ON THE MODELING OF INTUITION

A Framework for Exploration

Dustin Fast

CSCE A470 / Fall, 2018

University of Alaska, Anchorage

"To explain the integration of information, we need only exhibit mechanisms by which information is brought together and exploited by later processes."

-David Chalmers, Facing Up to The Problem of Consciousness

Contents

Introduction

General intelligence of the type we possess exists exclusively, for now, in the domain of the conscious human. Therefore, an understanding of the mechanisms leading to our conscious experience may be required if an artificial general intelligence is to one day be realized.

One defining aspect of human intelligence is our ability to subconsciously form new connections between abstract concepts, which then seem to "bubble up" to the forefront of our attention. This phenomenon, commonly called intuition, is likely responsible not only for our most startling and profound "Aha!" moments, but also for the seemingly arbitrary changes in our awareness of, say, the ticking of a clock on the wall.

Although intuition is, unfortunately, a system that exists inside us as a “black box” (we have no conscious access to its decision-making process), the ways that we experience these shifts of attention unwillingly and "out of the blue" provide powerful clues to its underlying mechanisms.

With the above in mind, a framework was developed to explore these ideas and within it an ensemble learning system (the *agent*) was developed, where the agent’s awareness is directed according to some optimization function (an *intuition)* with the goal of recognizing symbols in its search-space and forming new symbolic connections between them.

Technologies

The Agent was developed in the Python (3.6) programming language. The 3rd party libraries KarooGP and PyTorch were used for their genetic programming and machine learning capabilities, respectively.

Problem Domain

The agent was designed to exist an environment that would -

A. Afford the agent an opportunity to explore a complex, unknown search-space.
B. Be multi-context.
C. Provide mechanisms for signaling feedback to the agent.
D. Have the potential for practical application.

Therefore, a specific problem domain meeting these criteria should be chosen. A problem of this type and why it meets these criteria is described in section *Validation / Analysis*.

The Model

The intuitive model was conceived to represent a "sixth sense" interpretation of our intuition (no supernatural connotation intended), because subjectively (from the perspective of our awareness) experiencing intuition “feels” no different than experiencing input from any of our other garden-variety five senses, except for at least one notable difference: intuitive input carries with it contextual meaning and symbolic comprehension about our environment - ideas composed by filtering sensory input through the sieve of one’s accumulated life experience. In this way it can be thought of as a sixth sensory organ, different from the first five in that it serves information that has been pre-processed by our sub-conscience.

Along with this sixth-sense interpretation, the agent was also designed with the following observations of human behavior in mind -

- Observation 1

  a) We can react to events faster than we have time to logically determine a rational course of action (Klapproth, 2008). Regardless, we may still make very good “in-the-moment” approximations that seem to involve no conscious thought.

  b) We can articulate the rules we use to solve a given problem but are generally unable to explain why we chose to consider one specific set of rules over another (Pitrat, 2010).

  c) Shifts in changes to our awareness are often (if not always) autonomic (Shulman & Corbetta, 2002). For example, we cannot dictate when a song will become stuck in our head, or which clock we can suddenly hear ticking.

  Conclusion 1

  Some system operating below our level of consciousness exists for selectively serving information into our awareness.

- Observation 2

  Seemingly trivial and/or unproductive “mistakes” often come into our awareness. For example, songs DO get stuck in our head and we DO become suddenly aware of a ticking clock for no apparent reason. Contrapositively, we’re often oblivious to important environmental queues, especially when sufficiently distracted (a trait exploited by magicians and pick pockets alike).

Conclusion 2

Intuition is not perfect. However, "mistakes” have evolutionary value (e.g. genetic mutation compels biological adaptation). Therefore, as a biological system itself, it is reasonable to assume that the mechanism by which the subconscious learns to optimally processes and serve information is Darwinian in nature.

- Observation 3

  The state of our awareness affects our intuition. Contrarywise, the state of our intuition affects our awareness. To demonstrate this, consider what occurs when focusing one’s awareness on a particular topic; we become acutely aware of *it* and much less aware of everything *else*.

Conclusion 3

Awareness and intuition exist as a feedback loop, each guiding each other in lockstep.

- Observation 4

  We possess the ability hold, and operate on, a limited set of symbols in our head at one time. Summing three numbers in one’s head, for example.

Conclusion 4

Humans possess a short-term “working memory”.

- Observation 5

  It is human nature to meticulously explore our environment for personal gain. As evidence of this, I offer humankind’s domination of its natural habitat through technological innovation.

Conclusion 5

An environmentally-aware agent motivated by evolutionary forces and possessing an intuition may naturally act to explore its environment and actively develop structure from it.

Design

The intuitive model (Fig. 1) consists of 4 layers, labeled ***Classification***, ***Evolutionary***, and ***Logical***. Data is mostly feed-forward with a single feedback channel for backward-signaling fitness metrics. Two output channels also exist: performance metrics (e.g. for logging) and actual agent output representing it's current awareness.

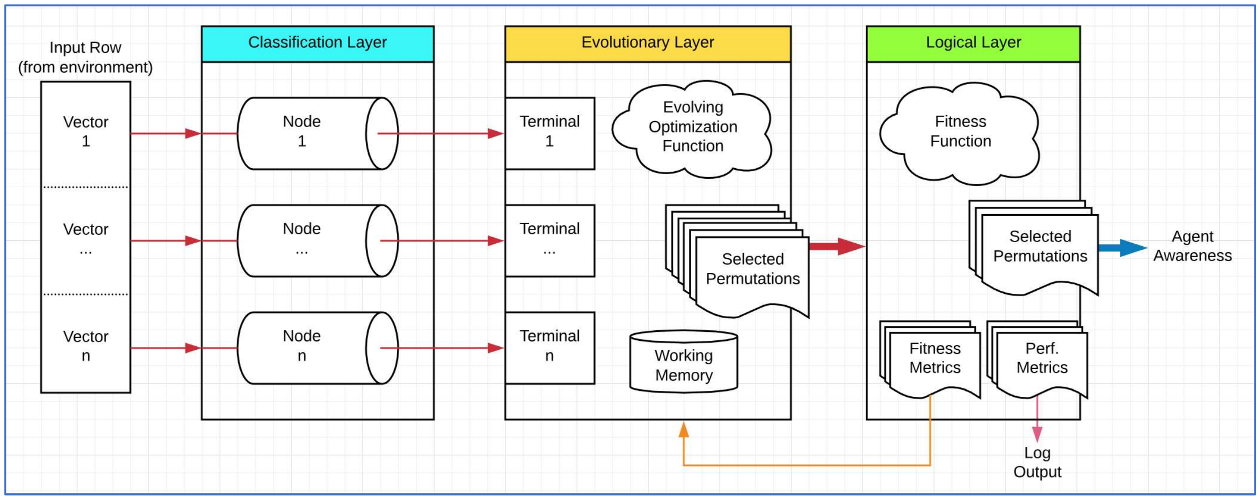


Figure 1

Layer 1 – The ***Classification*** Layer

Represents our ability to rapidly (intuitively) classify patterns found in our environment based on our previous exposure to similar patterns.

This layer is a set of classifiers built as an arrangement of `d` parallel **nodes** and can be thought of as an input bus, where each node is a single line on the.

Input

Input is received by this layer as a row of environment data. Each row is a collection of *vectors*, one for each node. Thus, input to each node is a sample of some subset of the agent's environment.

(Format note: `d` = the number of vectors in the input row, a number that should be consistent across all rows of environmental input data. If it is not consistent, the smallest width among all rows is used.

Operation

Each node assigns its given input vector a label (a *classification*) from its pool of pre-learned symbol labels.

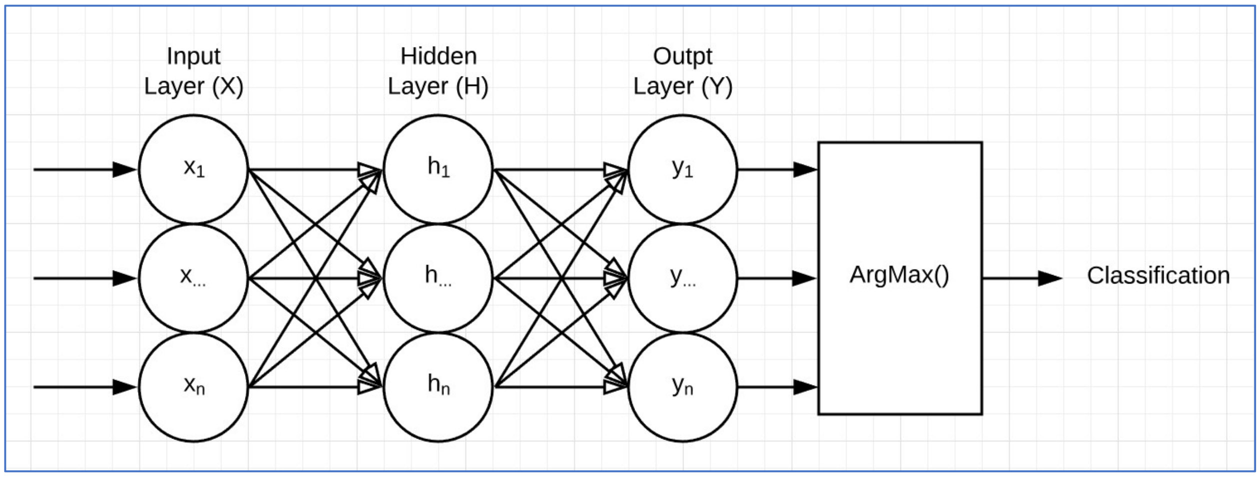
Learning note: Each node is trained independently (though still in parallel) in a supervised and offline fashion from a training/validation set specific to that that node.

Output

This layer's output is the set c of all node classifications for the current set of input row vectors.

Implementation

Each classification-layer node is a classifier implemented as an artificial neural network (ANN) of the following shape (Fig. 2). The depth of each ANN's layers is determined dynamically based on the properties of the training data used to train that node.



	X	**H**	**Y**
Input Function	Linear	Linear	Linear
Activation Function	Rectified Linear	Rectified Linear	Sigmoid

Figure 2

Layer 2 – The ***Evolutionary*** Layer

Represents our ability to form new symbolic connections from existing symbols in our environment.

This layer is composed of two parts, 1) An optimization function of `k` output templates, and 2) A short-term working memory of a predefined depth `j` allowing the layer to generate its output from both its current inputs (called *terminals*) and the previous `j` inputs.

Input

Input is received from the ***classification*** layer as set `C` (having size `d`) of symbols.

Operation

A string is generated from each output template as an arrangement of symbols from the symbol-pool `S`. where `S` is `C` plus the previous `j` instances of `C`.

Learning Note: Each template is optimized in an online manner as fitness is signaled back from the ***logical*** layer. In this way the agent, as it progresses, learns to optimally allocate its "attention" while still allowing "mistakes" to enter its awareness. These mistakes represent possible new conceptual connections and will be validated by the subsequent layer.

Output

This layer's output is the set `P` (having size `k`) of strings.

Implementation

The optimization function is represented internally as a set `K` of `k` genetically evolving binary expression trees. The leaf nodes of each tree denote a single index in `G`, thus the sequence of a tree's leaf nodes denotes a sequence of symbols from `G`.

For example, suppose:

`S` = {'D', 'R', 'W', 'V', 'U', 'E', 'G', 'T', 'O', 'A', 'F', 'H', 'K', 'S', 'L', 'O'},

and

`K` = {Tree A, Tree B}, where Tree A and Tree B are given by Fig. 3.

Then:

Tree A denotes the string "HELLO" and Tree B denotes the string "WORLD".

Therefore:

Output set `P` = {"HELLLO", "WORLD"}

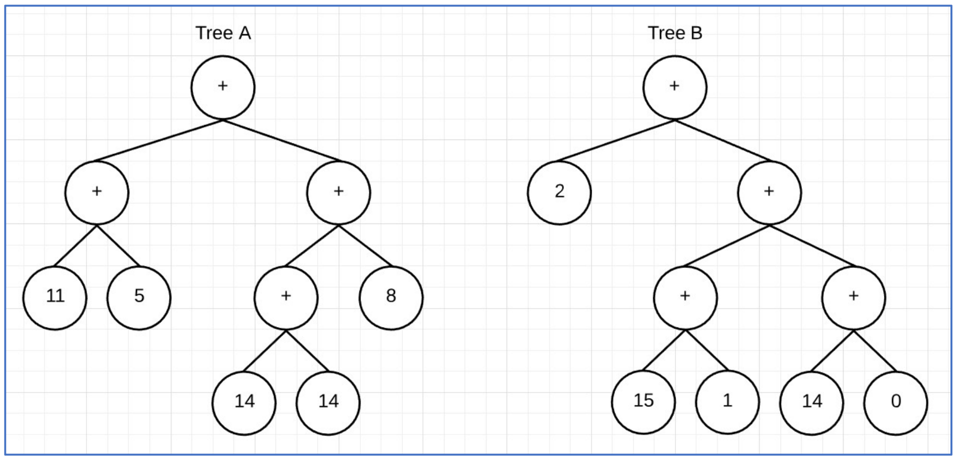


Figure 3

Layer 3 – The ***Logical*** Layer

Represents our ability to validate ideas against the environment.

This layer provides the mechanisms by which the agent validates the contents of its awareness against its environment as well as computing performance metrics such accuracy and processing time.

Input

Input is received from the ***evolutionary*** layer as set `P` of strings.

Operation

Each string in P is validated according to the context mode. Fitness information is then sent as feedback to the ***evolutionary*** layer.

Output

This layer's output is a set of strings representing the agents current awareness.
Initially these strings will appear random, but as the agent progresses they will begin to "narrow in" on the symbolic connections in its environment.

Implementation

The context mode defines this layer and is implemented as a function returning either True or False. For example – if `P` = {"HELLO", "WORLD"}, and `context mode` = is_noun(), then `fitness` = {0, 1}.

The Agent

The agent contains the intuitive model and steps each row of environmental input data through it.

Each row sent through the model constitutes a single **step**, and the goal of a step is to 1) classify the current row's data, 2) tokenize the classifications according to the optimization function, 3) assign each token a fitness value, 4) update the optimization function according to fitness, and 5) serve tokens to the agent.

Each step proceeds, goal-wise, as follows -

1. Input vectors `V` (segments of the agent's current input row*)* are fed into the ***classification*** layer.
    a. Each pre-trained classifier node `ni` assigns each `vi` a classification label `ci`.
2. Each `ci` (a *terminal symbol*) is received by the ***evolutionary***-layer's corresponding `terminali`.
    a. The current set of terminals are added to working memory. (If working memory now exceeds its predefined depth, the oldest set of terminals in working memory is removed.)
    b. A predefined number of permutations `P` are generated, each from one or more terminals in working memory according to the optimization function's current set `E` of expressions.
3. `P` is received by the ***logical*** layer.
    a. Each `pi` is evaluated for fitness according to the agent's *context mode* (ex: `is_noun(token)`). If `pi` is found to be a productive permutation, the fitness metric for `ei` is incremented and `pi` is added to output set `O`
4. Fitness data for `E` is signaled back and received by the ***evolutionary*** layer.
    a. A new generation of `E` is bred from (and replacing) the previous generation in a competitive environment.
5. `O` is made available to the agent.

Sample Agent

A "batteries included" agent is described in detail in the next section: *Validation / Analysis*.

Validation / Analysis

Upon completion of development, an agent was brought online, trained, and validated according to the following campaign –

Tunable Parameters

The agent module `agent.py` contains several customizable parameters. For this campaign, they were initialized as follows –

Agent

```
AGENT_NAME = 'agent1_memdepth1'   # Log file prefix
AGENT_ITERS = 10                  # Num times to iterate AGENT_INPUTFILES
```

Layer 1

```
L1_EPOCHS = 1000                 # Num L1 training epochs (per node)
L1_LR = .001                     # Classifier learning rate (all nodes)
L1_ALPHA = .9                    # Classifier lr momentum (all nodes)
```

Layer 2

```
L2_KERNEL_MODE = 1               # 1 = no case flip, 2 = w/case flip
L2_MUT_REPRO = 0.10              # Genetic mutation ration: Reproduction
L2_MUT_POINT = 0.40              # Genetic mutation ration: Point
L2_MUT_BRANCH = 0.10             # Genetic mutation ration: Branch
L2_MUT_CROSS = 0.40              # Genetic mutation ration: Crossover
L2_MAX_DEPTH = 5                 # Max is 10, per KarooGP (has perf affect)
L2_GAIN = .75                    # Fit/random ratio of the genetic pool
L2_MEMDEPTH = 1                  # Working mem depth, an iplier of L1's input sz
L2_MAX_POP = 50                  # Genetic population size (has perf affect)
L2_POOLSZ = int(L2_MAX_POP * .25)  # Genetic pool size
```

Layer 3

```
L3_CONTEXTMODE = Connector.is_python_func   # Agent's context mode
```

Input Files

```
# Agent input data set - length denotes the number of layer-one nodes
AGENT_INPUTFILES = ['static/datasets/letters0.csv',
                    'static/datasets/letters1.csv',
                    'static/datasets/letters2.csv',
                    'static/datasets/letters3.csv',
                    'static/datasets/letters4.csv',
                     'static/datasets/letters5.csv',
                     'static/datasets/letters6.csv',
                     'static/datasets/letters7.csv']

# Layer 1 training data (by node) - length must match len(AGENT_INPUTFILES)
L1_TRAINFILES = ['static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv',
                 'static/datasets/letter_train.csv']

# Layer 1 validation data (by node) - length must match len(AGENT_INPUTFILES)
L1_VALIDFILES = ['static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv',
                 'static/datasets/letter_val.csv']
```

Campaign Problem Domain

A problem domain was chosen to meet the criteria described in section *Introduction->Problem Domain*. They are shown here again for convenience.

The agent's environment should -

E. Afford the agent an opportunity to explore a complex, unknown search-space.

F. Be multi-context.

G. Provide mechanisms for signaling feedback to the agent.

H. Have the potential for practical application.

With these constraints in mind, the agent was applied to the task of learning Python programming language function names (with the eventual goal of combining with other agents to learn the Python programming language – for more information on this, see section *What's* Next). This meets our criteria very well, because Python -

A. Is a complex space in which an isolated learning environment may be constructed.

B. Provides various contexts. Ex: keywords, functions, arguments, classes, instances, programs, etc.

C. Exposes methods such as `keyword.iskeyword(str)` and `callable(str)`, enabling the agent to "ask" if some string represents a Python keyword or a callable function, respectively. Further, because the agent itself is written in Python, custom methods may be written to provide more specific feedback, such as `is_python(str)` which returns True if the string represents a valid Python program with no syntax errors and generating no exceptions.

D. Is highly reflexive, allowing a sufficiently advanced agent to write a version of itself that solves some arbitrary problem. Indeed, even writing its own feedback mechanisms for querying fitness heuristics.

Data Sets / Agent Shape

Given the problem domain, a corpus* of handwritten character image features and their associated labels was chosen as the agent's environmental input, with each feature vector in the set representing one of the 26 characters (A-Z) of the English alphabet.

The data set was segmented for use as follows –

File	**Purpose**	**Contents**
'static/datasets/letter_train.csv'	Training file for all L1 nodes	15,000 unique feature vectors from the corpus.
'static/datasets/letter_val.csv'	Validation file for all L1 nodes	The remaining 5,000 unique feature vectors from the corpus.
'static/datasets/letters0.csv'	Agent input rows, segment 1	1,000 feature vectors chosen randomly from the corpus
'static/datasets/letters1.csv'	Agent input, segment 2	"
…	Agent input, segments 3 - 7	"
'static/datasets/letters7.csv'	Agent input, segment 8	"

*University of California, Irvine, Machine Learning Repository, *Letter Recognition Data Set* (https://archive.ics.uci.edu/ml/datasets/Letter+Recognition)

With the data segmented this way, and given the tunable parameters previously described, an agent of the following shape results (Note **bold** items, Fig. 4 & 5):

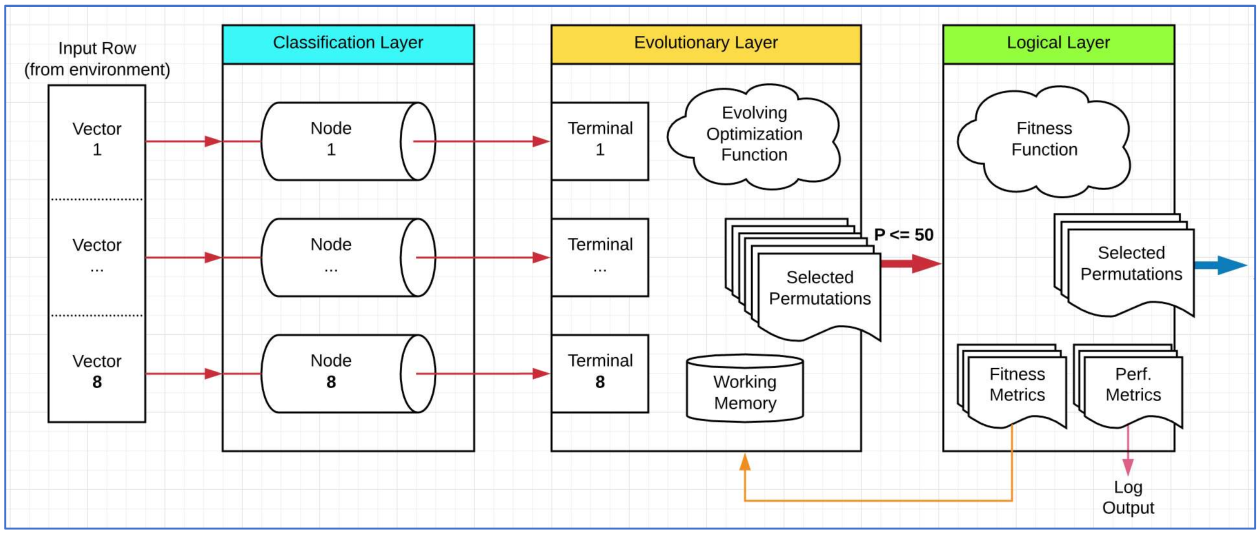


Figure 4

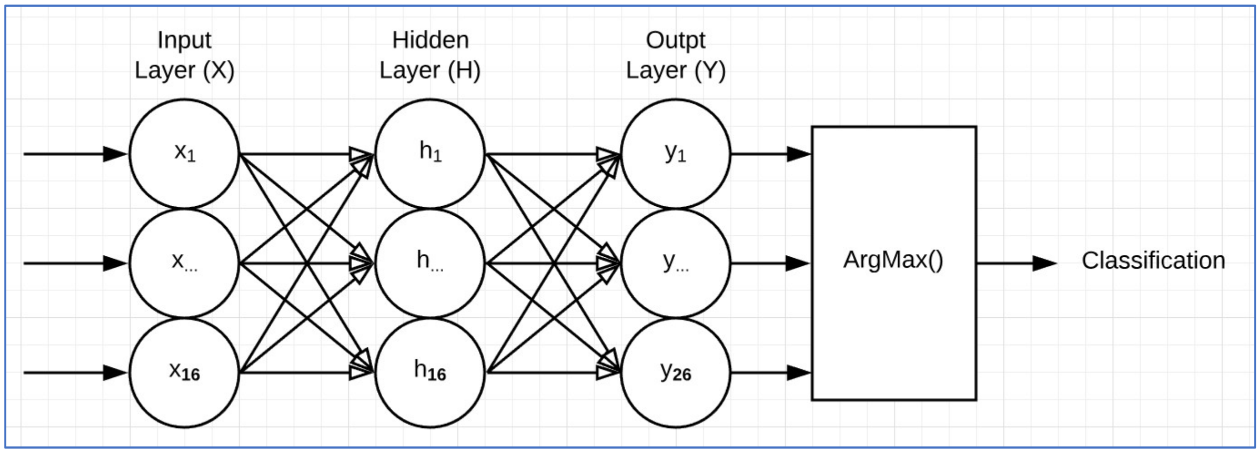


Figure 5

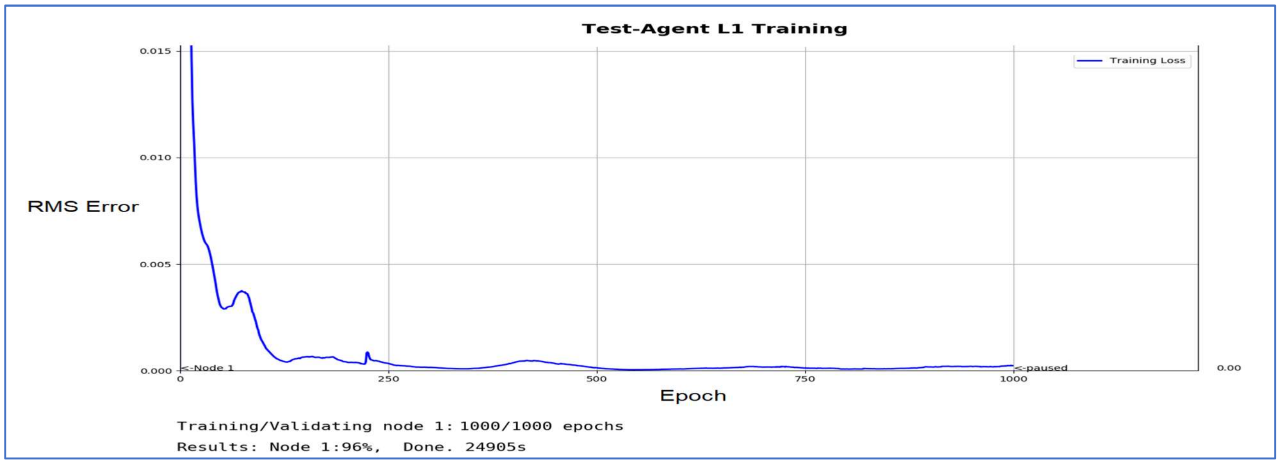
Layer 1 Training

The agent's layer-one nodes were trained over 1000 learning iterations and resulted in a 96% validation accuracy.

To execute training on all layer-one nodes, enter the following command at the terminal. Doing so will randomly initialize each node's weight's and biases before starting the training/validation routine.

```
./agent.py -l1_train
```

For this campaign, the 1th node's training results are shown by the graph in Fig. 6, demonstrating a well -trained network.



```
Training/Validating node 1: 1000/1000 epochs
Results: Node 1:96%,  Done. 24905s
```

Figure 6

Note: Each layer 1 node in this campaign uses the same training and validation file. However, each node is still trained individually, via the command shown, under the assumption that this is not the case. Therefore, it would have been more efficient to train one node and copy the resulting model to each other node, especially given the training time (24,095 seconds, or 6+ hours!) per node.

Running the Agent

With layer-one pre-trained, the agent was run on it's given input files and checked against the current context.

To run the agent, enter the following command at the terminal –

```
./agent.py
```

The agent's current context was set (according to the tunable parameters) to be the predefined function Connector.is_python_func, located in `connector.py` and restated here for convenience -

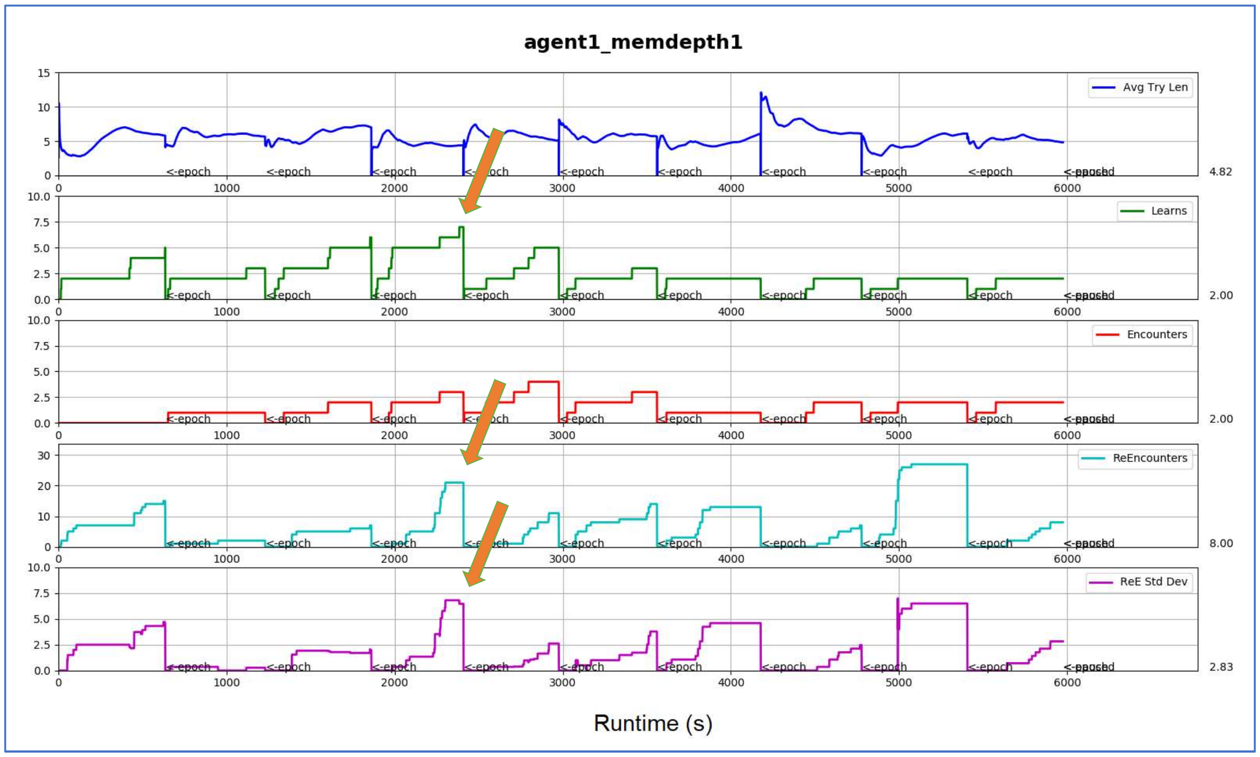
```
def is_python_func(p):
    """ Returns True iff the string p is a python function name.
    """
    try:
        return callable(eval(string))
    except:
        return False
```

This function takes a single parameter `p` (a string) where `p` is an element of set `P`, uses Python's capability for reflection to evaluate it as a callable function, and then returns either True or False, signaling that the string was (or was not) in fact a Python function name (even for, say, the function named 'callable') . It is in this way that the agent receives feedback from its environment about which permutations (generated by layer two) were found productive, and, over time, learns to evaluate it's search space more effectively based on this feedback.

As the agent runs, it iterates over its input data. For each of these *epochs*, the entire input set is iterated sequentially, thus each epoch has the agent encountering the input data in the same order as it did the previous epoch. What changes over time (in layer two) is the arrangement those inputs get served to layer three in. With this in mind, a "learning" agent may be demonstrated by noting an increase in the number of positive (Boolean True) feedback signals it receives for each epoch's unique productive permutations, as it is "narrowing in" on the range of symbols representative of a Python function.

Results

A trend of the kind described in the preceding section may be seen in our agent's output (Fig. 7). The primary points of interest to our analysis are noted by the arrows in Figure 7. By the first arrow, we see that the agent began extracting a progressively larger number of Python function names from its inputs over epochs 1 through 4. However, that number drops beginning with epoch 5. This might suggest overtraining and, indeed, the second two arrows suggest both a high number of repeat permutations as well as a large gap in their respective number of occurrences – the agent may have found 7 new function names in epoch 4, but near the end it was likely re-encountering the same one or two repeatedly. This would have quickly led to an elitist gene pool and may explain the numbers we see over the next six epochs.



Avg Try Len	The average length of each permutation `p`
Learns	The number of unique `p`'s seen this epoch found productive. Presumably this number should grow larger over successive epochs as the model learns to better search its environment,
Encounters	The number of unique productive `p`'s seen this epoch that were also seen in one or more previous epochs
ReEncounters	The number of non-unique productive `p`'s seen this epoch
ReE Std Dev	The standard deviation among the occurrences of each `p` accounted for in the ReEncounters metric. This figure provides a measure of the diversity among the productive arrangements the model is generating

Figure 7

What’s Next

Moving forward, these results must be shown to be repeatable and then learning parameters may be tuned (possibly dynamically, by the agent itself) to prevent the overtraining described immediately above.

In addition, noting the hierarchal nature of information (including that of our problem domain) the agent was designed to scale from a single agent, to a node-agent in a network of such agents, thereby bootstrapping an increasingly advanced intuition (Fig. 8). It is in this way that an agent might come to write its own programs, including other version and extensions of itself, in Python.

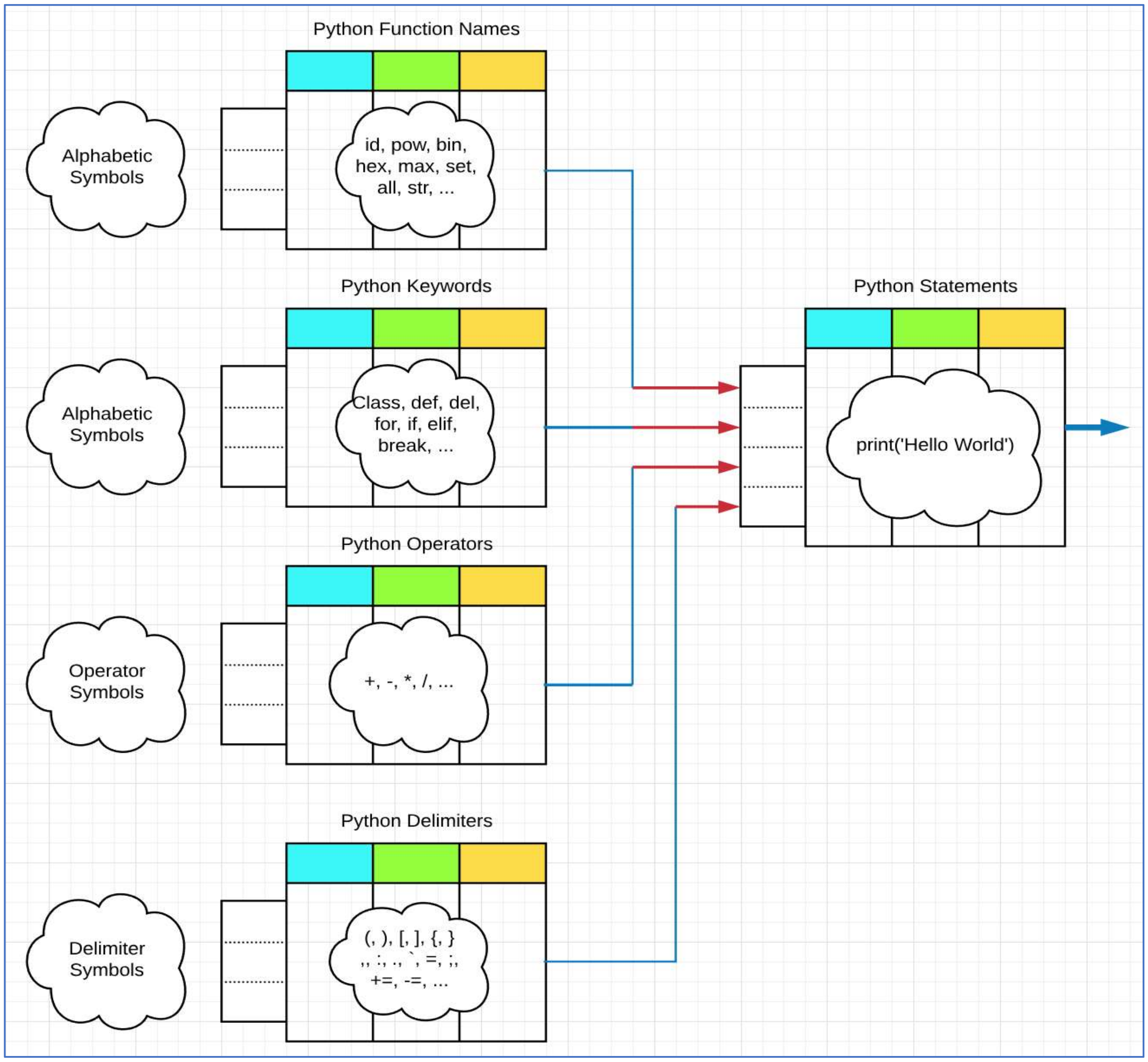


Figure 8

Code Base

The source code for the agent, including this file and others, is available at https://github.com/dustinfast/intuitive_agent.

Note: The following dependencies are required for operation:

Dependency	**Installation**
Karoo GP	N/A (see lib/karoo_gp)
Matplotlib	pip install matplotlib
Numpy	pip install numpy
Pandas	pip install pandas
PyTorch	See https://pytorch.org
Requests	pip install requests
Scikit-learn	pip install scikit-learn
Sympy	pip install sympy
Scipy	pip install scipy
Tensorflow	See https://www.tensorflow.org/install

References

Klapproth, F. (2008). *Time and Decision Making in Humans*. Cognitive, Affective, & Behavioral Neuroscience, 8 (4), 509-524.

Pitrat, J. (2010). *Artificial Beings*. Wiley-ISTE.

Shulman, G. L., & Corbetta, M. (2002). *Control of goal-directed and stimulus-driven attention in the brain*. Nature Reviews Neuroscience, 3(3), 201-215.