Project 3a: ANN

AI CS 570

Max McKinnon

April 6, 2012

Contents

[Abstract 3](#_Toc321405005)

[Algorithm 3](#_Toc321405006)

[Artificial Neural Network (ANN) 3](#_Toc321405007)

[Node 3](#_Toc321405008)

[Inputs 4](#_Toc321405009)

[Outputs 4](#_Toc321405010)

[Training 4](#_Toc321405011)

[Hand Training 4](#_Toc321405012)

[Hill Climbing 4](#_Toc321405013)

[Results 4](#_Toc321405014)

[Fitness Function 4](#_Toc321405015)

[Random Weights 5](#_Toc321405016)

[Hand-Trained Weights 5](#_Toc321405017)

[Algorithm-Trained Weights 5](#_Toc321405018)

[Success Rate 5](#_Toc321405019)

[Behavior Analysis 5](#_Toc321405020)

[Conclusion 7](#_Toc321405021)

[Bibliography 7](#_Toc321405022)

[Code Appendix 7](#_Toc321405023)

# Abstract

This project’s goal was to use an Artificial Neural Network (ANN) to safely land a moonlander. ANN’s can be used in problems of classifying and mapping inputs to outputs. In this project, an ANN was used to receive six floating point sensory inputs about the virtual environment and to output two floating point values: thrust and burn. The success rate of the ANN was 57%, but it is highly likely that the same ANN structure could perform significantly better with a better training algorithm. The training algorithm used was local hill climbing with random restarts.

# Algorithm

## Artificial Neural Network (ANN)

An ANN is made up of neurons and has many useful properties such as solving problems without a high degree of expert knowledge, being robust to input noise, being robust to network degradation, and being able to learn.

ANNs have their disadvantages as well. The ANN is popularly referred to as the tool to use when the problem is not understood. ANNs are opaque and harder to examine the internal logic of than other methods. It is difficult to examine whether an ANN performing correctly on test data will perform correctly on new data. If the ANN came up with general rules to solve the test data, it is likely to perform well on the new but similar data. However, if the ANN overfit the test data, it is likely to not perform well on the new but similar data.

Due to the opaqueness of the ANN, it is important to be familiar with what the ANN is capable of. Siegelmann and Sontag showed that a finite ANN with first-order inputs (no multiplication or higher order connections) can be Turing complete [1]. Common problem domains for ANNs include data regression, function approximation, data classification and recognition, data processing and filtering, and control systems.

The ANN used for the moonlander control in this project was a simple network: it was a feed-forward network with two hidden layers. Each hidden layer had 6 nodes. There was an additional biasing weight on each node in the network.

### Node

The artificial neurons used in this project’s network were simplified, first-order nodes. Each node had an array of weights to apply to the previous layer’s outputs. **aj,i** represented a node’s output and **wk,j,i** represented a weight: **i** indexed a node in the **j**th layer and **k** indexed a weight for the **i**th node in the **j**th layer. The node’s output then equaled the sigmoid function **g** of the sum.

Equation : Node Output

### Inputs

The six inputs to the ANN were height, X-Position, Y-Velocity, X-Velocity, wind, and fuel. Each time the moonlander code is run, the lander starts with random values for wind and Y-Velocity.

### Outputs

The two outputs from the ANN were burn and thrust. Burn was scaled from 0 to 20, and thrust was scaled from -2.5 to 2.5. These were educated guesses. I controlled the moonlander a few times and saw I never needed a burn above 10 and never needed a thrust above a magnitude of 1.0. To hopefully include the boundary conditions and extreme cases, I bumped these numbers up to 20 and 2.5. This is an example of inserting expert level knowledge: that means I explicitly put helpful, problem-specific information into the network. It is possible to let these scaled output values be variables as well and learn which values give the best performance.

## Training

The two types of training used were hand training and hill climbing.

### Hand Training

I first hand selected weights for the network. I chose weights that were simple and seemed like a good start to the problem. All weights (including biased weights) were set to 0 before inserting hand selected values. It turned out that the hand training was mostly worthless since it still crashed every time.

I set the weights so burn was tied to Y-Velocity. Then through a couple tests, I changed the scaling so that the moonlander fell more slowly.

I made similar weight changes for tying thrust to X-Velocity and wind and the moonlander stayed in the middle better than it previously did.

### Hill Climbing

For the hill climbing algorithm, the current ANN set of weights is checked against neighbor weights, and if the neighbor is better, it becomes the new current ANN. Neighbor ANNs were calculated by randomly changing one weight, performing a point mutation on the ANN. Table 1 shows the parameters for the hill climbing algorithm.

|  |  |
| --- | --- |
| **Hill Climbing Parameter** | **Value or Type** |
| Random Restarts | 10 |
| Neighbors to Check (Generations) | 10000 |
| Mutation Type | Point: new random value in legal range |

Table : Hill Climbing Parameters

# Results

## Fitness Function

In addition to the simple success condition of hitting the launch pad below a maximum speed, I included a fitness function to aid in evaluating the effectiveness of the ANN. The fitness function is shown in Equation 2. The best possible fitness is 0: landing exactly in the middle of the launch pad with no velocity. In order to give both attributes equal importance in contributing to fitness, they were scaled to the same range of 0 to 1 for success and above 1 for failure.

Equation : Fitness

The fitness function is very important for quantifying and comparing the effectiveness of the ANN’s performance. Due to fluctuations in wind and starting velocity, I took the average of 100 trials to determine the fitness. Without those changes in conditions, trials would have not helped as the rest of the moonlander simulation and ANN was entirely deterministic.

## Random Weights

Testing the fitness of random weights shows how large the state space is (and how much of it consists of very poor ANN weights). Comparing the randomly weighted ANN to the zero-weighted ANN, we see that the randomly weighted ANN is much worse in most cases. Keep in mind the maximum fitness to be considered a success is 2. Each of the two factors, landing speed and landing position, has to succeed, and the success threshold for each parameter is scaled to a value of 1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2666.94 | 7655.84 | 57.89 | 3134.64 | 1643.94 | 2822.81 | 1140.63 | 87.23 | 1096.1 | 3422.3 |

Table : Random Weights. Average fitness: 2372.83, All Crashes

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 94.2 | 93.28 | 88.24 | 87.44 | 90.68 | 84.25 | 83.89 | 94.93 | 93.99 | 91.44 |

Table : All Weights = 0. Average fitness: 90.23, All Crashes

## Hand-Trained Weights

Following the steps and reasoning outlined earlier in the Hand Training section, I set up the weights to propagate the input Y velocity to the output burn and the input X velocity to the output thrust.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 58.53 | 21.74 | 54.54 | 82.40 | 47.50 | 72.41 | 45.70 | 25.45 | 25.93 | 33.69 |

Table : Hand Weights. Average fitness: 46.79, All Crashes

The network is already doing better with a hand tweak, but overall, it is still performing terribly. None of the 10 trials successfully landed.

## Algorithm-Trained Weights

## Success Rate

The hill climbing algorithm produced an ANN that succeeded 57% of the time on new environments (57 out of 100 trials). The ANN was considered successful if it safely landed the moonlander in a new environment (new wind and initial velocity). This stark difference in success rates found throughout random restarts of the hill climbing algorithm suggests that the fitness landscape has many local optima. In order to get better ANN’s, a better training algorithm is needed.

## Behavior Analysis

The resulting behavior of the ANN could be summed up as “keep the moonlander in the center and the velocity below the maximum landing velocity at all times”.

For keeping the moonlander in the center, the ANN performed well. The X-Position oscillated and converged with respect to height. There was no evidence of the ANN going outside of this strategy, such as swinging in against the wind for a final landing. In Figure 1, the X-Position may look like it is oscillating wildly, but all three shown trials are within the legal -0.2 to +0.2 landing pad range for the entire simulation.

Figure : Height vs X-Position, Crash Thresholds = [-0.2, +0.2]

In Figure 2: Height vs Falling Speed, only two of the three cases succeeded. It was odd how the moonlander sped up to just under the maximum safe landing speed. I am not sure why this behavior arose. There was not an explicit fitness penalty for taking longer to land. It is likely that strategies that fell slowly ran out of fuel to correct for the wind and ending up crashing the moonlander.

The other behavior present was the slowing down to a falling speed of 4, the maximum safe landing speed. In this example, the ANN did not successfully control one of the moonlanders: although its slowing down behavior was correct, it did not slow down enough for a safe landing.

Figure : Height vs Falling Speed, Crash Threshold = 4.0

## Advantages and Disadvantages

The ANN’s strength was that it figured out how to control the moonlander through training. There was no human understanding of the problem other than selecting which inputs to feed into the network.

The ANN’s weakness was its low probability of success, 57%. From looking at the code for legal starting conditions, all conditions were solvable. The weakness was not necessarily in the ANN but in the training method. As shown in Figure 3, the algorithm improves its fitness quickly in the beginning, but convergence slows drastically after initial gains. As shown in Figure 4, there is no improved solution for about 9000 generations. The fact that it improved quickly several times after that indicates that a much better solution exists.

|  |  |
| --- | --- |
|  |  |
| Figure : Starting Fitness | Figure : Ending Fitness |

# Conclusion

The ANN showed that it was capable of safely landing a moonlander 57% of the time, not an impressive figure. However, it performed significantly better than the 0% successful randomly weighted ANN. The next step to take for the full project 3 is using a different training method. Hill climbing is clearly getting stuck in local optima, and I would think the ANN as it currently stands could achieve 100% success with a good enough set of weights. Experimenting with different training methods will vastly improve the ANN’s success.

# Bibliography

|  |  |
| --- | --- |
| [1] | H. Siegelmann and E. Sontag, "Turing computability with neural nets," *Appl. Math. Lett. 4 (6),* pp. 77-80, 1991. |

# Code Appendix

The code for the project is hosted at http://max-m.googlecode.com/svn/trunk/Lunar%20Lander/