNormalizer.DeNormalize(output[0]));

                    int index = 1;
                    IMLData output = null;
                    //Test Code

                    do
                    {
                        output = network.Compute(inputs);
                        index = Convert.ToInt32(itemNormalizer.DeNormalize(output[0]));

                    } while (index < 0 || index > items.Length - 1);



                    //// Gino Code for Genetic
                    //if (index < 0 || index > items.Length - 1)
                    //{
                    //    index = 0;
                    //}

                    Item itemReference = items[index];

                    if (!hasExceedMax)
                    {
                        hasExceedMax = !(CheckDuplicateItem(itemDict, index));
                    }

                    double itemMetricScore = PlanogramOptimizer.GetCalculatedWeightedMetrics(
                                                            itemReference, simSettings);

                    totalMetricScore += itemMetricScore;

                    shelfInstance.Add(itemReference, itemMetricScore);

                    slotNumber++;
                }

                shelves.Add(shelfInstance);
            }

            retVal.Shelves = shelves;
            retVal.Score = (hasExceedMax) ? 0 : totalMetricScore;
            retVal.TimeElapsed = DateTime.Now - simSettings.StartedOn;

            return retVal;
        }
    }
```