Project FANTOM

Flying, Autonomous, Neural Network, Training and Operational Machine

Anna Lucille Breck

A Neural Network approach to autonomous flying for the F-15E Strike Eagle

CSCI 4511W
University of Minnesota
Dan Challou, Ph.D.
December 2023

Abstract

1 Introduction

The McDonnell Douglas F-15E Strike Eagle was first introduced in 1988 as a long-range, high-speed bomber that comes equipped with electronic warfare systems [3]. It is capable of jamming signals, eliminating enemy fighters, and dropping payloads without the need for an escort. Due to its incredible capabilities, the F-15E became a crucial component of the Air Force's weaponry. However, the F-15E requires two airmen to operate: a pilot to maneuver the aircraft and a Weapons Systems Officer (WSO) to navigate, target, and drop the payload. This puts more lives at risk during missions, as if the F-15E were to be shot down, two lives would potentially be lost instead of one. To address this issue, an experiment is being conducted that aims to use a Neural Net to autonomously pilot an F-15E through a preset mission. This would eliminate the need for a pilot and WSO, therefore saving manpower and lives.

Neural Networks are a type of machine learning algorithm that imitates the human brain by replicating neurons and how they form connections. Due to these properties, Neural Networks are the best solution to replace airmen, as they replicate how a person thinks, adapts, and reacts. The structure of a Neural Network consists of layers of neurons that take in data through activation functions. These neurons are connected to neurons of other layers through weight matrices. The process of a Neural Network is intuitive and easy to follow. Data is given to a Neural Network in either vector or matrix form. Each data point follows a weighted path to neurons through matrix multiplication. Each entry of the resulting vector is passed through an activation function, which defines the value of the neuron. These vectors of neurons are layers, also known as hidden layers, as they are hidden in the middle of the network and are used to shape and connect the data to produce an output. The basic structure of a Neural Network is pictured in Figure 1.

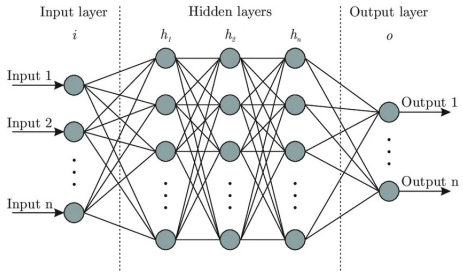

Figure 1: Neural Net Structure

2 Related Work

There have been several experiments implementing Neural Networks to self-driving mechanisms through out the many years since their development in the 1960s. The experiment I am conducting draws from two in particular,

2.1 DeepDriving

A research team at Princeton conducted an experiment where they were able to create and train a Convolutional Neural Network (ConvNet). The purpose of the ConvNet was to drive a racing simulator. The team used a behavior reflex approach to train the ConvNet. This approach involves creating a direct mapping from the sensory input to a series of avoidance indicators to make driving decisions using a caffe model. Caffe is a neural network structure made by Berkely that has several models made by a plethora of research teams. This experiment in particular used AlexNet, a well developed

2.2 Neural Network Fun With DCS

3 Approach

To obtain the most precise representation of an F-15E, I opted to run a test using the advanced simulator software known as DCS World. This platform is widely used during the initial training process in the F-15E B-course for pilots and WSOs, making it the most accurate publicly available simulation. Due

to its complexity and accuracy, DCS World has a select community of aircraft enthusiasts who possess extensive knowledge in all technical aspects of the aircraft. In light of this, the developers of DCS have incorporated numerous ways to customize missions and gather data, allowing real or replica aircraft control components to be used when flying the simulated aircraft. The Developers have also made customizing missions easy as well, allowing for action points, also known as waypoints to be set wherever the user desires. The location data of these way points (time to, position, action) can be returned through the integration system the DCS developers came up with.

3.1 Lua Integration

The developers created a file named "Export.lua" with ease of integration in mind. This file contains functions that enable transferring and inserting data via sockets. By connecting your own controllers and instruments to the simulator, you can enjoy a more realistic experience. Not only does this enable data collection to be sent to the Neural Net, but it also allows the Net to communicate back, which is beneficial for the experiment.

To start the process, the function *LuaExportStart()* opens the connection to the port where the Neural Net will receive and send the data. The function *LuaExportActivityNextEvent(t)* takes in a time variable t, collects data and sets it through the socket. The function works by first comparing the input time (the current sim time) to a global time variable that can be set, if the current time matches the global variable then the function proceeds with collecting data. The collected data includes the plane's current position in the DCS coordinate system (x, y, z), altitude, angle (pitch, roll, yaw), current airspeed, and current waypoint position. The global time variable is adjusted based on the distance of the current waypoint. If the plane is near the point, the time variable is adjusted slightly (0.5 seconds), while if it is far away, the adjustment is larger (10 seconds).

The function *LuaExportBeforeNextFrame()* checks for socket activity (commands sent from the Neural Network) and relays those commands to the simulator using the function *LoSetCommand(command, value)*. The input parameter command reads from a list of commands controlling each part of the aircraft. However, for this experiment, we are only concerned with includes joystick control, thrust control, weapon firing, and payload drop.

3.2 Neural Network Structure

As the aircraft's flight requires a large amount of data and its complexity cannot be addressed by a single Neural Net, I decided to break down the problem into five Neural Nets. The first Net receives the vector distances, y and x, from the aircraft's current position to the next waypoint to return the θ_p (pitch) and θ_y (yaw) values required to get the aircraft to that particular point. If the aircraft needs to climb, then the pilot must point the nose upwards, which results in a positive y-vector. Conversely, if the aircraft needs to descend, then the pilot must point the nose downwards, resulting in a negative y-vector. Since the F-15E does not have rudders like other aircraft, it relies solely on bank turns, rendering the yaw value from the first Neural Net useless. Therefore, the second Neural Net takes into account the yaw from the first, the current airspeed s, and the radial distance to the waypoint r, to calculate the bank angle necessary for the turn. Turning left or right is achieved by tilting the wings in the corresponding direction. For instance, to turn right, the left wing goes up, and to turn left, the right wing goes up. This is determined by looking at the yaw value; if it is positive, the left wing should go up, and if negative, the right wing should go up. According to the F-15E flight manual [1], the aircraft's altitude must be kept constant during the turn, which is handled by the third Neural Net. This Neural Net considers the current pitch, altitude, and altitude difference between the aircraft and the waypoint, allowing for the pitch to be corrected based on the current values. To ensure that the aircraft flies smoothly, the fourth component deals with thrust. This component takes into account the current airspeed, pitch and bank values, as well as the radial distance to the waypoint. It then calculates and returns the appropriate thrust value that should be set.

The last neural network can be a bit tricky to han-

dle, as it requires several inputs. These inputs include the current airspeed and position of the ego aircraft, as well as information on the locked target. If there is no locked target, this input will be labeled as "None." If there is a target, its position and velocity will be taken into account, with a velocity of 0 indicating a ground bombing target. Once all inputs are in place, the neural network will give either the fire command (if the velocity is not 0) or the drop command (if the velocity is 0). This ensures that the payload or firing will be done efficiently.

I used Neural Nets with two layers and 8 nodes per layer, since the data input was not extensive. For the first three layers, I chose the hyperbolic tangent function to enable both negative and positive values. For the fourth layer, I used the sigmoid function, and for the fifth layer, I used the step function.

3.3 Markov Process

Markov Process is a discrete probabilistic process that determines future states based on the current configurations known as Markov Chains. These chains are constructed of the vector of probabilities $\mathbf{u}^{(k)}$ demonstrating the k^{th} state vector and the transition matrix T whose entries show the transition probabilities. Together these values demonstrate the probability that the next state will occur in the form of [5]

$$\mathbf{u}^{(k+1)} = T\mathbf{u}^{(k)} \tag{1}$$

To achieve optimal outcomes for the initial weight matrices W_i, I utilized an approach that treats each weight matrix as a transition matrix and layers as probability vectors. This approach ensures that the values start out smaller and more polarized. The Markov Process requires the transition probabilities associated with each value of the vector to add up to 1 and remain positive. For the initial Neural Nets, I set each row to equal one and swapped the values to correspond to different tests. Essentially, every weight matrix will possess this property:

$$W_i = \begin{bmatrix} w_{0,0} & \dots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{n,0} & \dots & w_{n,n} \end{bmatrix}, w_{0,0} + \dots + w_{0,n} = 1 \tag{2}$$

I then can take these and swap them to ensure that every matrix is unique for the tests.

The output values of the Neural Nets are going to be trained to fit the commands for pitch (nose up/down), bank (wing position), and thrust, meaning that instead of going out as perfect angular values, they will come out as decimals between -1 and 1, corresponding to the amount the joystick/ thrust lever must be moved. The target Net will just either give the command or not give the command.

4 Experiment

In order to allow the neural network to fly the DCS simulator, it was necessary to calibrate the weights accurately. The testing process involved creating

multiple neural networks, having them fly in a unity simulator concurrently with preset waypoints, and then testing them using a genetic algorithm to identify the most successful Neural Nets. This procedure was repeated with newly generated Neural Nets until one was found that could successfully complete a mission.

4.1 The Unity Simulator

I used Unity Physics, which is a built-in physics program in Unity, and a 3D model of an F-15E to develop a method for testing multiple Neural Nets simultaneously. To begin with, I inserted preset waypoints and generated several F-15E models to match each batch test. Initially, the first generated Nets did not contain many successful aircraft, with most of them crashing. However, as each test progressed, the results got better and better through the Genetic algorithm testing.

4.2 Genetic Algorithms

Genetic algorithms are algorithms designed around the idea of natural selection, a population eliminating certain genes based on survival. In computer coding this idea centers around optimization, first, a population is set up containing a set of individuals. These individuals have characteristics referred to as

genomes, which can be mutated and altered to form

Algorithm 1 Fitness Function

```
for Neural Network in Population do
    Compare result to target
    if result is bad then
        Remove Neural Network from population
    end if
end for
Remaining Networks in the Population are then sent to generate the new populatation
```

new members. In line with natural selection the population is put through a test referred to as a fitness function, this function takes in a genome, values for testing, and a target value. The idea of the fitness functions I used for the testing process is given in the psuedocode, Algorithm 1. The Neural Networks are tested on time taken, crashes, physics, and target hits, the Networks that pass are set to the next phase for the generation of the new population.

During the Genetic algorithm process, the next phase is referred to as the crossover function. Its purpose is to splice together the top few individuals of the population to create the population for the next generation. In the experiment, this process is achieved by taking each weight matrix of the remaining population, identifying the minimum and maximum of each entry, and then creating new matrices based on that information. The **numpy.random** library is utilized specifically to carry out this task.

The final step in genetic algorithms is mutation. This process involves randomly altering the new genomes to ensure genetic diversity. To achieve this, I selected random values while integrating over the matrices, and transformed them into another decimal float value, also chosen at random. This helps to ensure that the new solutions can ensure new random solutions, just like in real like how some mutations like the lack of wisdom teeth can benefit the population as a whole.

After multiple iterations, a Neural Network was developed that could successfully complete a mission in DCS World.

5 Analysis

The successful Neural Net was able to complete several pretest missions upon testing and had a pretty decent efficiency. I was not measuring time in my experiment, but the simulation did run a bit slow, although that could be due to me running the Neural Net on the lab computers and the

5.1 Least Squares Error

6 Conclusion

References

[1] Manual for the f-15e for dcs world. *Eagle Dynamics*. URL https://www.digitalcombatsimulator.com/upload/DCS_ F-15E_ Flight_ Manual_ EN.pdf.

[2] A. K. C. Chen, A. Seff and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *IEEE International Conference on Computer Vision (ICCV)*, 2015. doi: http://dx.doi.org/10.1109/ICCV.2015.312.

[3] S. Davies. Be afraid of the dark. *Eagle Dynamics*, 2023. URL https://www.digitalcombatsimulator.com/Steve_ Davies_ Be_ Afraid_ of_ the_ Dark_ F-15E_ Book_ Sample.pdf.

[4] D. Knuth. Neural network fun with dcs (a dcs world forum post), December 28, 2019. URL https://forum.dcs.world/topic/221730-neural-network-fun-with-dcs/.

[5] P. Oliver. *Applied Linear Algebra (Second Edition)*. Springer International Publishing, Cham, Switzerland, 2010.