Hybrid Artificial Intelligence & Machine Learning Systems

Logan Kearsley
TJHSST Computer Systems Lab 2006-2007

Abstract:

The purpose of this project is to design a set of libraries for manipulating AI and Machine Learning algorithms and demonstration programs that combine the capabilities of multiple AI and Machine Learning models to produce a more flexible and versatile hybrid model.

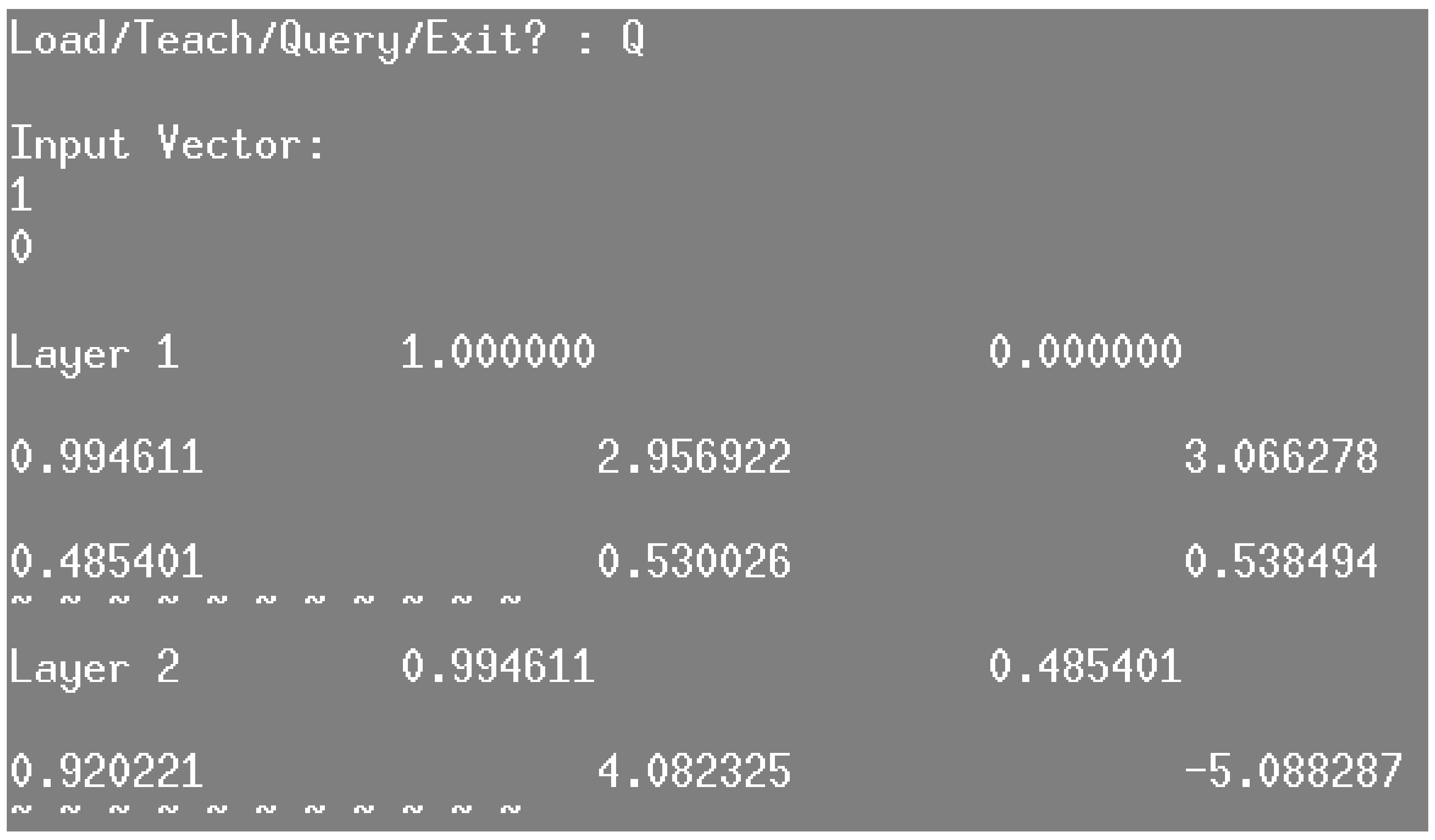
I have worked primarily with back-propagation neural networks and subsumption architectures.

The final demonstration program is a highly extensible character recognition system with the capacity to add new characters reasonably simply and rapidly.

Project Development:

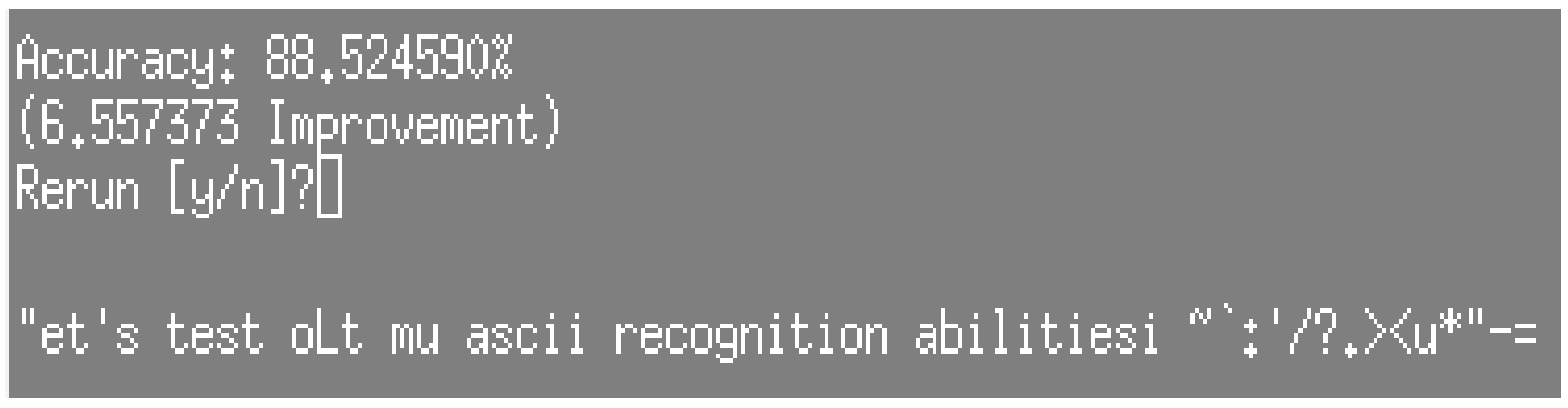
The programming for this project was done in C, using the gcc compiler and make. The neural network library makes use of a back-propagation learning algorithm with weight matrices to simulate neuron layers. I devised a new data type, neuro_t, to store all of the relevant information for a single net with arbitrary dimensions.

Development consisted of 3 major stages:
- Model Development- creating the code for the neural networks, including a back-propagation learning algorithm
- Algorithmic Testing- evaluating and improving the effectiveness of the AI algorithms mostly by feeding test programs random input/output associations to ensure that they learn properly.
- Application Testing- writing a character recognition program and testing with data from a custom bitmap font.



```
Load/Teach/Query/Exit? : Q

Input Vector:
1
0

Layer 1         1.000000                0.000000

0.994611                2.956922                3.066278

0.485401                0.530026                0.538494
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Layer 2         0.994611                0.485401

0.920221                4.082325                -5.088287
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

Testing Interface Screenshot: Weight matrices for computing boolean XOR.



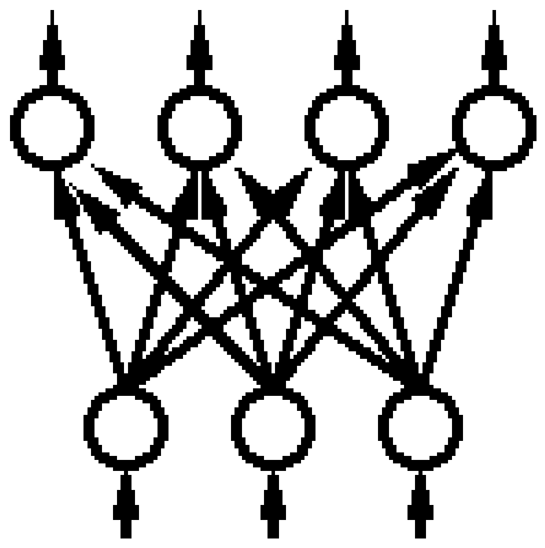
```
Accuracy: 88.524590%
(6.557373 Improvement)
Rerun [y/n]?

"et's test oLt mu ascii recognition abilitiesi ^`:'/?.><u*"-=
```

Training Program Screenshot:
The program compares its transcription to a user-provided answer file and uses the data to re-train its component networks.

Results:

The most generally useful result of this project is the neural network library, which allows for the easy creation and manipulation of arbitrarily-dimensioned BP networks, which have applications in a wide array of classification problems; in contrast, it has been extremely difficult to design a generalized subsumption architecture wrapper; except in special cases, it is likely best to simply write new subsuption programs from scratch.

The character recognition demonstration program is not easily generalizable for use with inputs other than the highly structured experimental data it was designed for; however, it has successfully demonstrated the increased extensibility and learning speed of the hybrid neural network / subsumption architecture system over a single large pure neural network.



In1	In2	In3	
W1,1	W1,2	W1,3	Out1
W2,1	W2,2	W2,3	Out2
W3,1	W3,2	W3,3	Out3
W4,1	W4,2	W4,3	Out4

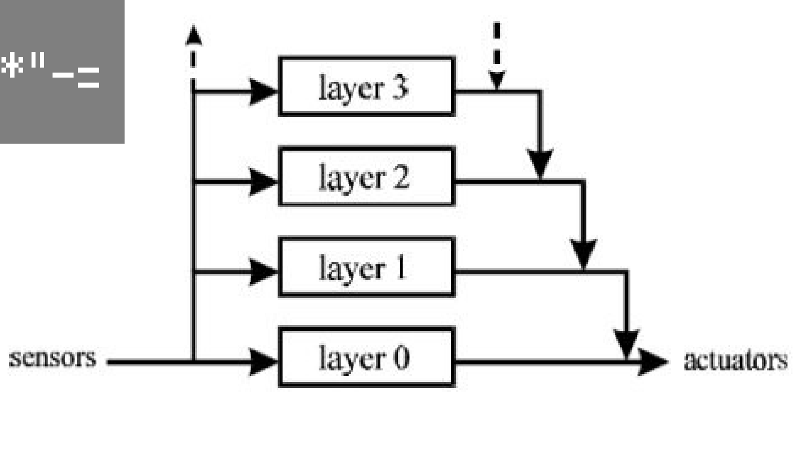
Example Network Layer &
the Associated Weight Matrix:
Storing weight matrices for each layer eliminates the need to simulate individual neurons; a zero-valued weight corresponds to a broken inter-neuron connection.

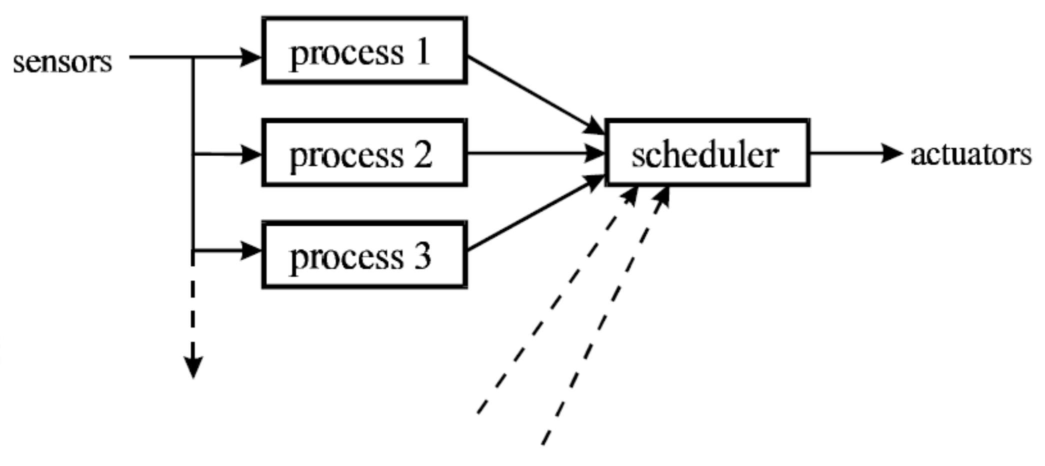
Background:

Most AI and Machine Learning research has consisted of developing separate models to maximize utility for a single problem type or to duplicate biological models. Comparatively little research has been done in AI systems combining the best aspects of multiple models to produce a more versatile hybrid system. 'Hybrid neural networks' were proposed in the late 1980s to incorporate symbolic processing and neural processing into a single system for speed and ease of control while retaining the flexibility of pure neural nets. In this project, I've attempted to combine multiple neural networks under a subsumption architecture model.

The subsumption architecture model was first used in 1984, primarily for controlling robots. Subsumption architectures make use of multiple simple rules or sub-programs (here defined by different networks) layered over each other to produce more complex emergent behaviors. Sub-programs may be arranged in a permanent hierarchy, or may compete for priority through a scheduler on each run through the program's main loop. In this project, I've used the competitive subsumption model.

Permanent Priority
vs. Competitive Subsumption:


layer 3
layer 2
layer 1
layer 0
sensors
actuators


sensors
process 1
process 2
process 3
scheduler
actuators