Training an Artificial Neural Network to solve Ballistics Problems

Nathan Singleton

CNU Advisor: Dr Heddle

Abstract:

For my capstone project I built a neural network, neural net for short, using the framework Neuroph Studio and trained it to solve different ballistics problems. Initially it was trained on problems without air resistance, but eventually linear friction was added to see how well an artificial neural network (ANN) could handle these problems. It was found that the neural net was indeed capable of handling this type of problem although depending on the structure of the neural net there were varying levels of accuracy. There was no noticeable difference in accuracy or time to train between the vacuum case and the linear drag case.

Introduction:

This project was intended to answer the question of how well a neural net can solve the class of problems known as 2D ballistics or projectile-motion problems. It was expected that a neural net should be able to solve a projectile problem pretty easily since neural nets are good at pattern recognition, or function approximation in this case. There are many different types of neural net but the one used for this project was a multi-layer perceptron trained using backpropagation.

The framework chosen for this project was Neuroph Studio. This framework is widely used in physics laboratories, including Jefferson Lab, where it is used for fast function approximation. In Jefferson Lab's Hall B, researchers are also evaluating Neuroph Studio for its utility developing an ANN to assist in reconstructing particle trajectories.

Neuroph Studio is an integrated development environment (IDE) for developing neural networks of different architectures. It has a Java API and provides base classes which must be

extended by the developer for the specific problem of interest.

Theory:

An artificial neural network is a computational system modelled on a biological neural net, for example a human brain. They are made up of layers and each layer contains some number of neurons. Neurons are little functions that some activation function. Each neuron in a layer is connected to the neurons in the previous and next layers by weighted connections. They take in the outputs of the previous layer, and sum them up based on the connection weights. Then the sum is put through the activation function before being passed to the next layer. Neural nets are used because they are good at computing approximate functions that depend on a large number of generally unknown variables. Neural nets are specified using architecture, activity rule, and learning rule. Neural net architecture describes what variables are involved and what their relationship is (eg. weights for neuron connections). The Activity Rule tells how neurons actions change in response to other neurons. The Learning Rule describes the way in which the neural network's activities change over time.

Neural Networks are particularly useful in certain situations because they can be used to infer functions from observations. In this particular case, by showing that a neural net can easily approximate a function for the range of a particle in a vacuum as well as a particle subject to air resistance. I am using Neural Nets because they are good at pattern recognition.

The neural net was trained using backpropagation. This is an algorithm that calculates the gradient of a loss function with respect to the weights of the neurons in the neural net. Using the loss function, backpropagation calculates an error value for each neuron in the output layer, which for the ballistics problems contained only one neuron. Then this error is

propagated backwards through the layers until each neuron in the ANN has an associated error which shows its approximate contribution to the final output. Theses errors are used to calculate the loss function's gradient. Then an optimization method using this gradient updates the weights of each neuron by trying to minimize the loss function. By minimizing error, it can approximate the solution curve and hopefully answer, with the desired accuracy it was trained to, new ballistics problems it is given.

Methods:

Using Neuroph Studio, we constructed and trained the neural network. The projectile motion problems we investigated have either closed form solutions or good approximate solutions. This allowed us to accumulate test data for training the ANN. The data were created in Microsoft Excel, and stored in the standard, human-readable csv format. Using this data we trained the network using backpropagation. That is, during training the neural net was given inputs to which the answer is known and told to run. It then compared its outputs to the target (known) value and adjusted the weights so that next time it ran its output was closer to the expected answer. As it trains the neural net, Neuroph Studio outputs a graph of the total mean square error of the output compared to the expected value. This graph should take the form of an exponential decay if the neural net is properly converging. The neural net was designed with two input neurons and one output neuron. This means it takes in two inputs and returns one output. We used velocity and initial flight angle (with respect to the horizontal axis) as inputs, and the range the projectile traveled as an output. In the case of a projectile in a vacuum we used the equation,

$$R_{vacuum} = \frac{V^2 sin(2\theta)}{g}$$

to find the target range that the particle should travel. The neural net is given this value during training to compare to so that it can minimize its MSE.

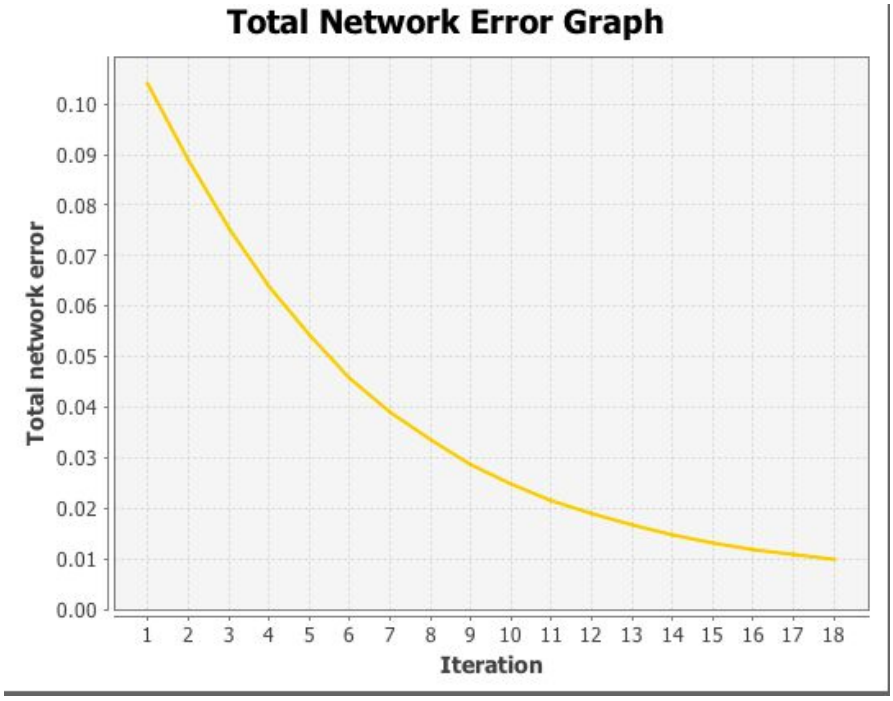
In the case of linear drag, we did not use the drag coefficient as an input, we used it as a parameter. It was used to solve for the terminal velocity of an object so the neural net was basically trained only to find the range of a specific projectile with linear drag. The equation used in this case was,

$$R_{linear\ drag} = R_{vac}(1 - \frac{4V_y}{3V_{ter}}) .$$

This function is a good approximation of linear drag when the force of linear drag is relatively small, such as in the case of air resistance.

Data:

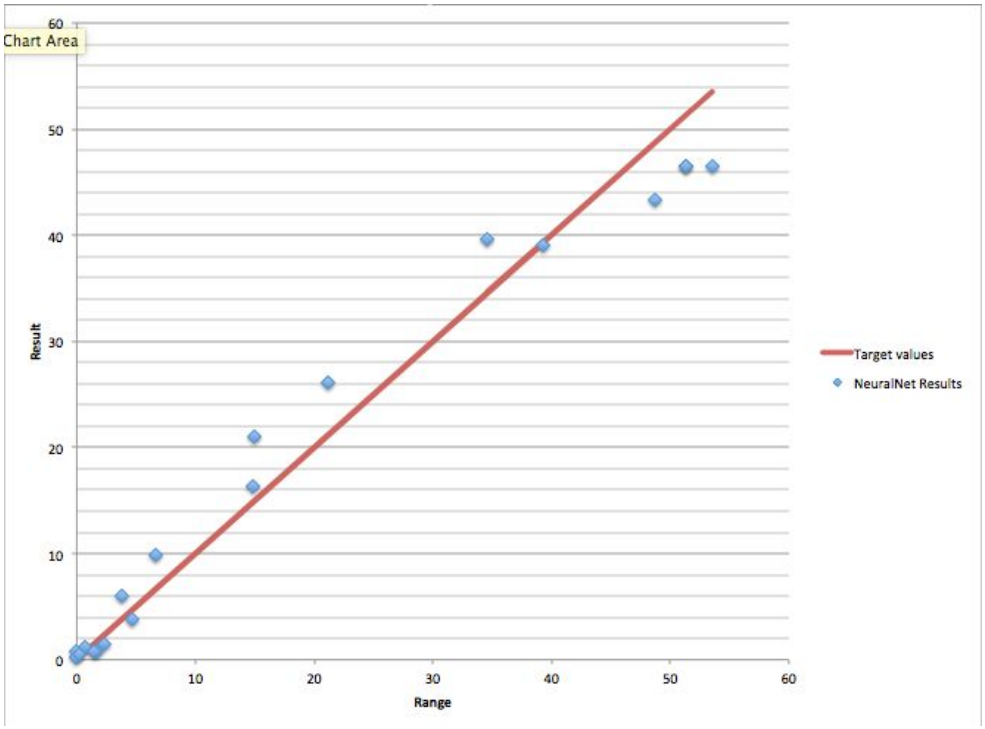
As the ANN trained it output a graph of the total network error per epoch of training. When it was successful, as shown below, the graph should take the form of an exponential decay.



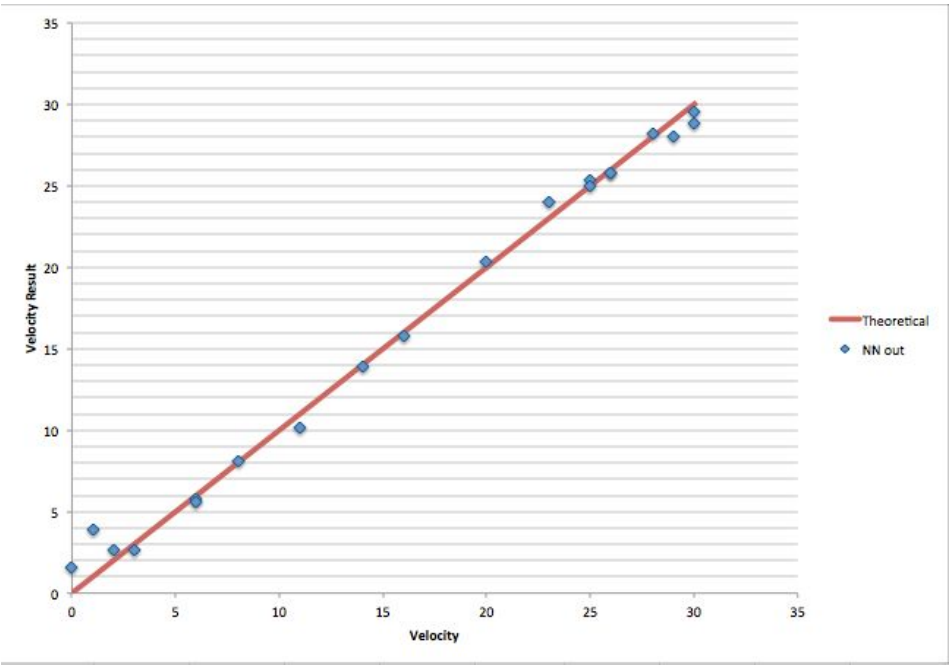
To enter data into the neural net, it first had to be normalized. Once this was done, the values could be given to the ANN and it would output a normalized value which then had to be unnormalized so that a comparison could be made. Below is the testing data for the neural net trained for range of motion in a vacuum.

Unnormalized						Normalized			
			.001 error						.001 error
Velocity	Angle	range	NN Output	difference		Velocity	Angle	Range	NN Output
6	19	2.26161358	1.52	0.74161358		0.2	0.63333333	0.02827017	0.019
11	4	1.71836176	0.72	0.99836176		0.36666667	0.13333333	0.02147952	0.009
3	23	0.66061818	1.2	0.53938182		0.1	0.76666667	0.00825773	0.015
25	3	6.66635608	9.92	3.25364392		0.83333333	0.1	0.08332945	0.124
16	27	21.1335052	26.08	4.94649484		0.53333333	0.9	0.26416881	0.326
6	12	1.49413461	0.72	0.77413461		0.2	0.4	0.01867668	0.009
26	24	51.2618267	46.56	4.70182674		0.86666667	0.8	0.64077283	0.582
0	25	0	0.8	0.8		0	0.83333333	0	0.01
28	21	53.5304485	46.48	7.05044851		0.93333333	0.7	0.66913061	0.581
20	29	34.614208	39.68	5.06579199		0.66666667	0.96666667	0.4326776	0.496
25	19	39.2641247	39.12	0.1441247		0.83333333	0.63333333	0.49080156	0.489
8	23	4.69772931	3.76	0.93772931		0.26666667	0.76666667	0.05872162	0.047
1	2	0.00711801	0.16	0.15288199		0.03333333	0.06666667	8.8975E-05	0.002
29	5	14.9018487	21.04	6.13815128		0.96666667	0.16666667	0.18627311	0.263
30	16	48.6660549	43.36	5.30605488		1	0.53333333	0.60832569	0.542
2	17	0.228242	0.56	0.331758		0.06666667	0.56666667	0.00285303	0.007
23	2	3.76542598	6	2.23457402		0.76666667	0.06666667	0.04706782	0.075
14	24	14.8628965	16.32	1.45710349		0.46666667	0.8	0.18578621	0.204
26	24	51.2618267	46.56	4.70182674		0.86666667	0.8	0.64077283	0.582
30	30	79.5329452	0						

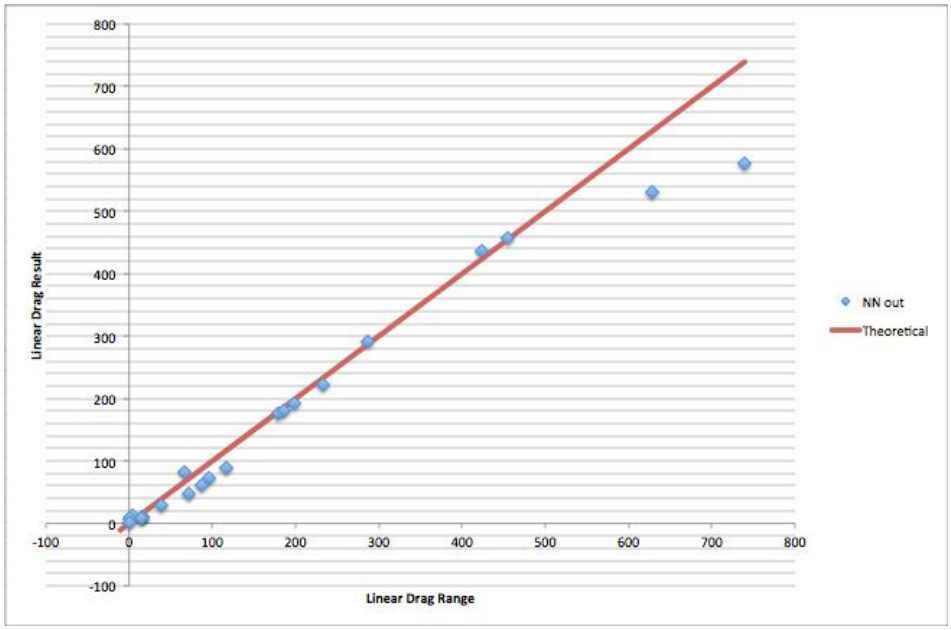
Using the outputs of the neural net once they were unnormalized, we took the outputted range and graphed it as a function of the expected range. By also graphing the expected range on the Y-axis versus itself, we get a line equivalent to y=x. Using the difference between this line and the points on the graph representing the outputs of the neural net, we can see approximately how well the neural net was able to find the range.



Another neural net of the same structure was trained by giving it the angle a projectile was fired and the range it went, and trained to find the velocity it was launched at. The neural net was able to output answers for this problem much more accurately, although it took significantly longer to train.



Finally, a third neural network, using the same structure as the previous two, was trained to find the range of a projectile subjected to linear air resistance. The projectile was assumed to be approximately the dimensions and mass of a beach ball. The linear drag coefficient and terminal velocity were used to calculate the range the projectile would go, but they were not given to the neural net. Only the initial velocity and angle, and range were given, so the neural net had to figure everything else out on its own.



As seen in the graph above, the neural net did a good job of approximating the range for smaller velocities, but as the range of the projectile increased it became less accurate. This may have been due to a bias in the training data towards the lower values. The data was randomly generated in Excel, so it may not have been evenly distributed across the whole range.

Discussion and Conclusions:

The neural net only took about one minute to train on a mid 2012 MacBook Pro, with a 2.9 GHz Intel Core i7 processor with 8GB RAM, to within a .01 total mean square error for ballistics without air resistance. It took approximately the same amount of time to train on data with linear drag. When a separate neural net was trained to output an initial velocity given the the initial angle fired and range of travel, it took about 3-5 minutes to train which is much longer than the other two. The average error of the outputs was .019 due to outliers having a higher error than values closer to the middle of the data range for the vacuum case. The greatest difference from target value was ~6 meters. In the case of linear drag the error was higher,

especially at larger range values, with the greatest difference being ~50 meters below the target range. In the case of the neural net finding the velocity, it was much more accurate with the largest difference being ~3 m/s.

Appendices:

- Neuroph Studio - a java neural net framework
- Excell - used to create datasets for the neural net to train and test on as well as formulating the graphs of the data

Bibliography

- Nielson, Micheal A. *Neural Networks and Deep Learning*. Determination Press. 2015. web.
- Pruneau, Claude. PHY 5200: Mechanical Phenomena. Physics and Astronomy Department, Wayne State University, December 2005.
- Soares, Fabio, and Souza, Alan. *Neural Network Programming with Java*. Birmingham-Mumbai. PACKT Publishing. 2016. pdf.
- open source. Neuroph Studio. SourceForge, September 2008. PC.