ASTUTEPEAK SOLUTION: ARTIFICIAL NEURAL NETWORK WITH PARTICLE SWARM OPTIMIZATION FOR A MATURE ENTERPRISE-LEVEL WEB EXPERIENCE MANAGEMENT SYSTEM’S SHARED-RESOURCES ENVIRONMENT

A thesis submitted in partial fulfillment of the requirements

For the degree of Masters of Science in Software Engineering

By Asya Kirgiz

May 2015
The thesis of Asya Kirgiz is approved:

________________________________________  ____________
Dr. Patrick Desbrow  
Date

________________________________________  ____________
Dr. Vahab Pournaghshband  
Date

________________________________________  ____________
Dr. Richard Covington, Chair  
Date

California State University, Northridge
Acknowledgement

I would like to thank the engineering and development team at CrownPeak Technology for their help and support. Especially, I would like to thank my committee member, Chief Information Officer and Vice President of Engineering at CrownPeak Technology Patrick Desbrow, Ed.D for his unparalleled input into this thesis project.

I also want to thank the faculty of Computer Science department at California State University Northridge for enabling me to expand my technical horizons and obtain the knowledge base and creative vision needed to make this project. I am particularly thankful to my committee chair and Computer Science Department Chair Dr. Rick Covington.
Table of Contents

Copyright Page..............................................................................................................ii
Signature Page .................................................................................................................iii
Acknowledgement .............................................................................................................iv
Abbreviations .................................................................................................................. vi
Table of Figures ..............................................................................................................vii
Abstract ..........................................................................................................................viii

1 INTRODUCTION ........................................................................................................... 1

2 PROBLEM DEFINITION AND LIMITATIONS ............................................................. 3
  2.1 Crown Peak Technologies – Company Overview ..................................................... 3
  2.2 WEM System Slowdown ......................................................................................... 4
  2.3 Leaky Bucket Solution ......................................................................................... 5
  2.4 Goals of AstutePeak ............................................................................................. 6

3 ARTIFICIAL NEURAL NETWORK OVERVIEW ......................................................... 7
  3.1 Feed-forward NN Basics ..................................................................................... 7
  3.2 Particle Swarm Optimization (PSO) Basics .......................................................... 8
  3.3 Example of NN with Particle Swarm Optimization .............................................. 10

4 ASTUTEPEAK DESIGN AND IMPLEMENTATION .................................................... 14
  4.1 Architecture ......................................................................................................... 14
  4.2 Feature Extraction .............................................................................................. 15
  4.3 Output Derivation ............................................................................................... 16
  4.4 Data Encoding .................................................................................................... 17
  4.5 Activation Functions ............................................................................................ 19
  4.6 PSO ....................................................................................................................... 20

5 ASTUTEPEAK RESULTS ............................................................................................ 21
  5.1 Experiment Iteration 1 ........................................................................................ 21
  5.2 Experiment Iteration 2 ....................................................................................... 22

6 ASTUTEPEAK CONCLUSIONS ................................................................................ 24
  6.1 Information Gathering Stage .............................................................................. 24
  6.2 Decision Making Process ................................................................................... 24
  6.3 Conclusions ......................................................................................................... 25
  6.4 Future Work ........................................................................................................ 26

BIBLIOGRAPHY ............................................................................................................. 27

APPENDIX A: Class Diagram ......................................................................................... 29
APPENDIX B: Code .......................................................................................................... 30
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>MSMQ</td>
<td>Microsoft Messaging Queue</td>
</tr>
<tr>
<td>PTO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>SAAS</td>
<td>Software As A Service</td>
</tr>
<tr>
<td>VB</td>
<td>Visual Basic</td>
</tr>
<tr>
<td>WEM</td>
<td>Web Experience Management</td>
</tr>
</tbody>
</table>
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Structure of CrownPeak's API</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Uncontrolled burst of API calls</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Controlled flow of API calls</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Leaky Bucket Algorithm</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Activation function in neural nets</td>
<td>7</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Laws of linear motion</td>
<td>8</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Calculation of (k+1) position vector</td>
<td>9</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Feed-forward NN</td>
<td>10</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Demo of first two iterations of 2 particle swarm</td>
<td>12</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Demo of error calculation for a position</td>
<td>13</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Message queueing architecture</td>
<td>14</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Inputs and outputs of AstutePeak</td>
<td>16</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Example of categorical value encoding</td>
<td>18</td>
</tr>
<tr>
<td>Figure 14</td>
<td>HyperTan activation function</td>
<td>19</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Softmax activation function</td>
<td>19</td>
</tr>
</tbody>
</table>
Abstract

ASTUTEPEAK SOLUTION: ARTIFICIAL NEURAL NETWORK WITH PARTICLE SWARM OPTIMIZATION FOR A MATURE ENTERPRISE-LEVEL WEB EXPERIENCE MANAGEMENT SYSTEM'S SHARED-RESOURCES ENVIRONMENT

By

Asya Kirgiz

Master of Software Engineering

AstutePeak is a machine learning solution created for an enterprise-level commercial software company named CrownPeak in the Web Experience Management (WEM) industry. This solution was designed and implemented to generate actionable predictions used to improve the processing performance of the company’s Content Management System (CMS).

The objective of this research and development project is to gain a deeper understanding and better control over the processing performance of custom written code-based template files used to create and manage website content. Software-based controls are currently in place to profile and limit the rate of resource requests made by templates.

The goal is to develop a neural network software component to discover patterns in template file code and to assist developers in the creation of high performing software.
CrownPeak Technologies provides cloud based Software as a Service (SaaS) Web Experience Management (WEM) platform for enterprise level web content hosting and web marketing enablement. The design and publishing processes of the system are governed by template files written using either the C# or Visual Basic (VB) programming language which calls CrownPeak API methods to perform common web management functions. These template files are linked to specific content assets at execution time to render web pages or websites.

This mature enterprise-level Web Experience Management (WEM) software platform is a shared multi-tenant environment used by hundreds of developers simultaneously and executing thousands of functions within each template [1]. A current software solution is in place to regulate the flow of resources requests. However, this solution lacks the precision of numerical distinction from one request to another. This results in fewer data point to perform the pattern analysis and ultimately determine the drivers of slow template performance.

The best method to collect the desired data metrics is from the CPU on each of the template processing servers. However, collecting run time information about CPU performance consumes significant processing resources and slows down the processes that are being profiled within the template.

The proposed solution is to create a neural network software component that uses a particle swarm optimization algorithm called the AstutePeak solution. The neural network component will collect data (a.k.a. learn) from only a few processing servers (a.k.a. instances) where a
decrease in runtime performance is tolerable. This will create a theoretical model that will approximate CPU metrics in order to achieve a better governance over the system’s run time.

The AstutePeak software component is a complex set of functions that uses several internal parameters. Each parameter is a mathematical value such as weights and biases and will be used to derive the predicted model algorithm. The model is populated as a part of a two-stage process.

The first stage is considered the training stage where mathematical values (a.k.a. data tuples) are send to the model. The component software will iteratively apply the Neural Network with Particle Swarm Optimization algorithm until the optimal set of values is determined with the desired accuracy of prediction.

Particle swarm optimization is utilized to determine the best parameters of the AstutePeak. This optimization is loosely modeled after the communal behavioral patterns where all particles share knowledge about and move toward the target value - the best set of network’s parameters.

The second stage is the testing or validation stage. A second set of similar mathematical values (a.k.a. data tuples) is submitted to the model. The results of the two stages are compared to determine if the model’s prediction remains consistent and similar to the actual CPU metrics captured at the beginning of the process.
2 PROBLEM DEFINITION AND LIMITATIONS

2.1 Crown Peak Technologies – Company Overview

CrownPeak Technologies offers a cloud based, software-as-a-service Web Experience Management and a hosting platform [2]. The company was founded in 2001. CrownPeak WEM is multi-tenant and platform agnostic. The system is compatible with open data source systems and provides comprehensive 3rd-party application integration and connectors. CrownPeak WEM features web analytics, site-search, and content versioning with rollback. More than 3,500 websites in over 75 languages are managed by CrownPeak. CrownPeak’s clients include Nissan, Skype, CoreLogic and Alico [3].

CrownPeak CMS API provides functions and classes that enable developers to create input forms for collecting client data, presenting the collected data on webpages and manipulating the publishing properties just to name a few. The CrownPeak CMS API is built on top of Microsoft .NET 4.0 software framework and supports development in both the C# and Visual Basic programming languages. Developers create software based templates and link them with content assets.

The CrownPeak CMS API consists of three layers:

![Figure 1 Structure of CrownPeak's API](image-url)

*Figure 1 Structure of CrownPeak's API*
Some examples of services that CrownPeak offers include authoring and collaborating on content, validating, staging and publishing assets as well as marketers-focused experience analysis.

Since client’s developers have full control over template files, the system is prone to be inefficient when a developer executes a poorly written template.

2.2 WEM System Slowdown

The system’s CPU can become overwhelmed by the intensity of service requests. When this occurs, the system experiences a significant slowdown and, in worst cases could fail. The proposed solution puts a software mechanism in place to control the flow of requests, and, therefore, prevent the CPU from being overwhelmed.

In the context of CrownPeak WEM, the most significant and the least handled cause for slowdown is inefficient CMS template files that make unnecessary or too many API calls.

Figure 2 Uncontrolled burst of API calls
This graph shows an uncontrolled burst of API calls. The spike in the graph illustrates the potential stress that can be placed on the system and the need for this issue to be addressed in some practical way.

The existing software-based process regulates the flow of requests by leveling them out over time, so that a burst of resource requests made by templates during run-time is controlled from within the process.

This control is achieved by applying regulation algorithms on the flow of API calls generated by any single entity thus ensuring that the number of requests will not go above a specified rate limit.

![Controlled flow of API calls](image)

*Figure 3 Controlled flow of API calls*

### 2.3 Leaky Bucket Solution

A Leaky Bucket buffer is an excellent solution for regulating the flow of resource requests by different users [4]. It limits excessive use of the key resources and provides the control to punish poorly performing templates while limiting a negative impact on the processing time of other users’ templates. CrownPeak’s Director of Engineering and Lead Architect Fares Noueihe designed and implemented an adaptation of the Leaky Bucket algorithm that controls throttling in between requests once the rate limit has been reached. Specifically, it prevents most infinite
recursion scenarios and enables rendering timeout instead of relying on IIS HTTP request timeout [5].

![Leaky Bucket Algorithm](image)

*Figure 4 Leaky Bucket Algorithm*

### 2.4 Goals of AstutePeak

Directly profiling performance of CPU activities is not a feasible approach since collecting real time CPU performance data consumes a significant amount of the processing power.

Alternatively a machine learning approach may provide the data to accomplish the goal. Once the model is developed, the CPU performance can be predicted by the AstutePeak solution based on metrics that are relatively straightforward to collect.

One of the immediate benefits of AstutePeak is the ability to calculate exactly how much “water” should be put into the bucket for each particular request. The so-called *Impact Modifier* determines the impact on CPU that each particular request exerts. This *impactModifier* is best approximated by a linear correlation to the CPU’s performance parameters [6].
3.1 Feed-forward NN Basics

Like most other predictive models, Artificial Neural Networks (ANN) algorithms accept a set of input data and then predicts output values [7]. Some components of a feed-forward Neural Networks (NN) are processing layers with nodes, constant weights and biases parameters, activation functions, and encoding/decoding techniques [8].

The most common NN has three layers: input layer, hidden layer and output layer. One can add more hidden layers to get a deep learning network. However, the studies have shown that the three layered NN can accomplish as much as any NN with more than three layers [9].

The data is entered into the nodes of the input layer.

Then the input vector with weights adjustments is summed over all input values. Next, an activation function takes these sum and calculates the values that are consequently entered into the nodes of the hidden layer. In a similar manner, another activation function then processes the values of the hidden layer and outputs the final values into the nodes of the output layer.

![Activation function in neural nets](image)

*Figure 5 Activation function in neural nets*
Once the model is formed, the NN takes in the input values and outputs its prediction for the target values. The model works by applying functions with constant parameters to the input in order to produce the output [10].

During the training phase, NN estimates the best parameters (weights and biases). During this process, we are using a set of data tuples that includes both input and target values. We hold these input and target values as constants and use an optimization technique to estimate the best possible weights and biases. Once the desired accuracy of prediction is achieved, the discovered weights and biases become constant parameters of our NN and we are ready to predict outputs for a new set of input data.

3.2 Particle Swarm Optimization (PSO) Basics

PSO is loosely modeled after behavior of a swarm of bees that flies toward a goal. The bees share knowledge about best estimation of the target position at each step [11]. Each bee, or a particle, is a single point in an n-dimensional space. PSO estimates the best solution by iteratively moving the particles toward the target point in an n-dimensional space. PSO uses the laws of linear motion to improve the positions, velocities and acceleration of the particles between iterations.

\[
\begin{align*}
    v_f &= v_i + at \\
    x_f &= x_i + v_i t + \frac{1}{2}at^2 \\
    v_f^2 &= v_i^2 + 2a(x_f - x_i) \\
    x_f &= x_i + \frac{1}{2}(v_i + v_f)t
\end{align*}
\]

Figure 6 Laws of linear motion

A best position represents a point with coordinate values that are the closest to the goal. All particles are trying to find the best point by moving toward it. In other words, with each iteration PSO finds an improved estimation of the desired position.
Finally, to compute the error, which is the discrepancy between any given position and the target position, PSO uses the Mean Squared Error function.

Below is an algorithm for PSO at a high level:

*Initialize each particle to random state (position, velocity, error, best-position, and best-error)*
*Save best position of ANY particle to global-best*
*Loop until done*
  *For each particle in swarm*
    *Compute new particle velocity (1)*
    *Use new velocity to compute new position (2)*
    *Compute error of new position*
    *If new error better then best-error*
      *Best-position = new position*
    *If new error better than global-best*
      *Global-best = new position*
  *End For*
*End Loop*
*Return global-best position*
3.3 Example of NN with Particle Swarm Optimization

Consider a simple example of a feed-forward NN with PSO to understand the algorithm [12].

Let us assume that a fully connected NN has 4 nodes in the input layer, 5 nodes in the hidden layer and 3 nodes in the output layer. Let’s further assume that both activation functions are linear. In Figure 8 the black lines that connect nodes between different layers represent weights of the NN. Besides the weights, other parameters of the NN are biases of the hidden and the output layers.

![Feed-forward NN](image)

*Figure 8 Feed-forward NN*

To calculate the total number of weights and biases, we add the weights between the input nodes and the hidden nodes, the weights between the output nodes and the hidden nodes, and the biases of the hidden nodes and the biases of the output nodes [13].

There are:

- a matrix of $4 \times 5 = 20$ weights from the input nodes to the hidden nodes,
- a vector with 5 biases for the hidden nodes,
• a matrix of 5 x 3 = 15 weights from the hidden nodes to the output nodes,
• a vector with 3 biases for output nodes.

There are total of 20 + 5 + 15 + 3 = 43 weights and biases.

Please also assume that our swarm has 2 particles. Swarm’s best position is the position of any particle with smallest mean squared error, which is called best error.

Each particle is initialized to a random:

• position, a vector in 43-dimensional space
• error for a given position
• velocity, 43-dimensional vector
To calculate the error of any given position, we reconstitute the weights and biases from a position’s coordinates and then apply the NN with these values to the entire training data to calculate the mean squared error.

Figure 9 Demo of first two iterations of 2 particle swarm
For example, to compute the error of the Particle 2’s position in the Iteration 2, the coordinates are broken into matrices and vectors. Specifically, recall that the NN has a 4 x 5 matrix of weights from the input layer to the hidden layer. Thus the first 20 coordinates of the position are the values of this 4 x 5 matrix.

Next 5 values are the biases of the hidden layer. The following 15 values are used in the 5 X 3 matrix of weights from the hidden layer to the output layer. Last 3 coordinates are biases values of the output layer.

----- Particle 2 -------

<table>
<thead>
<tr>
<th>Position:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8.61 0.16 8.57 7.87 -1.26 0.73 -8.66 -8.54 9.55 -6.44 -6.74 7.16 2.32 5.87 -6.26</td>
</tr>
<tr>
<td>-8.66 1.72 8.44 -7.83 0.19 9.30 4.73 3.80 2.23 -4.67 6.35 3.21 1.98 9.53 9.33</td>
</tr>
</tbody>
</table>

-------------------Use position as weights and biases of the NN:

Weights from input to hidden layer:

| -8.41 0.16 8.57 9.87 -1.26 |
| 0.73 -8.66 -0.54 9.55 -6.44 |
| -8.74 9.16 2.32 5.87 -6.26 |
| -9.66 1.72 8.44 -7.83 0.19 |

Biases of hidden layer:

| 9.30 4.73 3.80 2.23 -4.67 |

Weights from hidden to output layer:

| 6.35 3.71 1.98 |
| 9.53 9.33 -8.14 |
| 4.26 4.48 -2.21 |
| -5.51 -1.38 -4.41 |
| 4.85 -8.42 -8.78 |

Biases of output layer:

| 5.80 -6.77 -5.09 |

Error is a mean squared error of NN with above specifications on entire training set.

Error = 9.43

Figure 10 Demo of error calculation for a position
4.1 Architecture

The AsutePeak solution is a portable console application that connects to a private Microsoft Messaging Queue (MSMQ) on any network of machines to receive the data that will be used in the prediction process. MSMQ is a light persistent messaging service that is integrated with the company’s system. MSMQ is selected for its built-in asynchronous message processing [14].

Figure 11 Message queueing architecture

Once the AstutePeak application starts receiving data it de-queues, de-serializes and encodes it before storing into a batch of data tuples for future processing. A categorical data encoding method was based on the research that suggests that effect coding, described below, speeds up the training of the net [15].

The PSO algorithm was chosen based on research that decisively demonstrated the weights converge (i.e. PSO estimates the target weights) faster than other techniques such as the Back Propagation algorithm [16].
4.2 Feature Extraction

During the template execution, the profiling process collects real time information about each rendered method. Method name, count of SQL operations, thread ID, CPU usage and the time taken to render are measured to name a few.

For the initial version of AstutePeak, seven features are selected. Five input values are: categorical string methodName, categorical string className, categorical string contextName, binary patternDetected and a numeric timeTaken. And two output values are numeric cpuDelta, which is the time in milliseconds that the processor has spent on this method, and the numeric memPeak that encapsulates information about the peak physical memory and the virtual memory used to render this method.

Collaboration with CrownPeak’s in-house template developer Oscar Bonifassi leaded to creation of a new binary input value patternDetected. This feature will initially be collected for several methods only.

PatternDetected = IF method is “theSpecificMethod” THEN “theSpecificClass” is used

The patternDetected is very-situation specific and it is defined uniquely for each method. The presence of this pattern suggests whether the best practice in template coding is used.

For example, a developer wants to access database and display some attributes of the data retrieved. In this case, a bad practice would be pulling all attributes of the requested data and then selecting the ones to display. The best practice in this situation is to create a FilterParam object that specifies which attributes are desired and pass this FilterParam along with getFilterList call to only retrieve attributes needed [17].
4.3 Output Derivation

Profiling on the CPU level is very valuable in evaluating the performance of the system. On the other hand, collecting data about CPU performance requires significant resources and noticeably slows down processes that are being measured.

Valuable data is a real time performance metrics of the system. Keeping the above difficulty in mind, during the learning phase of AstutePeak, instances that are going to be profiled on CPU performance are thoughtfully selected so that not to overstep the system’s expected performance standards.

Total processor time is calculated as the difference (delta) between the processor’s time at the ending of profiling and the processor’s time at the beginning of profiling and is measured in milliseconds.

\[
\text{cpuDelta} = (\text{endingSysInfo.TotalProcessorTime} - \text{startSysInfo.TotalProcessorTime});
\]
Where \textit{endingsysInfo} and \textit{startSysInfo} are process put in place to implement profiling the run
time rendering of the system’s functions.

\textit{memPeak} value is based on three metrics: maximum amount of physical memory used by the
process, maximum amount of memory in the virtual memory paging file used by the process and
maximum amount of virtual memory used.

\begin{align*}
\text{memPeakWorkingSet} &= (\text{endSysInfo.PeakWorkingSet64} - \text{startSysInfo.PeakWorkingSet64}); \\
\text{memPeakPagedMemorySize64} &= (\text{endSysInfo.PeakPagedMemorySize64} - \text{startSysInfo.PeakPagedMemorySize64}); \\
\text{memPeakVirtualMemorySize64} &= (\text{endSysInfo.PeakVirtualMemorySize64} - \text{startSysInfo.PeakVirtualMemorySize64});
\end{align*}

\subsection{4.4 Data Encoding}

A success of the entire AstutePeak solution hugely depends on a series of good choices that need
to be made about various components of the network. One of the crucial decisions is data
encoding technique. Since neural networks inherently process numeric data, the categorical data,
such as method and class names need to be encoded with numeric values. The challenge here is
in preserving the data’s inner interconnectivity and meaning when assigning numeric values for
categories.

For categorical input data that is not binary a -1, +1 effects-coding schema is chosen. Suppose
the input variable \textit{methodName} can take on four categorical values: “GetLink”, “GetPanels”,

“Load”, and “LoadUser”. With effects-coding for four categorical values, the first three categorical values are encoded with three-bits, using zeros and a one:

\[
GetLink = (0 \ 0 \ 1)\; \text{GetPanels} = (0 \ 1 \ 0)\; \text{Load} = (1 \ 0 \ 0)
\]

The last categorical value, LoadUser, is encoded with three -1 values: LoadUser = (-1 -1 -1)

Each encoded input value will need a node in an input layer of NN. For example, a categorical value “GetLink” is encoded to \((0 \ 0 \ 1)\) where each bit inputted into its own node.

![Figure 13 Example of categorical value encoding](image)

For numeric input data such as timeTaken variable, the data is normalized to keep the network working within a numeric interval, which speeds up the learning process and produces a better predictor. Gaussian normalization is a useful way to prepare the numeric data for analysis [18].

For example, each timeTaken value \(t\) will be recalculated to 
\(t' = (t - \text{mean})/\text{stdDev}\)

By the same token, the numeric output data is being outputted in normalized form and needs to be de-normalized for further interpretations. Thus, when AstutePeak predicts cpuDelta to be \(d'\), the predicted value is 
\(d = d' \times \text{stdDev} + \text{mean}\)

The binary value patternDetected is simply coded to 1 when true and to -1 when it is false.
4.5 Activation Functions

Two activation functions process and “push” forward the values from one layer to the other. The Hyperbolic Tangent (HyperTan) function is used between the input and hidden layers. The inputs represent the values entering the input layer and the outputs represent the values exiting the hidden layer [19].

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Figure 14 HyperTan activation function

And to transition from the hidden layer to output layer, the Softmax activation function is used. Softmax ensures that all returned values are between 0 and 1. Recall that the raw output value is further de-normalized into an expected output value.

Figure 15 Softmax activation function
4.6 PSO

For the AstutePeak solution, the goal of PSO is to find the best weights and biases within the neural network algorithm. In addition, a particle is defined as a vector of all weights and biases at any given iteration. Thus, the number of coordinates of a point, the dimension of the space, is the total number of all weights and biases of captured with the AstutePeak solutions algorithm.

\[ totalWeights = numInput \times numHidden + numHidden \times numOutput + numHidden + numOutput \]

To sum, a position of a particle at any given time (iteration) is a point with \( totalWeights \) number of coordinates. A “correct” position that we are trying to find are the weights and biases that calculate a given input into a given output.
This section provides the results of the AstutePeak as was described in the previous sections.

Each iteration of the experiments will be reviewed and evaluated against the goals of the project.

For both iterations, the values for categorical input features methodName and className are listed in DataLibrary:

```
```

```
```

Both experiments use a 12-particle swarm.

The accuracy is calculated using a simple algorithm bellow.

```
For each data tuple in testing data set
    Parse data tuple into x-values and t-values
    Compute y-values using the model
    If t-values equal to y-values
        ++numCorrect
    Else
        ++numWrong
End For
Accuracy = numCorrect/(numCorrect + numWrong)
```

For example, if the size of a test dataset is 5 and the model computed the outputs correctly 3 times and missed them 2 times, then the accuracy equals to 3 out of 5 or 60%.

### 5.1 Experiment Iteration 1

The size of the test dataset is 20% of the size of the train dataset. In this iteration of the experiments, the size of train dataset is 40, thus the size of test dataset is 8.
The 75% accuracy of the model that learned on only 40 data tuples is a positive sign. However, given that only a small subset of all possible *methodName* and *className* is considered, the results need further validation to be reliable.

### 5.2 Experiment Iteration 2

In this iteration the train dataset has 400 data tuples, therefore, the test dataset consists of 80 data tuples.
Setting trainSize = 400
Setting numParticles = 12

Training is complete

Best weights and biases are found

<table>
<thead>
<tr>
<th>-9.4</th>
<th>-10.0</th>
<th>2.8</th>
<th>1.1</th>
<th>9.9</th>
<th>6.5</th>
<th>7.2</th>
<th>-10.0</th>
<th>0.3</th>
<th>10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.0</td>
<td>-2.3</td>
<td>-10.0</td>
<td>10.0</td>
<td>-10.0</td>
<td>8.8</td>
<td>8.3</td>
<td>9.2</td>
<td>2.4</td>
<td>7.7</td>
</tr>
<tr>
<td>10.0</td>
<td>2.0</td>
<td>7.9</td>
<td>-10.0</td>
<td>-10.0</td>
<td>-10.0</td>
<td>-2.0</td>
<td>-3.8</td>
<td>-2.0</td>
<td>10.0</td>
</tr>
<tr>
<td>6.5</td>
<td>1.7</td>
<td>10.0</td>
<td>10.0</td>
<td>3.1</td>
<td>-0.7</td>
<td>-2.4</td>
<td>-3.0</td>
<td>9.9</td>
<td>-2.2</td>
</tr>
<tr>
<td>-10.0</td>
<td>0.6</td>
<td>-7.5</td>
<td>-1.2</td>
<td>-4.1</td>
<td>6.3</td>
<td>0.3</td>
<td>10.0</td>
<td>-3.6</td>
<td>-1.3</td>
</tr>
<tr>
<td>10.0</td>
<td>-10.0</td>
<td>1.8</td>
<td>-2.9</td>
<td>-4.6</td>
<td>10.0</td>
<td>8.2</td>
<td>-1.9</td>
<td>-10.0</td>
<td>5.1</td>
</tr>
<tr>
<td>5.5</td>
<td>10.0</td>
<td>-0.8</td>
<td>-2.7</td>
<td>-10.0</td>
<td>-5.6</td>
<td>-1.8</td>
<td>10.0</td>
<td>-10.0</td>
<td>2.9</td>
</tr>
<tr>
<td>5.9</td>
<td>3.3</td>
<td>0.4</td>
<td>6.1</td>
<td>10.0</td>
<td>10.0</td>
<td>-10.0</td>
<td>-7.7</td>
<td>-2.3</td>
<td>10.0</td>
</tr>
<tr>
<td>-7.4</td>
<td>10.0</td>
<td>-7.5</td>
<td>4.3</td>
<td>5.1</td>
<td>2.1</td>
<td>0.5</td>
<td>7.6</td>
<td>-10.0</td>
<td>-7.8</td>
</tr>
<tr>
<td>-0.7</td>
<td>-4.4</td>
<td>-8.5</td>
<td>10.0</td>
<td>-9.3</td>
<td>-4.1</td>
<td>-2.7</td>
<td>-10.0</td>
<td>0.8</td>
<td>-1.6</td>
</tr>
<tr>
<td>7.8</td>
<td>-3.5</td>
<td>-9.7</td>
<td>-4.8</td>
<td>-6.0</td>
<td>-6.0</td>
<td>-10.0</td>
<td>0.3</td>
<td>-2.7</td>
<td>10.0</td>
</tr>
<tr>
<td>-10.0</td>
<td>10.0</td>
<td>8.2</td>
<td>-10.0</td>
<td>-8.5</td>
<td>8.9</td>
<td>-5.9</td>
<td>-10.0</td>
<td>-6.9</td>
<td>3.0</td>
</tr>
<tr>
<td>2.2</td>
<td>-10.0</td>
<td>-10.0</td>
<td>-6.5</td>
<td>-10.0</td>
<td>-4.5</td>
<td>-2.0</td>
<td>-10.0</td>
<td>5.4</td>
<td>-10.0</td>
</tr>
<tr>
<td>-2.1</td>
<td>7.1</td>
<td>0.8</td>
<td>-7.6</td>
<td>-1.8</td>
<td>-4.9</td>
<td>4.8</td>
<td>-3.4</td>
<td>-10.0</td>
<td>-10.0</td>
</tr>
<tr>
<td>-4.5</td>
<td>-5.4</td>
<td>10.0</td>
<td>10.0</td>
<td>9.2</td>
<td>9.0</td>
<td>-5.9</td>
<td>7.6</td>
<td>10.0</td>
<td>7.6</td>
</tr>
<tr>
<td>-9.6</td>
<td>-5.4</td>
<td>-3.1</td>
<td>-10.0</td>
<td>9.5</td>
<td>6.1</td>
<td>2.6</td>
<td>-5.7</td>
<td>-1.8</td>
<td>10.0</td>
</tr>
<tr>
<td>3.5</td>
<td>-5.1</td>
<td>-2.7</td>
<td>1.2</td>
<td>-0.8</td>
<td>2.0</td>
<td>-3.6</td>
<td>0.4</td>
<td>-9.2</td>
<td>-5.0</td>
</tr>
<tr>
<td>3.3</td>
<td>10.0</td>
<td>10.0</td>
<td>-7.1</td>
<td>-2.8</td>
<td>6.3</td>
<td>-6.9</td>
<td>7.5</td>
<td>-7.8</td>
<td>-10.0</td>
</tr>
<tr>
<td>-9.1</td>
<td>3.3</td>
<td>9.3</td>
<td>-5.5</td>
<td>-10.0</td>
<td>-10.0</td>
<td>10.0</td>
<td>-1.5</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>1.6</td>
<td>-7.3</td>
<td>0.2</td>
<td>2.0</td>
<td>-0.0</td>
<td>-10.0</td>
<td>10.0</td>
<td>8.5</td>
<td>10.0</td>
</tr>
<tr>
<td>-6.7</td>
<td>6.1</td>
<td>3.5</td>
<td>10.0</td>
<td>-8.2</td>
<td>10.0</td>
<td>-1.4</td>
<td>10.0</td>
<td>-5.6</td>
<td>3.6</td>
</tr>
<tr>
<td>-9.6</td>
<td>-1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accuracy is: 0.84

The accuracy of the model that learned on 400 data tuples is 84%. The improvement of the accuracy from the previous iteration is expected and furthermore this improvement verifies that the AstutePeak is functioning correctly and it increases its prediction accuracy with an increase in the train dataset size.
6.1 Information Gathering Stage

The description of the entire implementation process is based on the implementation log that was kept since the research stage of this project. This implementation log is widely quoted.

In the early stage of research, the focus was on the evaluation of all available performance data points. The purpose was to find features that are the most suggestive and that are reasonably easy to collect.

First efforts included “simulating the rendering of a “bad” template file by starting out with a well performing template, making changes to output.aspx template file and then previewing the page” [20]. The above steps trigger the execution of the template code and the process that collects real time performance data.

It became clear that collecting real time CPU performance data would create a very high burden on the processing servers. This option has to be abandoned and a new approach must be used to accomplish the project goals.

6.2 Decision Making Process

From the beginning, it was important to make the components of AstutePeak Solution as flexible as possible and to allow for predicting (a.k.a. learning) scenarios in the future. For example, the number and the types of output nodes are adjustable. In addition, the data encoding function must be capable of encoding and decoding any type of data, including categorical, numerical, binary, and more.
Next, difficulty arises with the implementation of a real-time queue of performance profiling data (broken down by methods). This challenge is related to the correct sequence of AstutePeak’s initialization that enables various classes and their processes. A few different approaches like background worker threads and Amazon Web Services (AWS) queues were attempted with limited success. In the end, the Microsoft Messaging Queue (MSMQ) was determined to be an effective solution to this part of the project [21].

6.3 Conclusions

The primary goals of this project were accomplished and deemed a success by both the CrownPeak technology leadership [22] and this researcher.

Thus far, the AstutePeak program learned from only one instance, AdvantGeneral, that is an in-house testing instance. The accuracy of the prediction is as high as 82%. By extension, we can conclude that the theoretical model that was formed can approximate CPU metrics with 82% accuracy or better. However, more time is required to verify these results.

Immediate benefit of the AstutePeak model is the discovery of weights from input methodName nodes to the hidden layer nodes. These weights represent impact that each method has on the system. In other words, they are impactModifiers. Recall that impactModifiers are the values that modify each method to represent its complexity during the execution of the leaky bucket algorithm. With the results produced by AstutePeak, we have a numerical precision we wanted to distinguish one resource request from the next. However, the actual improvement of the leaky bucket algorithm can only be validated over time.
6.4 Future Work

There are two primary recommendations for future work related to this project topic. This way, CrownPeak will be able to fully benefit from the investment made by this researcher over the last 7 months.

First, this project used only 11 of the 150 major methods and 8 of the 20 major classes. It is recommended that the process defined here be expanded to all of the remaining methods, classes and contexts in the CrownPeak CMS environment. Also, the `patternDetected` metric should be implemented in the Performance Profiling App and added to the messages sent to the MSMQ.

Second, it is recommended that the project components be incorporated into an upcoming release of the CrownPeak commercial software offering. In addition, it is recommended that data is continued to be collected and processed in order to refactor the weights and biases on a regular basis.
BIBLIOGRAPHY


[15] S. V. C. E. Fitkov-Norris, "Evaluating the Impact of Categorical Data Encoding and Scaling on Neural Network Classification Performance: The Case of Repeat Consumption
of Identical Cultural Goods," *Engineering Applications of Neural Networks*  


APPENDIX A: Class Diagram
using CrownPeakApp.Instrumentation;
using System;
using System.Threading;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Messaging;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace AstuteProcessor
{
    class AstutePeakProgram
    {
        static MessageQueue mq;
        static int trainCounter = 0;
        static int testCounter = 0;
        static int trainSize = 1000;
        static int testSize = (int)(0.2 * trainSize);
        static bool gatherTrainData;
        static string[][] trainData;
        static double[][] encodedTrainData;
        static string[][] testData;
        static double[][] encodedTestData;
        static Encoder e;
        static int numInput;
        static int numHidden;
        static int numOutput;
        static string[] colTypes;
        static NeuralNetwork nn;
        static int numParticles = 12;
        static int maxEpochs = trainSize;
        static double exitError = 0.060;
static double probDeath = 0.005;

static void Main(string[] args)
{
    gatherTrainData = true;
    //
    trainData = new string[trainSize][];
    testData = new string[testSize][];
    // set up number of nodes in AstutePeak
    numInput = DataLibrary.methodsList.Length - 1 + DataLibrary.classesList.Length - 1 + 1;
    numOutput = 2; //CPUdelta and peakMem
    numHidden = (numInput + numOutput)/2; //adjustable
    colTypes = new string[5] { "categoricalX", "categoricalX", "numericX", "numericY", "numericY" };
    // set up a private MSMQ named "CMSAstutePeak"
    String queueName = String.Format( @"\ privately\CMSAstutePeak", Environment.MachineName);
    if (!MessageQueue.Exists(queueName))
    {
        mq = MessageQueue.Create(queueName);
    }
    else
    {
        mq = new MessageQueue(queueName);
    }
    mq.ReceiveCompleted += mq_ReceiveCompleted;
    mq.BeginReceive();
    Thread.Sleep(Timeout.Infinite);
    Console.WriteLine("Completed AstutePeak\n");
    Console.ReadKey();
}

static void mq_ReceiveCompleted(object sender, ReceiveCompletedEventArgs e) // accumulate data from MSMQ
{
    XmlSerializer serializer = new XmlSerializer(typeof(CallEntity));
    // Declare an object variable of the type to be deserialized.
    CallEntity call;
    call = (CallEntity)serializer.Deserialize(e.Message.BodyStream);
    //save into matrix of raw data tuples
    string[] sf = SelectFeatures(call);
    if (sf != null && gatherTrainData)
    {
        trainData[trainCounter] = sf;
        trainCounter++;
        Console.WriteLine("Saved a tuple into trainData\n");
        Console.WriteLine(trainCounter);
    }
    else if (sf != null && !gatherTrainData)
    {
        testData[testCounter] = sf;
        testCounter++;
        Console.WriteLine("Saved a tuple into testData\n");
    }
Console.WriteLine(testCounter);
}

/// send train data to AstutePeak for supervised learning
if (trainCounter >= trainSize && gatherTrainData)
{
    gatherTrainData = false;
    nn = new NeuralNetwork(numInput, numHidden, numOutput);
    encodedTrainData = Encode(trainData);
    double[] bestWeights = nn.Train(encodedTrainData, numParticles, maxEpochs, exitError, probDeath);
    Console.WriteLine("Best weights and biases are found\n");
    ShowVector(bestWeights, 10, 1, true);
    Console.ReadKey();
}

/// caculate and display accuracy
if (testCounter >= testSize && !gatherTrainData)
{
    Console.WriteLine("The acuracy of the NN with these best weights is:\n");
    encodedTestdata = Encode(testData);
    double accuracy = nn.Accuracy(encodedTestdata);
    Console.WriteLine("Accuracy is " + accuracy + "\n");
    Console.ReadKey();
    return;
}

(MessageQueue)sender).BeginReceive();

public static string[] SelectFeatures(CallEntity call) // {methodName, className, timeTaken, cpuDelta, memPeak}
{
    string[] rawTuple = null;
    CallEntity entity = call;

    string[] fullNames = entity.MethodName.Split(' 
);

    string methodName = fullNames[3];
    if (!DataLibrary.methodsList.Contains(methodName)) return null;
    else rawTuple[0] = methodName;

    string className = fullNames[2];
    if (!DataLibrary.classesList.Contains(className)) return null;
    rawTuple[1] = className;

    double timeTaken = entity.timeTaken;
    rawTuple[2] = timeTaken.ToString();

    double cpuDelta = (double)entity.cpuDelta;

    return rawTuple;
}

    double memPeak = (double)entity.memPeakWorkingSet;
}

return rawTuple;
}

public static double[][] Encode(string[][] data)
{
    // categorical x-data : MethodName
    // categorical x-data : ClassName
    // numeric x-data : TimeTaken
    //
    // numeric y-data : cpuDelta
    // numeric y-data : memPeak

double[][] encodedData;
    e = new Encoder(data, colTypes);
    encodedData = e.EncodeAll(data);
    return encodedData;
}

static void ShowVector(double[] vector, int valsPerRow, int decimals, bool newLine)
{
    for (int i = 0; i < vector.Length; ++i)
    {
        if (i % valsPerRow == 0) Console.WriteLine("");
            Console.Write(vector[i].ToString("F" + decimals).PadLeft(decimals + 4) + " ");
    }
    if (newLine == true) Console.WriteLine(" ");
}

class NeuralNetwork
{
    // Code adapted from Visual Studio Magazine article:

}

class Particle
{
    public double[] position; // equivalent to NN weights
    public double error; // measure of fitness
    public double[] velocity;

    public double[] bestPosition; // best position found so far by this Particle

}
public double bestError;

public Particle(double[] position, double error, double[] velocity, double[] bestPosition, double bestError)
{
    this.position = new double[position.Length];
    position.CopyTo(this.position, 0);
    this.error = error;
    this.velocity = new double[velocity.Length];
    velocity.CopyTo(this.velocity, 0);
    this.bestPosition = new double[bestPosition.Length];
    bestPosition.CopyTo(this.bestPosition, 0);
    this.bestError = bestError;
}

class Encoder
{
    // Code adapted from Visual Studio Magazine article:

}

static class DataLibrary
{
    // holds arrays of all possible categorical values
    // for future lists can be updated ad hoc
    // methodsList, classesList, contextList
}